Higher-order bialgebraic semantics

SERGEY GONCHAROV
School of Computer Science, University of Birmingham, UK
(email: s.goncharov@bham.co.uk*)*

STEFAN MILIUS
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
(email: stefan.milius@fau.de*)*

LUTZ SCHRÖDER
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
(email: lutz.schroeder@fau.de*)*

STELIOS TSAMPAS
Centre for Formal Methods and Future Computing, Syddansk Universitet, Denmark
(email: stelios@imada.sdu.dk*)*

HENNING URBAT
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
(email: henning.urbat@fau.de*)*

Abstract

Compositionality proofs in higher-order languages are notoriously involved, and general semantic frameworks guaranteeing compositionality are hard to come by. In particular, Turi and Plotkin's bialgebraic abstract GSOS framework, which provides off-the-shelf compositionality results for first-order languages, so far does not apply to higher-order languages. In the present work, we develop a theory of abstract GSOS specifications for higher-order languages, in effect transferring the core principles of Turi and Plotkin's framework to a higher-order setting. In our theory, the operational semantics of higher-order languages is represented by certain dinatural transformations that we term *(pointed) higher-order GSOS laws*. We give a general compositionality result that applies to all systems specified in this way and discuss how compositionality of combinatory logics and the λ-calculus w.r.t. a strong variant of Abramsky's applicative bisimilarity are obtained as instances.

1 Introduction

The framework of *Mathematical Operational Semantics*, introduced by [56], elucidates the operational semantics of programming languages, and guarantees compositionality of programming language semantics in all cases that it covers. In this framework, operational semantics are presented as distributive laws, varying in complexity, of a monad over a comonad in a suitable category. An important example is that of *GSOS laws*, i.e. natural transformations of type

$$\varrho_X\colon \Sigma(X \times BX) \to B\Sigma^\star X,$$

with endofunctors $\Sigma, B\colon \mathbb{C} \to \mathbb{C}$ respectively specifying the *syntax* and *behaviour* of the system at hand. The idea is that a GSOS law represents a set of inductive transition rules that specify how programs are run. For instance, the choice of $\mathbb{C} = \mathbf{Set}$ and $B = (\mathscr{P}_\omega)^L$, where $\mathscr{P}_\omega$ is the finite powerset functor and L a set of transition labels, leads to the well-known

Journal of Functional Programming, Volume 36, Article 1 DOI: 10.46298/jfp.17738
Submitted April 2024; accepted March 2026

GSOS rule format by [10] for specifying labeled transition systems. For that reason, Turi and Plotkin's framework is often referred to as *abstract GSOS*.

The semantic interpretation of GSOS laws is conveniently presented in a bialgebraic setting (cf. Section 2.2). Every GSOS law ϱ (2.1) canonically induces a bialgebra

$$\Sigma(\mu\Sigma) \xrightarrow{\iota} \mu\Sigma \xrightarrow{\gamma} B(\mu\Sigma)$$

on the object $\mu\Sigma$ of programs freely generated by the syntax functor Σ, where the algebra structure ι inductively constructs programs and the coalgebra structure γ describes their one-step behaviour according to the given law ϱ. The above bialgebra is thus the *operational model* of ϱ. Dually, its *denotational model* is a bialgebra

$$\Sigma(\nu B) \xrightarrow{\alpha} \nu B \xrightarrow{\tau} B(\nu B),$$

which extends the final coalgebra νB (to be thought of as the domain of abstract program behaviours) of the behaviour functor B. Both the operational and the denotational model are characterized by universal properties, namely as the *initial ϱ-bialgebra* and the *final ϱ-bialgebra*, respectively. This immediately entails a key feature of abstract GSOS: The semantics is automatically *compositional*, in that behavioural equivalence (e.g. bisimilarity) of programs is a congruence with respect to the operations of the language. The bialgebraic framework has been used widely to establish further correspondences and obtain compositionality results, see e.g. the work of [8], [35], [18], [39], and [20].

As a first step towards extending the abstract GSOS framework to languages with *variable binding*, such as the π-calculus by [41] and the λ-calculus, [17] use the theory of *presheaves* to establish an abstract categorical foundation of syntax with variable binding, and develop a theory of capture-avoiding substitution in this abstract setting. Based on these foundations, the semantics of *first-order* languages with variable binding, more precisely that of the π-calculus and value-passing CCS, see [40], is formulated in terms of GSOS laws on categories of presheaves by [19]. We also introduce higher-order bialgebras and construct the initial such bialgebra.

However, the question of the mathematical operational semantics of the λ-calculus, or generally that of higher-order languages, still remains a well-known issue in the literature (see e.g. the introductory paragraph by [31]). Indeed, in order to give the semantics of a higher-order language in terms of some sort of a distributive law of a syntax functor over some choice of a behaviour functor, one needs to overcome a number of fundamental problems. For instance, for a generic set X of programs, the most obvious set of "higher-order behaviours over X" would be X^X, the set of functions that expect an input program in X and produce a new program in X. Of course, the assignment $X \mapsto X^X$ is not functorial in X but bifunctorial; more precisely, it yields a bifunctor

$$B(X,Y) = Y^X \colon \mathbf{Set}^{\mathsf{op}} \times \mathbf{Set} \to \mathbf{Set}$$

of mixed variance. Working with mixed variance bifunctors as a basis for higher-order behaviour makes the situation substantially more complex in comparison to Turi and Plotkin's original setting. In particular, natural transformations alone will no longer suffice as the technical basis of a framework involving mixed variance functors, and it is not a priori clear what the right notion of coalgebra for a mixed variance functor should be. In this paper, we address these issues, with a view to obtaining a general congruence result.

Contributions. We develop a theory of abstract GSOS for higher-order languages, extending Turi and Plotkin's original first-order framework. We model such languages abstractly in terms of syntax endofunctors of the form $\Sigma = V + \Sigma' \colon \mathbb{C} \to \mathbb{C}$ (for an endofunctor $\Sigma' \colon \mathbb{C} \to \mathbb{C}$ representing the constructors of the language and a choice of an object $V \in \mathbb{C}$ to be thought of as an object of variables), and behaviour bifunctors $B \colon \mathbb{C}^{\mathsf{op}} \times \mathbb{C} \to \mathbb{C}$. The key concept introduced in our paper is that of a *V-pointed higher-order GSOS law*: a family of morphisms

$$\varrho_{X,Y} \colon \Sigma(jX \times B(jX, Y)) \to B(jX, \Sigma^\star(jX + Y))$$

dinatural in $X \in V/\mathbb{C}$ and *natural* in $Y \in \mathbb{C}$, with $j \colon V/\mathbb{C} \to \mathbb{C}$ denoting the forgetful functor from the coslice category $V/\mathbb{C}$ to $\mathbb{C}$. Similar to the first-order case, we think of a higher-order GSOS law as encoding the set of inductive small-step operational rules of a higher-order language. We show that each V-pointed higher-order GSOS law inductively determines an operational model given by a *higher-order bialgebra*

$$\Sigma(\mu\Sigma) \xrightarrow{\iota} \mu\Sigma \xrightarrow{\gamma} B(\mu\Sigma, \mu\Sigma)$$

on the initial algebra $\mu\Sigma$ of program terms. In analogy to the first-order case, the operational model is the initial higher-order bialgebra for the given higher-order GSOS law.

From a coalgebraic standpoint, the morphism $\gamma \colon \mu\Sigma \to B(\mu\Sigma, \mu\Sigma)$ is a coalgebra for the restricted *endofunctor* $B(\mu\Sigma, -) \colon \mathbb{C} \to \mathbb{C}$. Our semantic domain of choice is the final $B(\mu\Sigma, -)$-coalgebra (Z, ζ), the object of abstract behaviours determined by the functor $B(\mu\Sigma, -)$. We obtain a morphism $\mathsf{coit}\,\gamma \colon \mu\Sigma \to Z$ by coinductively extending γ; that is, we take the unique coalgebra morphism into the final coalgebra:

$$\begin{tikzcd}
\mu\Sigma \arrow[r, "\gamma"] \arrow[d, dashed, "\mathsf{coit}\,\gamma"'] & B(\mu\Sigma, \mu\Sigma) \arrow[d, "B(\mu\Sigma, \mathsf{coit}\,\gamma)"] \\
Z \arrow[r, "\zeta"] & B(\mu\Sigma, Z)
\end{tikzcd}$$

Importantly, in contrast to first-order abstract GSOS, the final coalgebra (Z, ζ) generally does *not* extend to a final higher-order bialgebra; in fact, a final bialgebra does not usually seem to exist (Example 4.25). As a consequence, proving compositionality in our higher-order setting is substantially more challenging than in the first-order case, requiring entirely new techniques and additional assumptions on the base category $\mathbb{C}$ and the functors Σ and B. Specifically, we investigate higher-order GSOS laws in a *regular* category $\mathbb{C}$.

As our main compositionality result, we show that the kernel pair of $\mathsf{coit}(\gamma) \colon \mu\Sigma \to Z$ (which under mild conditions coincides with the coalgebraic bisimilarity relation) is a congruence. We demonstrate the expressiveness of higher-order abstract GSOS by modeling two important examples of higher-order systems. We draw our first example, the *SKI combinator calculus* by [12], from the world of combinatory logic, which we represent using a higher-order GSOS law on the category of sets. For our second example we move on to a category of presheaves, on which we model the call-by-name and the call-by-value λ-calculus. In all of these examples, we demonstrate that the induced semantics corresponds to strong variants of *applicative bisimilarity*, originally introduced by [3].

Organization. In Section 2 we provide a brief introduction to the core categorical concepts that are used throughout this paper. Moving on, in Section 3 we discuss examples from combinatory logic and present a basic rule format for higher-order languages that illustrates the principles behind our approach. Section 4 is where we define our key notion of pointed higher-order GSOS law and prove our main compositionality result (Theorem 4.14). In Section 5 we show how to implement the call-by-name and call-by-value λ-calculus in our abstract framework. We conclude the paper with a discussion of further developments and potential avenues for future work in Section 6.

Related work. Formal reasoning on higher-order languages is a long-standing research topic (e.g. [52], [3], [4]). The series of workshops on *Higher Order Operational Techniques in Semantics*, see [27], played an important role in establishing the so-called *operational methods* for higher-order reasoning. The two most important such methods are *logical relations* (see [54], [53], [43], [15]) and *Howe's method* (see [32, 33]), both of which remain in use to date. Other significant contributions towards reasoning on higher-order languages were made by Sangiorgi [48, 49] and Lassen [36]. While GSOS-style frameworks ensure compositionality for free by mere adherence to given rule formats, both logical relations and Howe's method instead have the character of robust but inherently complex methods whose instantiation requires considerable effort.

Recently, notable progress has been made towards generalizing Howe's method by [31] and [11], based on previous work on *familial monads* and operational semantics by [30]. According to the authors, their approach departs from Turi and Plotkin's bialgebraic framework exactly because that framework did not cover higher-order languages at the time. [13] give a general account of congruence proofs, and specifically Howe's method, for applicative bisimilarity for λ-calculi with algebraic effects, based on the theory of relators. [28] present a foundational account of logical relations as *structure-preserving* relations in a reflexive graph category.

Rule formats like the GSOS format by [10] have been very useful for guaranteeing congruence of bisimilarity at a high level of generality. However, rule formats for higher-order languages have been scarce. An important example is that of the *promoted tyft/tyxt* rule format, see [9] and [42], which has similarities to our presentation of combinatory logic in Section 3, but it is unclear whether or not it is an instance of a general, categorical format. The rule format of Howe [33] was presented in the context of Howe's method. A variant of Howe's format was recently developed by Hirschowitz and Lafont [31].

The present paper is an extended and fully revised version of our contribution to POPL 2023, see [21]. In comparison to the latter, we include additional details and explanations that had to be omitted for lack of space, in particular full proofs, which we hope will further illuminate our results.

Subsequent work following our POPL 2023 paper is discussed in Section 6.

2 Preliminaries

2.1 Category theory

We assume familiarity with basic notions from category theory such as limits and colimits, functors, natural transformations, and monads; see e.g. [37]. For the convenience of the reader, we review some terminology and notation used in the paper.

Products and coproducts. Given objects X_1, X_2 in a category $\mathbb{C}$, we write $X_1 \times X_2$ for their product, $\mathsf{fst}\colon X_1 \times X_2 \to X_1$ and $\mathsf{snd}\colon X_1 \times X_2 \to X_2$ for the projections, and $\langle f_1, f_2\rangle\colon Y \to X_1 \times X_2$ for the pairing of morphisms $f_i\colon Y \to X_i$, $i = 1, 2$. Dually, we write $X_1 + X_2$ for the coproduct, $\mathsf{inl}\colon X_1 \to X_1 + X_2$ and $\mathsf{inr}\colon X_2 \to X_1 + X_2$ for the injections, and

$[g_1, g_2]\colon X_1 + X_2 \to Y$ for the copairing of morphisms $g_i\colon X_i \to Y$, $i = 1, 2$. Moreover, we let $\nabla = [\mathsf{id}_X, \mathsf{id}_X]\colon X + X \to X$ denote the codiagonal.

Dinatural transformations. Given bifunctors $F, G\colon \mathbb{C}^{\mathsf{op}} \times \mathbb{C} \to \mathbb{D}$ of mixed variance, a *dinatural transformation* from F to G is a family of morphisms

$$\sigma_X\colon F(X,X) \to G(X,X) \qquad (X \in \mathbb{C})$$

such that for every morphism $f\colon X \to Y$ of $\mathbb{C}$, the hexagon below commutes.

$$\begin{tikzcd}
 & F(X,X) \arrow[r, "\sigma_X"] & G(X,X) \arrow[dr, "G(\mathsf{id},f)"] & \\
F(Y,X) \arrow[ur, "F(f,\mathsf{id})"] \arrow[dr, "F(\mathsf{id},f)"'] & & & G(X,Y) \\
 & F(Y,Y) \arrow[r, "\sigma_Y"] & G(Y,Y) \arrow[ur, "G(f,\mathsf{id})"'] &
\end{tikzcd}$$

Regular categories. A *regular epimorphism* is an epimorphism that is the coequaliser of some pair of parallel morphisms. A category is *regular* if (1) it has finite limits, (2) for every morphism $f\colon A \to B$, the kernel pair $p_1, p_2\colon E \to A$ of f has a coequalizer, and (3) regular epimorphisms are stable under pullback. In a regular category, every morphism $f\colon A \to B$ admits a factorization $A \xrightarrow{e} C \xrightarrow{m} B$ into a regular epimorphism e followed by a monomorphism m; specifically, e is the coequalizer of the kernel pair of f[1], and m is the unique factorizing morphism. Indeed, the main purpose of regular categories is to provide a notion of image factorization of morphisms that relates to kernels of morphisms in a similar way as in set theory and universal algebra. Examples of regular categories include the category **Set** of sets and functions, the category $\mathbf{Set}^{\mathbb{C}}$ of (covariant) presheaves on a small category $\mathbb{C}$ and natural transformations, more generally every elementary topos, and every category of algebras over a signature (see below). In all these cases, regular epimorphisms and monomorphisms are the (componentwise) surjective and injective morphisms, resp.

Algebras. Given an endofunctor Σ on a category $\mathbb{C}$, a Σ-*algebra* is a pair (A, a) of an object A (the *carrier* of the algebra) and a morphism $a\colon \Sigma A \to A$ (its *structure*). A *morphism* from (A, a) to a Σ-algebra (B, b) is a morphism $h\colon A \to B$ of $\mathbb{C}$ such that $h \cdot a = b \cdot \Sigma h$. (We denote composition of morphisms by '$\cdot$'.) Algebras for Σ and their morphisms form a category $\mathbf{Alg}(\Sigma)$, and an *initial* Σ-algebra is simply an initial object in that category. We denote the initial Σ-algebra's carrier $\mu\Sigma$ if it exists, and its structure by $\iota\colon \Sigma(\mu\Sigma) \to \mu\Sigma$. Moreover, we write

$$\mathsf{it}(a)\colon (\mu\Sigma, \iota) \to (A, a)$$

for the unique morphism from $\mu\Sigma$ to the algebra (A, a); we often drop parentheses and write $\mathsf{it}\, a$ for $\mathsf{it}(a)$.

More generally, a *free* Σ-*algebra* on an object X of $\mathbb{C}$ is a Σ-algebra $(\Sigma^\star X, \iota_X)$ together with a morphism $\eta_X\colon X \to \Sigma^\star X$ of $\mathbb{C}$ such that for every algebra (A, a) and every morphism $h\colon X \to A$ in $\mathbb{C}$, there exists a unique Σ-algebra morphism $h^\star\colon (\Sigma^\star X, \iota_X) \to (A, a)$ such that $h = h^\star \cdot \eta_X$. If free algebras exist on every object, then their formation induces a monad $\Sigma^\star\colon \mathbb{C} \to \mathbb{C}$, the *free monad* on Σ, see [7]. Every Σ-algebra (A, a) induces an Eilenberg–Moore algebra $\hat{a}\colon \Sigma^\star A \to A$, viz. $\hat{a} = \mathsf{id}_A^\star$ for the identity morphism $\mathsf{id}_A\colon A \to A$.

[1] The kernel pair of $f\colon A \to B$ is the pullback $p_1, p_2\colon K \to A$ of f along itself.

Example 2.1 (Algebras over a signature)**.** The most familiar example of functor algebras is that of algebras over a signature. An *algebraic signature* consists of a set Σ of operation symbols together with a map $\mathsf{ar}\colon \Sigma \to \mathbb{N}$ associating to every operation symbol f its *arity* $\mathsf{ar}(\mathsf{f})$. Symbols of arity 0 are called *constants*. Every signature Σ induces the polynomial set functor $X \mapsto \coprod_{\mathsf{f}\in\Sigma} X^{\mathsf{ar}(\mathsf{f})}$, which we denote by the same letter Σ. We identify an element $(x_1,\dots,x_{\mathsf{ar}(\mathsf{f})})$ of the summand $X^{\mathsf{ar}(\mathsf{f})}$ with the flat Σ-term $\mathsf{f}(x_1,\dots,x_{\mathsf{ar}(\mathsf{f})})$. An algebra for the functor Σ then is equivalently an algebra for the signature Σ, i.e. a set A equipped with an operation $\mathsf{f}^A\colon A^n \to A$ for every n-ary operation symbol $\mathsf{f} \in \Sigma$. Morphisms between Σ-algebras are maps respecting the algebraic structure.

Given a set X of variables, the free algebra $\Sigma^\star X$ is the Σ-algebra of terms generated by Σ with variables from X. In particular, the free algebra on the empty set is the initial algebra $\mu\Sigma$, and it is formed by all *closed terms* of the signature. For every Σ-algebra (A,a), the induced Eilenberg–Moore algebra $\hat{a}\colon \Sigma^\star A \to A$ is given by the map evaluating terms over A in the algebra.

A relation $\sim\, \subseteq A \times A$ on a Σ-algebra A is called a *congruence* if, for every n-ary operation symbol $\mathsf{f} \in \Sigma$ and all elements $a_i, b_i \in A$ $(i = 1,\dots,n)$,

$$a_i \sim b_i \quad (i = 1,\dots,n) \qquad \text{implies} \qquad \mathsf{f}^A(a_1,\dots,a_n) \sim \mathsf{f}^A(b_1,\dots,b_n).$$

Note that unlike other authors, we do not require congruences to be equivalence relations. An equivalence relation $\sim$ is a congruence if and only if it is the *kernel* of some morphism, i.e. there exists a morphism $h\colon A \to B$ to some Σ-algebra B such that

$$a \sim b \qquad \text{iff} \qquad h(a) = h(b).$$

Coalgebras. Dually to the notion of algebra, a *coalgebra* for an endofunctor B on $\mathbb{C}$ is a pair (C,c) of an object C (the *state space*) and a morphism $c\colon C \to BC$ (its *structure*). A *morphism* from (C,c) to a B-coalgebra (D,d) is a morphism $h\colon C \to D$ of $\mathbb{C}$ such that $Bh \cdot c = d \cdot h$. Coalgebras for B and their morphisms form a category $\mathsf{Coalg}(B)$, and a *final* B-coalgebra is a final object in that category. If it exists, we denote the corresponding state space and the structure as νB and $\tau\colon \nu B \to B(\nu B)$. Moreover, for each coalgebra (C,c) we denote the unique coalgebra morphism by

$$\mathsf{coit}(c)\colon (C,c) \to (\nu B, \tau).$$

Again, we often write $\mathsf{coit}\, c$ for $\mathsf{coit}(c)$. Informally, a coalgebra is a categorical abstraction of a state-based system. The final coalgebra νB is the domain of all possible abstract behaviours that B-coalgebras may expose, and $\mathsf{coit}(c)$ sends every state of a coalgebra to its abstract behaviour. Accordingly, given coalgebras (C,c) and (C',c') for a functor $B\colon \mathbf{Set} \to \mathbf{Set}$, two states $x \in C$ and $x' \in C'$ are *behaviourally equivalent*, denoted $x \equiv x'$, if they are identified in the final coalgebra:

$$x \equiv x' \iff \mathsf{coit}(c)(x) = \mathsf{coit}(c')(x)$$

Behavioural equivalence is closely related to the notion of bisimilarity. A *bisimulation* between coalgebras (C,c) and (C',c') is a relation $R \subseteq C \times C'$ that can be equipped with a coalgebra structure $r\colon R \to BR$ such that both projection maps $p\colon (R,r) \to (C,c)$ and $p'\colon (R,r) \to (C',c')$ are coalgebra morphisms. Two states $x \in C$ and $x' \in C'$ are *bisimilar*, denoted $x \sim x'$, if they are contained in some bisimulation. For functors B preserving weak pullbacks, bisimilarity coincides with behavioural equivalence, see [47]:

$$x \sim x' \iff x \equiv x'.$$

Example 2.2. A coalgebra $c\colon C \to (\mathcal{P}_\omega C)^L$ for the functor $BX = (\mathcal{P}_\omega X)^L$ on **Set**, where $\mathcal{P}_\omega$ is the finite power set functor and L is a fixed set of labels, is precisely a labeled transition system (LTS). Given LTS (C, c) and (C', c'), a relation $R \subseteq C \times C'$ is a bisimulation if it preserves and reflects transitions, i.e. for each pair $R(x, x')$ and each label $a \in L$,

- if $x \xrightarrow{a} y$ then there exists $y' \in C'$ such that $x' \xrightarrow{a} y'$ and $R(y, y')$;
- if $x' \xrightarrow{a} y'$ then there exists $y \in C$ such that $x \xrightarrow{a} y$ and $R(y, y')$.

Since the functor B preserves weak pullbacks, bisimilarity is behavioural equivalence.

2.2 Abstract GSOS

In the following we briefly review the categorical *abstract GSOS* framework by [56] for modeling the operational semantics of (first-order) languages. We will refer to it as *first-order abstract GSOS* for distinction with the higher-order extension developed in Section 4. The framework is parametric in two endofunctors $\Sigma, B\colon \mathbb{C} \to \mathbb{C}$ on a category $\mathbb{C}$ with binary products, where Σ is assumed to have an initial algebra $\mu\Sigma$ and to generate a free monad $\Sigma^\star$. The functors Σ and B represent the *syntax* and *behaviour* of a language; in particular, $\mu\Sigma$ is regarded as the algebra of closed program terms. In abstract GSOS, the (small-step) operational semantics of a language is specified by a *GSOS law of* Σ *over* B, viz. a natural transformation

$$\varrho_X\colon \Sigma(X \times BX) \to B\Sigma^\star X \quad (X \in \mathbb{C}). \tag{2.1}$$

Informally, ϱ encodes the operational rules of the underlying language: for every constructor f of the language, it specifies the one-step-behaviour of programs $\mathsf{f}(-, \cdots, -)$, i.e. the Σ-terms they transition into next, depending on the one-step behaviours of all the operands.

Example 2.3. Consider a process algebra with a parallel composition operator $(p, q) \mapsto p \parallel q$ specified by the operational rules

$$\frac{p \xrightarrow{a} p'}{p \parallel q \xrightarrow{a} p' \parallel q} \qquad\qquad \frac{q \xrightarrow{a} q'}{p \parallel q \xrightarrow{a} p \parallel q'} \tag{2.2}$$

where a ranges over a fixed set L of action labels. To model this specification in abstract GSOS, one takes the polynomial functor $\Sigma X = X \times X$ corresponding to the binary operator $\parallel$, and the behaviour functor $BX = (\mathcal{P}_\omega X)^L$ representing labeled transition systems. The rules (2.2) induce a GSOS law of Σ over B, i.e. a natural transformation

$$\varrho_X\colon X \times (\mathcal{P}_\omega X)^L \times X \times (\mathcal{P}_\omega X)^L \to (\mathcal{P}_\omega(\Sigma^\star X))^L \qquad (X \in \mathbf{Set}),$$

whose components simply encode the two rules into a function:

$$\varrho_X(p, f, q, g) = \lambda a. \{p' \parallel q : p' \in f(a)\} \cup \{p \parallel q' : q' \in g(a)\}.$$

[56] observed more generally that GSOS laws of polynomial functors Σ over the functor $BX = (\mathcal{P}_\omega X)^L$ correspond to specifications in the *GSOS rule format* by [10]. The rules (2.2) for parallel composition are instances of GSOS rules.

Every GSOS law ϱ canonically induces an operational and a denotational model, which both form *bialgebras* for the given law ϱ. Formally, a ϱ-*bialgebra* (X, a, c) consists of an object $X \in \mathbb{C}$, a Σ-algebra $a\colon \Sigma X \to X$ and a B-coalgebra $c\colon X \to BX$ such that the left-hand diagram below commutes. A *morphism* from (X, a, c) to a ϱ-bialgebra (X', a', c') is a $\mathbb{C}$-morphism $h\colon X \to X'$ that is both a Σ-algebra morphism and a B-coalgebra morphism, i.e. the right-hand diagram commutes:

$$\begin{tikzcd} \Sigma X \arrow[r, "a"] \arrow[d, "\Sigma\langle \mathsf{id}, c\rangle"'] & X \arrow[r, "c"] & BX \\ \Sigma(X \times BX) \arrow[rr, "\varrho_X"] & & B\Sigma^\star X \arrow[u, "B\hat{a}"'] \end{tikzcd} \qquad \begin{tikzcd} \Sigma X \arrow[r, "a"] \arrow[d, "\Sigma h"'] & X \arrow[r, "c"] \arrow[d, "h"] & BX \arrow[d, "Bh"] \\ \Sigma X' \arrow[r, "a'"] & X' \arrow[r, "c'"] & BX' \end{tikzcd}$$

We think of a ϱ-bialgebra (X, a, c) as a *model* of the law ϱ: the algebra $a\colon \Sigma X \to X$ interprets the operations of the language, and the coalgebra $c\colon X \to BX$ is a transition system whose transitions are given by the operational rules specified by ϱ.

The universal property of the initial algebra $(\mu\Sigma, \iota)$ entails that there exists a unique B-coalgebra structure $\gamma\colon \mu\Sigma \to B(\mu\Sigma)$ such that $(\mu\Sigma, \iota, \gamma)$ is a ϱ-bialgebra. This is the initial ϱ-bialgebra, i.e. the initial object in the category of bialgebras and their morphisms. It is called the *operational model* of ϱ, as it is thought of as the transition system on terms specified by the rules corresponding to ϱ.

Dually, if B has a final coalgebra $(\nu B, \tau)$, it uniquely extends to a ϱ-bialgebra $(\nu B, \alpha, \tau)$. This is the final ϱ-bialgebra, and it is called the *denotational model* of ϱ. We think of νB as the object of abstract behaviours of systems of type B, and of the denotational model as interpreting the operations of the language at the level of abstract behaviours.

It follows that, both by initiality and finality, there exists a unique bialgebra morphism beh_ϱ from the operational model $(\mu\Sigma, \iota, \gamma)$ to the denotational model $(\nu B, \alpha, \tau)$:

$$\begin{tikzcd} \Sigma(\mu\Sigma) \arrow[r, "\iota"] \arrow[d, "\Sigma\mathsf{beh}_\varrho"'] & \mu\Sigma \arrow[r, "\gamma"] \arrow[d, dashed, "\mathsf{beh}_\varrho"] & B(\mu\Sigma) \arrow[d, "B\mathsf{beh}_\varrho"] \\ \Sigma(\nu B) \arrow[r, "\alpha"] & \nu B \arrow[r, "\tau"] & B(\nu B) \end{tikzcd}$$

The map beh_ϱ gives the denotational semantics of the language: it assigns to each program in $\mu\Sigma$ its abstract behaviour in νB.

For $\mathbb{C} = \mathbf{Set}$ and a polynomial functor Σ, the fact that beh_ϱ is a Σ-algebra morphism immediately implies that *behavioural equivalence*, namely the relation $\equiv$ on $\mu\Sigma$ given by

$$p \equiv q \quad\iff\quad \mathsf{beh}_\varrho(p) = \mathsf{beh}_\varrho(q),$$

is a congruence on the initial algebra $\mu\Sigma$. This means that the behaviour of a program term $\mathsf{f}(p_1, \ldots, p_n)$ is completely determined by the behaviour of its subterms $p_1, \ldots, p_n$; in other words, the denotational semantics induced by ϱ is *compositional*.

3 Combinatory logic

Our goal in the sequel is to extend Turi and Plotkin's abstract GSOS in a way that *higher-order* languages, such as the λ-calculus, can be modeled and reasoned about in the abstract categorical framework. We ease the reader into our theory by first considering a combinatory logic, the *SKI calculus*, originally introduced by [12]. It forms a computationally complete fragment of the untyped λ-calculus that does not feature variables, which for now allows us to bypass the technical intricacies arising from binding and substitution; these are treated later in Section 5. Specifically, we work with a variant of the SKI calculus featuring auxiliary operators, first introduced by [14]. We refer to this variant as *extended* combinatory logic, or

xCL for short. It is as expressive as the standard calculus but allows for a simpler coalgebraic presentation.

3.1 Extended combinatory logic

The set Λ_{CL} of **xCL**-terms is generated by the grammar

$$\Lambda_{\mathsf{CL}} ::= S \mid K \mid I \mid \mathsf{app}(\Lambda_{\mathsf{CL}}, \Lambda_{\mathsf{CL}}) \mid S'(\Lambda_{\mathsf{CL}}) \mid K'(\Lambda_{\mathsf{CL}}) \mid S''(\Lambda_{\mathsf{CL}}, \Lambda_{\mathsf{CL}}).$$

The binary operation app corresponds to function application; we usually write $s\,t$ for $\mathsf{app}(s,t)$. The standard combinators (constants) S, K and I represent the λ-terms

$$S = \lambda x.\,\lambda y.\,\lambda z.\,(x\,z)\,(y\,z), \qquad K = \lambda x.\,\lambda y.\,x, \qquad I = \lambda x.\,x.$$

The unary operators S' and K' capture application of S and K, respectively, to one argument: $S'(t)$ behaves like $S\,t$, and $K'(t)$ behaves like $K\,t$. Finally, the binary operator S'' is meant to capture application of S to two arguments: $S''(s,t)$ behaves like $(S\,s)\,t$. In this way, the behaviour of each combinator can be described in terms of *unary* higher-order functions; for example, the behaviour of S is that of a function taking a term t to $S'(t)$.

The small-step operational semantics of **xCL** is given by the rules displayed in Figure 1, where p, p', q, t range over terms in Λ_{CL}. The rules determine a labeled transition system

$$\to\ \subseteq \Lambda_{\mathsf{CL}} \times (\Lambda_{\mathsf{CL}} + \{\_\}) \times \Lambda_{\mathsf{CL}} \tag{3.1}$$

by induction on the structure of terms in Λ_{CL}, with $\{\_\}$ denoting the lack of a transition label. In this instance the set Λ_{CL} of labels coincides with the state space of the transition system. Note that every $t \in \Lambda_{\mathsf{CL}}$ either admits a single unlabeled transition $t \to t'$ or a family of labeled transitions $(t \xrightarrow{s} t_s)_{s\in\Lambda_{\mathsf{CL}}}$; thus, the transition system is deterministic. The intention is that unlabeled transitions correspond to *reductions* (i.e. computation steps) and that labeled transitions represent *higher-order behaviour*: a transition $t \xrightarrow{s} t_s$ means that the program t acts as a function that outputs t_s on input s, where s is itself a program term. An important observation towards specifying an abstract format in Section 3.2 is that labeled transitions are uniformly defined for every input $s \in \Lambda_{\mathsf{CL}}$, in that operators do not inspect the structure of s.

Remark 3.1. The incorporation of the auxiliary operators S', S'' and K' leads to a more consistent semantics, where all combinators accept exactly one argument. In addition, this allows us to bypass the issue of a program equivalence potentially telling combinators apart based on the number of their input arguments. Interestingly, [14] also introduce the auxiliary operators S', S'' and K' to avoid precisely this issue (see §5.2 in *op. cit.*). The added operators do not alter the functional behaviour of programs compared to the standard SKI calculus, except for adding more unlabeled transitions. For example, the conventional rule $S\,t\,s\,e \to (t\,e)\,(s\,e)$ for the S-combinator (see e.g. [29]) is rendered as the following sequence of transitions in **xCL**:

$$S\,t\,s\,e \to S'(t)\,s\,e \to S''(t,s)\,e \to (t\,e)\,(s\,e).$$

The first transition exists because (1) $S \xrightarrow{t} S'(t)$ by the rule for S; (2) therefore $S\,t \to S'(t)$ by the second rule of application; (3) therefore $S\,t\,s\,e \to S'(t)\,s\,e$ by the first rule of application. Similarly for the second and third transition.

Like every labeled transition system, Λ_{CL} comes with a notion of *(strong) bisimilarity*; see Example 2.2. Explicitly, a relation $R \subseteq \Lambda_{\mathsf{CL}} \times \Lambda_{\mathsf{CL}}$ is a *bisimulation* if, for all $p\,R\,q$,

$$\frac{}{S \xrightarrow{t} S'(t)} \qquad \frac{}{S'(p) \xrightarrow{t} S''(p,t)} \qquad \frac{}{S''(p,q) \xrightarrow{t} (p\,t)\,(q\,t)}$$

$$\frac{}{K \xrightarrow{t} K'(t)} \qquad \frac{}{K'(p) \xrightarrow{t} p} \qquad \frac{}{I \xrightarrow{t} t} \qquad \frac{p \to p'}{p\,q \to p'\,q} \qquad \frac{p \xrightarrow{q} p'}{p\,q \to p'}$$

Fig. 1. Operational semantics of the **xCL** calculus.

(1) $p \to p' \implies \exists q'.\, q \to q' \wedge p' \mathrel{R} q'$;
(2) $q \to q' \implies \exists p'.\, p \to p' \wedge p' \mathrel{R} q'$;
(3) $\forall t \in \Lambda_{\mathsf{CL}},\ p \xrightarrow{t} p' \implies \exists q'.\, q \xrightarrow{t} q' \wedge p' \mathrel{R} q'$;
(4) $\forall t \in \Lambda_{\mathsf{CL}},\ q \xrightarrow{t} q' \implies \exists p'.\, p \xrightarrow{t} p' \wedge p' \mathrel{R} q'$.

We write $\sim$ for the greatest bisimulation, viz. the union of all bisimulations (which is itself a bisimulation), and call two programs p and q *bisimilar* if $p \sim q$. The relation $\sim$ identifies programs that are indistinguishable in terms of computation steps and functional behaviour, in that they produce bisimilar terms given the same input.

Example 3.2. The terms $(S\,K)\,I$ and $(S\,K)\,K$ transition as follows:

$$(S\,K)\,I \to S'(K)\,I \to S''(K,I) \xrightarrow{t} (K\,t)\,(I\,t) \to K'(t)\,(I\,t) \to t,$$
$$(S\,K)\,K \to S'(K)\,K \to S''(K,K) \xrightarrow{t} (K\,t)\,(K\,t) \to K'(t)\,(K\,t) \to t.$$

It follows that $(S\,K)\,I \sim (S\,K)\,K$.

The set Λ_{CL} of **xCL**-terms forms the initial algebra for the algebraic signature

$$\Sigma = \{\, S/0, K/0, I/0, S'/1, K'/1, S''/2, \mathsf{app}/2 \,\},$$

with arities as indicated, and bisimilarity is respected by all operations of the language:

Proposition 3.3 (Compositionality of **xCL**)**.** *The bisimilarity relation $\sim$ is a congruence.*

Proof. In the following, by a *context* we mean a term $C \in \Sigma^\star\{\cdot\}$ in which the variable '$\cdot$' (the 'hole' of the context) appears at most once. We denote a context by $C[\cdot]$, and we write $C[p] = C[p/\cdot] \in \Lambda_{\mathsf{CL}}$ for the closed term obtained by substituting $p \in \Lambda_{\mathsf{CL}}$ for the hole in $C[\cdot]$. An equivalence relation $R \subseteq \Lambda_{\mathsf{CL}} \times \Lambda_{\mathsf{CL}}$ is a congruence if and only if the following relation is contained in R:

$$\hat{R} = \{(C[p], C[q]) \in \Lambda_{\mathsf{CL}} \times \Lambda_{\mathsf{CL}} \mid C[\cdot] \text{ is a context and } p \mathrel{R} q\}.$$

Thus, our task is to prove $\hat{\sim} \subseteq \sim$. To this end, it suffices to prove that $\hat{\sim}$ is a bisimulation up to transitive closure. This means that for every context $C[\cdot]$ and $p, q \in \Lambda_{\mathsf{CL}}$ such that $p \sim q$,

- either there exist $p', q' \in \Lambda_{\mathsf{CL}}$ such that $C[p] \to p'$, $C[q] \to q'$ and $p' \hat{\sim}^\star q'$,
- or for every $t \in \Lambda_{\mathsf{CL}}$, there exist $p', q' \in \Lambda_{\mathsf{CL}}$ such that $C[p] \xrightarrow{t} p'$, $C[q] \xrightarrow{t} q'$ and $p' \hat{\sim}^\star q'$,

where $\hat{\sim}^\star$ denotes the transitive closure of $\hat{\sim}$. This then implies that $\hat{\sim}^\star$ is a bisimulation. [2] Consequently we obtain $\hat{\sim} \subseteq \hat{\sim}^\star \subseteq \sim$ because $\sim$ is the greatest bisimulation.

[2] In other words, in **xCL**, *bisimulation up to transitive closure* is *sound* for bisimilarity. The proof is simple, and we omit details. For more information on up-to techniques, see e.g. [46, 50].

We proceed by structural induction on $C[\cdot]$. The cases where $C[\cdot]$ is not an application term are straightforward. For instance, for $C[\cdot] = S''(r, C'[\cdot])$, the property in question can be read from the diagram

$$\begin{tikzcd} S''(r, C'[p]) \arrow[d, "t"'] \arrow[r, phantom, "\mathrel{\widehat{\sim}}"] & S''(r, C'[q]) \arrow[d, "t"] \\ (r\,t)\,(C'[p]\,t) \arrow[r, phantom, "\mathrel{\widehat{\sim}}^{\star}"] & (r\,t)\,(C'[q]\,t) \end{tikzcd}$$

In fact, the two terms on the bottom are even related by $\mathrel{\widehat{\sim}}$. For application terms, we distinguish cases as follows.

- If $C[\cdot] = C'[\cdot]\,r$ and the transition of $C[p]$ comes from an unlabeled transition $C'[p] \to p'$, then the transition of $C[p]$ is $C'[p]\,r \to p'\,r$, and by induction, we have q' such that $C'[q] \to q'$ and $p' \mathrel{\widehat{\sim}}^{\star} q'$. This implies $C[q] = C'[q]\,r \to q'\,r$ and moreover $p'\,r \mathrel{\widehat{\sim}}^{\star} q'\,r$; for the latter we use that the relation $\mathrel{\widehat{\sim}}^{\star}$ is a congruence, which follows from the fact that $\mathrel{\widehat{\sim}}$ is a congruence by definition and transitive closures of congruences are congruences.
- If $C[\cdot] = C'[\cdot]\,r$ and the transition of $C[p]$ comes from a labeled transition $C'[p] \xrightarrow{r} p'$, then the transition of $C[p]$ is $C'[p]\,r \to p'$. By induction, we have q' such that $C'[q] \xrightarrow{r} q'$ and $p' \mathrel{\widehat{\sim}}^{\star} q'$, and then $C'[q]\,r \to q'$.
- If $C[\cdot] = r\,C'[\cdot]$ and the transition of $C[p]$ comes from an unlabeled transition $r \to r'$, then we have $r\,C'[p] \to r'\,C'[p]$ and $r\,C'[q] \to r'\,C'[q]$ which completes the case since $r'\,C'[p] \mathrel{\widehat{\sim}} r'\,C'[q]$, so $r'\,C'[p] \mathrel{\widehat{\sim}}^{\star} r'\,C'[q]$.
- Finally, suppose that $C[\cdot] = r\,C'[\cdot]$ and the transition of $C[p]$ comes from a labeled transition of r. According to the rules in Figure 1, for every $t \in \Lambda_{\mathsf{CL}}$ we have $r \xrightarrow{t} r'[t/x]$ for some term r' with one free variable x such that r' depends only on r but not on t. Hence, $r \xrightarrow{C'[p]} r'[C'[p]/x]$, $r\,C'[p] \to r'[C'[p]/x]$ and similarly $r\,C'[q] \to r'[C'[q]/x]$. Since $C'[p] \mathrel{\widehat{\sim}} C'[q]$, we conclude that $r'[C'[p]/x] \mathrel{\widehat{\sim}}^{\star} r'[C'[q]/x]$. (Note that the variable x can appear multiple times in r', so we generally do not have $r'[C'[p]/x] \mathrel{\widehat{\sim}} r'[C'[q]/x]$. This explains the need for the transitive closure $\mathrel{\widehat{\sim}}^{\star}$.) □

The proof of Proposition 3.3 is laborious, as it requires tedious case distinctions and a carefully chosen up-to technique, although the latter could have been avoided by working with multi-hole contexts. The present proof is also tailored to a specific language, one among many systems exhibiting higher-order behaviour. In the sequel, we describe an abstract, categorical representation of such higher-order systems that guarantees the compositionality of the semantics. In particular, we shall demonstrate that Proposition 3.3 emerges as an instance of a general compositionality result (Theorem 4.14).

3.2 A simple higher-order rule format

From a coalgebraic perspective, the deterministic labeled transition system (3.1) on **xCL**-terms forms a coalgebra

$$\gamma\colon \Lambda_{\mathsf{CL}} \to \Lambda_{\mathsf{CL}} + \Lambda_{\mathsf{CL}}^{\Lambda_{\mathsf{CL}}} \tag{3.2}$$

for the set functor $Y \mapsto Y + Y^{\Lambda_{\mathsf{CL}}}$, where the two summands of the codomain represent unlabeled and labeled transitions, respectively. Note that the coalgebra (3.2) can be regarded as an instance of an *applicative transition system* (see [3]) with β-reduction. We can abstract away from the set Λ_{CL} of labels and consider γ as a system of the form

$$Y \to Y + Y^X.$$

For higher-order systems such as $\gamma\colon \Lambda_{\mathsf{CL}} \to \Lambda_{\mathsf{CL}} + \Lambda_{\mathsf{CL}}^{\Lambda_{\mathsf{CL}}}$, we expect $X = Y$, underlining the fact that inputs come from the state space of the system. Note that the assignment

$$B(X,Y) = Y + Y^X\colon \mathbf{Set}^{\mathsf{op}} \times \mathbf{Set} \to \mathbf{Set} \tag{3.3}$$

gives rise to a *bifunctor* that is contravariant in X and covariant in Y. On the side of syntax, the signature of **xCL** yields the polynomial endofunctor $\Sigma\colon \mathbf{Set} \to \mathbf{Set}$ given by

$$\Sigma X = \coprod\nolimits_{\mathsf{f}\in\{S,K,I,S',K',S'',\mathsf{app}\}} X^{\mathsf{ar}(\mathsf{f})},$$

As a first step towards our higher-order abstract GSOS framework, we next introduce a simple concrete rule format for higher-order combinatory calculi that generalizes **xCL**.

Definition 3.4 ($\mathcal{HO}$ rule format)**.** Fix the countably infinite set

$$\mathcal{V} = \{\, x \,\} \;\cup\; \{\, x_i, y_i, y_i^z \;:\; i \in \{1,2,3,\dots\} \text{ and } z \in \{x, x_1, x_2, x_3, \dots\}\,\}.$$

of metavariables and an algebraic signature Σ.

(1) An $\mathcal{HO}$ *rule* for an operation symbol $\mathsf{f} \in \Sigma$ is an expression of the form

$$\frac{(x_j \to y_j)_{j\in W} \qquad (x_i \xrightarrow{z} y_i^z)_{i\in\overline{W},\, z\in\{x_1,\dots,x_n\}}}{\mathsf{f}(x_1,\dots,x_n) \to t} \tag{3.4}$$

or

$$\frac{(x_j \to y_j)_{j\in W} \qquad (x_i \xrightarrow{z} y_i^z)_{i\in\overline{W},\, z\in\{x,x_1,\dots,x_n\}}}{\mathsf{f}(x_1,\dots,x_n) \xrightarrow{x} t} \tag{3.5}$$

where $x, x_i, y_i, y_i^{x_j} \in \mathcal{V}$, $n = \mathsf{ar}(\mathsf{f})$, $W \subseteq \{1,\dots,n\}$, $\overline{W} = \{1,\dots,n\} \smallsetminus W$, and $t \in \Sigma^\star\mathcal{V}$ is a term depending only on the variables occurring in the premise; that is, in the rule (3.4) the term t can depend on the variables x_i ($i = 1,\dots,n$), y_j ($j \in W$), and $y_i^{x_j}$ ($i \in \overline{W}$, $j = 1,\dots,n$), and in (3.5) it can additionally depend on x and y_i^x ($i \in \overline{W}$).

(2) An $\mathcal{HO}$ *specification* for Σ is a set of $\mathcal{HO}$ rules such that for each n-ary $\mathsf{f} \in \Sigma$ and each $W \subseteq \{1,\dots,n\}$, there is exactly one rule of the form (3.4) or (3.5) in the set.

Intuitively, for every given rule the set $W \subseteq \{1,\dots,n\}$ determines which of the operands of $\mathsf{f}(x_1,\dots,x_n)$ perform a reduction and which exhibit higher-order behaviour, i.e. behave like functions. For $i \in \overline{W}$, the format dictates that said functions can be applied to a left-side variable x_j or the input label x, and then the output $x_i(x_j) = y_i^{x_j}$ or $x_i(x) = y_i^x$ can be used in the conclusion term t. The uniformity is apparent: rules cannot make any assumptions on the input label x or on other left-side variables that are used as arguments on the premises.

Example 3.5. The rules of **xCL** in Figure 1 form an $\mathcal{HO}$ specification modulo suitable renaming of variables and adding dummy premises. For illustration, let us consider the second rule for application. Using the variables $x_1, x_2, y_1^{x_2}$ instead of p, q, p', this rule can be rewritten as

$$\mathsf{app2}\,\frac{x_1 \xrightarrow{x_2} y_1^{x_2}}{x_1\, x_2 \to y_1^{x_2}}.$$

This is not yet an $\mathcal{HO}$ rule, since the latter require a complete list of premises. However, by filling in the missing premises in every possible way, $\mathsf{app2}$ is equivalent to the following two

$\mathcal{HO}$ rules. These rules correspond to y_2 being a reducing term or to y_2 computing a function, respectively, that is, to the choices $W = \emptyset$ and $W = \{2\}$ in (3.4):

$$\texttt{app2-a}\ \frac{x_1 \xrightarrow{x_1} y_1^{x_1} \quad x_1 \xrightarrow{x_2} y_1^{x_2} \quad x_2 \xrightarrow{x_1} y_2^{x_1} \quad x_2 \xrightarrow{x_2} y_2^{x_2}}{x_1\, x_2 \to y_1^{x_2}}$$

$$\texttt{app2-b}\ \frac{x_1 \xrightarrow{x_1} y_1^{x_1} \quad x_1 \xrightarrow{x_2} y_1^{x_2} \quad x_2 \to y_2}{x_1\, x_2 \to y_1^{x_2}}$$

Similarly, the combinator rule

$$\frac{}{S'(p) \xrightarrow{t} S''(p,t)}$$

is turned into the two rules

$$\frac{x_1 \to y_1}{S'(x_1) \xrightarrow{x} S''(x_1,x)} \qquad \frac{x_1 \xrightarrow{x_1} y_1^{x_1}}{S'(x_1) \xrightarrow{x} S''(x_1,x)}$$

by using the variables x_1, x instead of p, t and adding the required dummy premises for x_1.

Generalizing the case of **xCL**, every $\mathcal{HO}$ specification induces a $B(\mu\Sigma, -)$-coalgebra

$$\gamma\colon \mu\Sigma \to \mu\Sigma + \mu\Sigma^{\mu\Sigma} \tag{3.6}$$

carried by the initial algebra $\mu\Sigma$ of closed Σ-terms that runs program terms according to the given $\mathcal{HO}$ rules. Again, its bisimilarity relation is compatible with all language operations:

Proposition 3.6 (Compositionality of $\mathcal{HO}$)**.** *For every $\mathcal{HO}$ specification, the bisimilarity relation $\sim$ of the induced coalgebra $\gamma\colon \mu\Sigma \to B(\mu\Sigma, \mu\Sigma)$ is a congruence on $\mu\Sigma$.*

The proof is similar to that of Proposition 3.3; we omit it because it is subsumed by our general compositionality result for higher-order abstract GSOS (Theorem 4.14).

We are now prepared to make the key observation leading to our notion of *higher-order GSOS law* developed in Section 4. Recall from Section 2.2 that GSOS specifications are in bijective correspondence with GSOS laws of a polynomial functor Σ over $BX = (\mathcal{P}_\omega X)^L$, i.e. natural transformations of the form

$$\varrho_X\colon \Sigma(X \times (\mathscr{P}_\omega X)^L) \to (\mathscr{P}_\omega(\Sigma^\star X))^L \qquad (X \in \mathbf{Set}).$$

Similarly, $\mathcal{HO}$ specifications correspond to certain (di)natural transformations that distribute a polynomial functor Σ over the behaviour bifunctor $B(X,Y) = Y + Y^X$:

Theorem 3.7. *For every algebraic signature Σ, there is a bijective correspondence between $\mathcal{HO}$ specifications for Σ and families of maps*

$$\varrho_{X,Y}\colon \Sigma(X \times B(X,Y)) \to B(X, \Sigma^\star(X+Y)) \qquad (X, Y \in \mathbf{Set}) \tag{3.7}$$

dinatural in X and natural in Y.

Remark 3.8. Before turning to the proof, let us elaborate on the statement of the theorem and discuss the underlying intuitions.

(1) The (di)naturality conditions on the family $\varrho = (\varrho_{X,Y})_{X,Y\in\mathbf{Set}}$ assert that the hexagon

$$\begin{tikzcd}
& \Sigma(X\times B(X,Y)) \arrow[r, "\varrho_{X,Y}"] & B(X,\Sigma^\star(X+Y)) \arrow[dr, "B(\mathsf{id},\Sigma^\star(f+\mathsf{id}))"] & \\
\Sigma(X\times B(X',Y)) \arrow[ur, "\Sigma(\mathsf{id}\times B(f,\mathsf{id}))"] \arrow[dr, "\Sigma(f\times B(\mathsf{id},\mathsf{id}))"'] & & & B(X,\Sigma^\star(X'+Y)) \\
& \Sigma(X'\times B(X',Y)) \arrow[r, "\varrho_{X',Y}"] & B(X',\Sigma^\star(X'+Y)) \arrow[ur, "B(f,\Sigma^\star(\mathsf{id}+\mathsf{id}))"'] &
\end{tikzcd} \tag{3.8}$$

commutes for all sets X, X', Y and functions $f\colon X \to X'$, and moreover that the rectangle

$$\begin{tikzcd}
\Sigma(X\times B(X,Y)) \arrow[r, "\varrho_{X,Y}"] \arrow[d, "\Sigma(\mathsf{id}\times B(\mathsf{id},g))"'] & B(X,\Sigma^\star(X+Y)) \arrow[d, "B(\mathsf{id},\Sigma^\star(\mathsf{id}+g))"] \\
\Sigma(X\times B(X,Y')) \arrow[r, "\varrho_{X,Y'}"] & B(X,\Sigma^\star(X+Y'))
\end{tikzcd} \tag{3.9}$$

commutes for all sets X, Y, Y' and functions $g\colon Y \to Y'$.

(2) The need for *di*naturality comes from the mixed variance of the behaviour bifunctor B, which in turn is caused by the fact that variables are used both as states (covariantly) and as labels (contravariantly). The role of dinaturality is then the same as otherwise played by naturality: It ensures on an abstract level that the rules are parametrically polymorphic, that is, they do not inspect the structure of their arguments.

In more technical terms, (di)naturality enables the use of the Yoneda lemma to establish the bijective correspondence of Theorem 3.7. Explicitly, the bijection maps an $\mathcal{HO}$ specification $\mathcal{R}$ to the family (3.7) defined as follows. Given $X, Y \in \mathbf{Set}$ and

$$w = \mathsf{f}((u_1,v_1),\dots,(u_n,v_n)) \in \Sigma(X\times B(X,Y)),$$

consider the unique rule in $\mathcal{R}$ matching f and $W = \{j\in\{1,\dots,n\} : v_j\in Y\}$. If that rule is of the form (3.4), then

$$\varrho_{X,Y}(w) \in \Sigma^\star(X+Y) \subseteq B(X,\Sigma^\star(X+Y))$$

is the term obtained by taking the term t in the conclusion of (3.4) and applying the substitutions

$$x_i \mapsto u_i \ (i\in\{1,\dots,n\}), \quad y_j \mapsto v_j \ (j\in W), \quad y_i^{x_j} \mapsto v_i(u_j) \ (i\in\overline{W}, j\in\{1,\dots,n\}).$$

If the rule is of the form (3.5), then

$$\varrho_{X,Y}(w) \in \Sigma^\star(X+Y)^X \subseteq B(X,\Sigma^\star(X+Y))$$

is the map $u\mapsto t_u$, where the term t_u is obtained by taking the term t in the conclusion of (3.5) and applying the above substitutions along with

$$x \mapsto u \qquad \text{and} \qquad y_i^x \mapsto v_i(u) \quad (i\in\overline{W}).$$

(3) Thus, the intuition behind Theorem 3.7 is that ϱ encodes a set of $\mathcal{HO}$-rules into a (di-)natural, i.e. parametrically polymorphic, family of functions. The use of two sets X and Y in $\varrho_{X,Y}$ reflects that rules may have premises $x \xrightarrow{x'} y$ with two types of variables, namely variables $x, x' \in X$ that can appear both as inputs (covariantly) and labels (contravariantly), and variables $y\in Y$ that appear (covariantly) as outputs. The coproduct in $\Sigma^\star(X+Y)$ ensures that both types of variables can appear in output terms of rules.

Example 3.9. Let ϱ be the family corresponding to the $\mathcal{HO}$ specification of **xCL**, see Example 3.5. Given

$$w = (u_1, v_1)\,(u_2, v_2) = \mathsf{app}((u_1, v_1), (u_2, v_2)) \in \Sigma(X \times B(X, Y))$$

where $v_1 \in Y^X$, one has $\varrho_{X,Y}(w) = v_1(u_2)$, according to the rule `app2`.

Proof of Theorem 3.7. (1) Removing the syntactic sugar from Definition 3.4, we see that $\mathcal{HO}$ specifications for a signature Σ correspond bijectively to elements of the set

$$\prod_{\substack{\mathsf{f}\in\Sigma\\ W\subseteq \mathsf{ar}(\mathsf{f})}} \Big(\Sigma^\star\big(\mathsf{ar}(\mathsf{f}) + W + \mathsf{ar}(\mathsf{f}) \times \overline{W}\big) + \Sigma^\star\big(\mathsf{ar}(\mathsf{f}) + 1 + W + (\mathsf{ar}(\mathsf{f}) + 1) \times \overline{W}\big)\Big). \tag{3.10}$$

Here, we identify the natural number $\mathsf{ar}(\mathsf{f})$ with the set $\{1, \dots, \mathsf{ar}(\mathsf{f})\}$. Recall that a choice of $W \subseteq \{1, \dots .\mathsf{ar}(\mathsf{f})\}$ determines which of the operands of f perform a reduction (see Definition 3.4 and the explanations afterwards), and we let $\overline{W} = \mathsf{ar}(\mathsf{f}) \smallsetminus W$ denote the complement. Thus the summands under $\Sigma^\star$ spell out which variables may be used in the conclusion of the respective rule. For instance, the rule `app2-b` of Example 3.5 corresponds to the element

$$(2, 1) \in \mathsf{ar}(\mathsf{app}) \times \overline{W} \subseteq \Sigma^\star\big(\mathsf{ar}(\mathsf{app}) + W + \mathsf{ar}(\mathsf{app}) \times \overline{W}\big)$$

where the variable $y_1^{x_2}$ is identified with $(2, 1)$, and $\mathsf{ar}(\mathsf{app}) = 2$ is the arity of the application operator, and $W = \{2\}$.

We are thus left to prove that elements of the set (3.10) are in a bijective correspondence with (di)natural transformations of type (3.7).

(2) For functors $F, G\colon \mathbf{Set}^{\mathsf{op}} \times \mathbf{Set} \times \mathbf{Set} \to \mathbf{Set}$ we let $\mathsf{DiNat}_{X,Y}(F(X, X, Y), G(X, X, Y))$ denote the collection of all families of maps $\varrho_{X,Y}\colon F(X, X, Y) \to G(X, X, Y)$ dinatural in $X \in \mathbf{Set}$ and natural in $Y \in \mathbf{Set}$. Then we have the following chain of bijections:

$$\begin{aligned}
&\mathsf{DiNat}_{X,Y}\big(\Sigma(X \times B(X, Y)), B(X, \Sigma^\star(X + Y))\big)\\
&= \mathsf{DiNat}_{X,Y}\big(\textstyle\coprod_{\mathsf{f}\in\Sigma} (X \times B(X, Y))^{\mathsf{ar}(\mathsf{f})}, B(X, \Sigma^\star(X + Y))\big)\\
&\cong \textstyle\prod_{\mathsf{f}\in\Sigma} \mathsf{DiNat}_{X,Y}\big((X \times B(X, Y))^{\mathsf{ar}(\mathsf{f})}, B(X, \Sigma^\star(X + Y))\big)\\
&= \textstyle\prod_{\mathsf{f}\in\Sigma} \mathsf{DiNat}_{X,Y}\big((X \times (Y + Y^X))^{\mathsf{ar}(\mathsf{f})}, B(X, \Sigma^\star(X + Y))\big)\\
&\cong \textstyle\prod_{\mathsf{f}\in\Sigma} \mathsf{DiNat}_{X,Y}\big(X^{\mathsf{ar}(\mathsf{f})} \times (Y + Y^X)^{\mathsf{ar}(\mathsf{f})}, B(X, \Sigma^\star(X + Y))\big)\\
&\cong \textstyle\prod_{\mathsf{f}\in\Sigma} \mathsf{DiNat}_{X,Y}\big(X^{\mathsf{ar}(\mathsf{f})} \times \coprod_{W\subseteq\mathsf{ar}(\mathsf{f})} Y^W \times (Y^X)^{\overline{W}}, B(X, \Sigma^\star(X + Y))\big) && (*)\\
&\cong \textstyle\prod_{\mathsf{f}\in\Sigma} \mathsf{DiNat}_{X,Y}\big(X^{\mathsf{ar}(\mathsf{f})} \times \coprod_{W\subseteq\mathsf{ar}(\mathsf{f})} Y^W \times Y^{X\times\overline{W}}, B(X, \Sigma^\star(X + Y))\big) && (**)\\
&\cong \textstyle\prod_{\mathsf{f}\in\Sigma} \mathsf{DiNat}_{X,Y}\big(\coprod_{W\subseteq\mathsf{ar}(\mathsf{f})} X^{\mathsf{ar}(\mathsf{f})} \times Y^W \times Y^{X\times\overline{W}}, B(X, \Sigma^\star(X + Y))\big) && (*)\\
&\cong \textstyle\prod_{\mathsf{f}\in\Sigma} \prod_{W\subseteq\mathsf{ar}(\mathsf{f})} \mathsf{DiNat}_{X,Y}\big(X^{\mathsf{ar}(\mathsf{f})} \times Y^{W+X\times\overline{W}}, B(X, \Sigma^\star(X + Y))\big).
\end{aligned}$$

In the two steps marked $(*)$ we use that products distribute over coproducts in **Set**, and in the step marked $(**)$ we use that $(A^B)^C \cong A^{B\times C}$ for all $A, B, C \in \mathbf{Set}$. We claim that each factor of the last product satisfies

$$\begin{aligned}
&\mathsf{DiNat}_{X,Y}\big(X^{\mathsf{ar(f)}}\times Y^{W+X\times\overline{W}}, B(X,\Sigma^\star(X+Y))\big)\\
&\qquad\cong \mathsf{DiNat}_{X,Y}\big(X^{\mathsf{ar(f)}}\times Y^{W+X\times\overline{W}}, \Sigma^\star(X+Y)\big) + \\
&\qquad\quad\ \mathsf{DiNat}_{X,Y}\big(X^{\mathsf{ar(f)}}\times Y^{W+X\times\overline{W}}, (\Sigma^\star(X+Y))^X\big).
\end{aligned} \tag{3.11}$$

To see this, let ϱ be a family of maps in $\mathsf{DiNat}_{X,Y}\big(X^{\mathsf{ar(f)}}\times Y^{W+X\times\overline{W}}, B(X,\Sigma^\star(X+Y))\big)$. Consider the diagram below, where $!_X\colon X\to 1$ and $!_Y\colon Y\to 1$ are the unique maps into the singleton set 1. The upper part of the diagram commutes by naturality in Y and the lower part by dinaturality in X.

$$\begin{tikzcd}
X^{\mathsf{ar(f)}}\times Y^{W+X\times\overline{W}} \arrow[r, "\varrho_{X,Y}"] \arrow[d, "X^{\mathsf{ar(f)}}\times(!_Y)^{\mathsf{id}+X\times\mathsf{id}}"'] & \Sigma^\star(X+Y)+(\Sigma^\star(X+Y))^X \arrow[d, "\Sigma^\star(\mathsf{id}+!_Y)+(\Sigma^\star(\mathsf{id}+!_Y))^X"] \\
X^{\mathsf{ar(f)}}\times 1^{W+X\times\overline{W}} \arrow[r, "\varrho_{X,1}"] & \Sigma^\star(X+1)+(\Sigma^\star(X+1))^X \arrow[d, "\Sigma^\star(!_X+\mathsf{id})+(\Sigma^\star(!_X+\mathsf{id}))^X"] \\
X^{\mathsf{ar(f)}}\times 1^{W+1\times\overline{W}} \arrow[u, "\mathsf{id}\times 1^{\mathsf{id}+!_X\times\mathsf{id}}", "\cong"'] \arrow[d, "(!_X)^{\mathsf{ar(f)}}\times\mathsf{id}"'] & \Sigma^\star(1+1)+(\Sigma^\star(1+1))^X \\
1^{\mathsf{ar(f)}}\times 1^{W+1\times\overline{W}} \arrow[r, "\varrho_{1,1}"] & \Sigma^\star(1+1)+(\Sigma^\star(1+1))^1 \arrow[u, "\mathsf{id}+(\Sigma^\star(1+1))^{!_X}"']
\end{tikzcd}$$

Note that every map of type $1\to A+B$ into a coproduct (disjoint union) simply chooses an element of one the summands A or B. In particular, this applies to the map $\varrho_{1,1}$: It has domain $1^{\mathsf{ar(f)}}\times 1^{W+1\times\overline{W}}\cong 1$, so it chooses an element of $\Sigma^\star(1+1)$ or $(\Sigma^\star(1+1))^1$. Note that this element is independent of the given objects X and Y since the component $\varrho_{1,1}$ is independent from X and Y. It follows that the image of the upper leg

$$(\Sigma^\star(!_X+\mathsf{id})+(\Sigma^\star(!_X+\mathsf{id}))\cdot(\Sigma^\star(\mathsf{id}+!_Y)+(\Sigma^\star(\mathsf{id}+!_Y))^X)\cdot\varrho_{X,Y}$$

of the above commutative diagram is either a single element of $\Sigma^\star(1+1)$ or a single element of $(\Sigma^\star(X+1))^X$. This implies that the image of $\varrho_{X,Y}$ must be contained in one of the summands of its codomain: It is either a subset of $\Sigma^\star(X+Y)$ for every X,Y, or a subset of $(\Sigma^\star(X+Y))^X$ for every X,Y. This proves the isomorphism (3.11).

(3) It remains to show that the two summands in (3.11) are isomorphic to the corresponding summands in (3.10). For the first one, let $\mathsf{Nat}_X(F(X),G(X))$ denote the collection of natural transformations between functors $F,G\colon \mathbf{Set}\to\mathbf{Set}$. Then

$$\begin{aligned}
&\mathsf{DiNat}_{X,Y}\big(X^{\mathsf{ar(f)}}\times Y^{W+X\times\overline{W}}, \Sigma^\star(X+Y)\big)\\
&\cong \mathsf{Nat}_X\big(X^{\mathsf{ar(f)}}, \mathsf{Nat}_Y(Y^{W+X\times\overline{W}}, \Sigma^\star(X+Y))\big)\\
&\cong \mathsf{Nat}_Y\big(Y^{W+\mathsf{ar(f)}\times\overline{W}}, \Sigma^\star(\mathsf{ar(f)}+Y)\big)\\
&\cong \Sigma^\star(\mathsf{ar(f)}+W+\mathsf{ar(f)}\times\overline{W}).
\end{aligned}$$

The last two isomorphisms use the Yoneda lemma, and the first one is given by currying:

$$\varrho \quad\mapsto\quad (\,\lambda x\in X^{\mathsf{ar(f)}}.\,(\varrho_{X,Y}(x,-))_Y\,)_X.$$

Note that for every $x\in X^{\mathsf{ar(f)}}$ the family $(\varrho_{X,Y}(x,-)\colon Y^{W+X\times\overline{W}}\to\Sigma^\star(X+Y))_Y$ is indeed natural in Y; the naturality squares are equivalent to the ones witnessing naturality of $\varrho_{X,Y}$ in

Y. Similarly, the family $(\lambda x \in X^{\mathsf{ar(f)}}.\,(\varrho_{X,Y}(x,-))_Y)_X$ is natural in X; the naturality squares are equivalent to the commutative hexagons witnessing dinaturality of $\varrho_{X,Y}$ in X.

Much analogously, we have

$$\begin{aligned}
&\mathsf{DiNat}_{X,Y}\big(X^{\mathsf{ar(f)}} \times Y^{W+X\times\overline{W}}, (\Sigma^\star(X+Y))^X\big)\\
\cong\ &\mathsf{Nat}_X\big(X^{\mathsf{ar(f)}} \times X, \mathsf{Nat}_Y(Y^{W+X\times\overline{W}}, \Sigma^\star(X+Y))\big)\\
\cong\ &\mathsf{Nat}_X\big(X^{\mathsf{ar(f)}+1}, \mathsf{Nat}_Y(Y^{W+X\times\overline{W}}, \Sigma^\star(X+Y))\big)\\
\cong\ &\mathsf{Nat}_Y\big(Y^{W+(\mathsf{ar(f)}+1)\times\overline{W}}, \Sigma^\star(\mathsf{ar(f)}+1+Y)\big)\\
\cong\ &\Sigma^\star(\mathsf{ar(f)}+1+W+(\mathsf{ar(f)}+1)\times\overline{W}).
\end{aligned}$$

This concludes the proof. □

3.3 Nondeterministic xCL

Just as the λ-calculus, combinatory logic can be enriched with other features, such as nondeterminism, and the theory of applicative bisimulations can be readily developed for such extensions. For the λ-calculus this has been pioneered by Sangiorgi [48]. For example, consider an extension of **xCL** with a binary operator $\oplus$ representing nondeterministic choice. The grammar of the extended language $\mathbf{xCL}^\oplus$ is given by

$$\Lambda^\oplus_{\mathsf{CL}} ::= S \mid K \mid I \mid \mathsf{app}(\Lambda^\oplus_{\mathsf{CL}}, \Lambda^\oplus_{\mathsf{CL}}) \mid S'(\Lambda^\oplus_{\mathsf{CL}}) \mid K'(\Lambda^\oplus_{\mathsf{CL}}) \mid S''(\Lambda^\oplus_{\mathsf{CL}}, \Lambda^\oplus_{\mathsf{CL}}) \mid \Lambda^\oplus_{\mathsf{CL}} \oplus \Lambda^\oplus_{\mathsf{CL}}.$$

On the side of the operational semantics, $\mathbf{xCL}^\oplus$ has the same rules as **xCL** (see Figure 1), plus the following two rules for resolving nondeterminism:

$$\frac{}{p \oplus q \to p} \qquad\qquad \frac{}{p \oplus q \to q}$$

This semantics calls for the modification of the behaviour bifunctor $B(X,Y) = Y + Y^X$ to

$$B^\oplus(X,Y) = \mathscr{P}_\omega(Y + Y^X)\colon \mathbf{Set}^{\mathsf{op}} \times \mathbf{Set} \to \mathbf{Set}, \tag{3.12}$$

where $\mathscr{P}_\omega$ is the finite powerset functor. Sets of nondeterministic transition rules such as those of $\mathbf{xCL}^\oplus$ then correspond to families of functions

$$\varrho_{X,Y}\colon \Sigma(X \times B^\oplus(X,Y)) \to B^\oplus(X, \Sigma^\star(X+Y)) \qquad (X, Y \in \mathbf{Set})$$

dinatural in X and natural in Y. In analogy to Proposition 3.3, we have the following compositionality result:

Proposition 3.10. *Bisimilarity for $\mathbf{xCL}^\oplus$ is a congruence.*

Rather than giving another proof by induction on the syntax, we will derive this proposition from our abstract congruence result (Theorem 4.14). This highlights the advantage of the genericity achieved by working in a category-theoretic framework.

4 Higher-order abstract GSOS

We present the main contribution of our paper, a theory of abstract GSOS for higher-order languages that retains the key feature of Turi and Plotkin's first-order framework, namely that (under mild conditions) compositionality of specifications comes for free.

4.1 Higher-order GSOS laws

The results of the previous section, most notably Theorem 3.7, suggest a path towards modeling higher-order languages in an abstract, purely categorical fashion: Present their small-step operational semantics in terms of families of morphisms

$$\varrho_{X,Y}\colon \Sigma(X \times B(X,Y)) \to B(X, \Sigma^{\star}(X+Y)), \tag{4.1}$$

dinatural in $X \in \mathbb{C}$ and natural in $Y \in \mathbb{C}$, parametric in a base category $\mathbb{C}$ and two functors $\Sigma\colon \mathbb{C} \to \mathbb{C}$ and $B\colon \mathbb{C}^{\mathsf{op}} \times \mathbb{C} \to \mathbb{C}$ representing the syntax and behaviour of the language. The initial Σ-algebra $\mu\Sigma$ should thus correspond to the terms of the language and their dynamics should be modelled after a $B(\mu\Sigma, -)$-coalgebra structure $\gamma\colon \mu\Sigma \to B(\mu\Sigma, \mu\Sigma)$.

With the developments in Section 5 in mind, we will actually work with a slightly more general format where X is required to be a *pointed object*:

Notation 4.1. Given a fixed object V of a category $\mathbb{C}$, we let $V/\mathbb{C}$ denote the coslice category of *V-pointed objects*. Its objects are pairs (X, p_X) of an object $X \in \mathbb{C}$ and a morphism $p_X\colon V \to X$ of $\mathbb{C}$. A *morphism* from (X, p_X) to (Y, p_Y) is a morphism $h\colon X \to Y$ of $\mathbb{C}$ such that $h \cdot p_X = p_Y$. We write

$$j\colon V/\mathbb{C} \to \mathbb{C}$$

for the forgetful functor given by $(X, p_X) \mapsto X$ and $h \mapsto h$.

We think of V as a set of variables, and of a V-pointed object (X, p_X) as a set X of program terms in free variables from V with an embedding $p_X\colon V \to X$ of the variables.

Assumptions 4.2. From now on, we fix the following data:

(1) a category $\mathbb{C}$ with finite limits and colimits;

(2) an object $V \in \mathbb{C}$ of variables;

(3) two functors $\Sigma\colon \mathbb{C} \to \mathbb{C}$ and $B\colon \mathbb{C}^{\mathsf{op}} \times \mathbb{C} \to \mathbb{C}$, where Σ is of the form $\Sigma = V + \Sigma'$ for some $\Sigma'\colon \mathbb{C} \to \mathbb{C}$ and has free algebras on every object, hence generates a free monad $\Sigma^{\star}$. In addition, we require the functor $B(\mu\Sigma, -)$ to admit a final coalgebra, where $\mu\Sigma = \Sigma^{\star}0$.

Definition 4.3. A *V-pointed higher-order GSOS law* of Σ over B is a family of morphisms

$$\varrho_{X,Y}\colon \Sigma(jX \times B(jX,Y)) \to B(jX, \Sigma^{\star}(jX+Y)) \tag{4.2}$$

dinatural in $X \in V/\mathbb{C}$ and natural in $Y \in \mathbb{C}$.

Remark 4.4. More explicitly, dinaturality in $X \in V/\mathbb{C}$ means that the hexagon (3.8) commutes for all objects $(X, p_X), (X', p_{X'}) \in V/\mathbb{C}$ and $Y \in \mathbb{C}$ and all morphisms $f\colon (X, p_X) \to (X', p_{X'})$ in $V/\mathbb{C}$. Similarly, naturality in $Y \in \mathbb{C}$ means that the rectangle (3.9) commutes for all objects $(X, p_X) \in V/\mathbb{C}$ and $Y, Y' \in \mathbb{C}$ and all morphisms $g\colon Y \to Y'$ in $\mathbb{C}$.

Laws of the form (4.1) emerge from (4.2) by choosing $V = 0$, the initial object of $\mathbb{C}$. When running the semantics, both X and Y will be instantiated to the free algebra $\mu\Sigma$. Abstracting from this choice ensures that program terms are used in a parametrically polymorphic, uniform way.

Every object $X \in \mathbb{C}$ induces an endofunctor $B(X, -)\colon \mathbb{C} \to \mathbb{C}$. For instance, the transition system $\gamma\colon \Lambda_{\mathsf{CL}} \to \Lambda_{\mathsf{CL}} + \Lambda_{\mathsf{CL}}^{\Lambda_{\mathsf{CL}}}$ from (3.2) is a $B(\mu\Sigma, -)$-coalgebra. The state space Λ_{CL} is the initial Σ-algebra for the corresponding polynomial set functor Σ; the codomain is $B(\mu\Sigma, \mu\Sigma)$. The definition of the map γ is inductive on the structure of terms. It turns out to be an instance of a definition by structural induction in which we assign to a V-pointed higher-order GSOS law its canonical operational model. For technical reasons, we formulate the abstract definition of γ yet more generally, by parametrizing it with a Σ-algebra (A, a) — the motivating instance is obtained by instantiating A with the initial algebra $\mu\Sigma$.

Remark 4.5. For every Σ-algebra (A, a), we regard A as a V-pointed object with point

$$p_A = \big(V \xrightarrow{\mathsf{inl}} V + \Sigma' A = \Sigma A \xrightarrow{a} A\big).$$

Note that if $h\colon (A, a) \to (B, b)$ is a morphism of Σ-algebras, then h is also a morphism of the corresponding V-pointed objects.

Lemma 4.6. *Given a V-pointed higher-order GSOS law ϱ, every Σ-algebra (A, a) induces a unique morphism $a^\clubsuit\colon \mu\Sigma \to B(A, A)$ in $\mathbb{C}$ such that the following diagram commutes.*

$$\begin{tikzcd}
\Sigma(\mu\Sigma) \arrow[rrr, "\iota"] \arrow[d, "\Sigma\langle \mathsf{it}\, a,\, a^\clubsuit\rangle"'] & & & \mu\Sigma \arrow[d, "a^\clubsuit"] \\
\Sigma(A \times B(A, A)) \arrow[r, "\varrho_{A,A}"] & B(A, \Sigma^\star(A + A)) \arrow[r, "B(\mathsf{id}, \Sigma^\star\nabla)"] & B(A, \Sigma^\star A) \arrow[r, "B(\mathsf{id}, \hat{a})"] & B(A, A)
\end{tikzcd} \tag{4.3}$$

Here $(\mu\Sigma, \iota)$ is the initial Σ-algebra, $\hat{a}\colon \Sigma^\star A \to A$ is the Eilenberg–Moore algebra corresponding to the Σ-algebra (A, a), and we regard A as V-pointed as in Remark 4.5.

Proof. *Existence of $a^\clubsuit$.* By initiality of $\mu\Sigma$, there exists a unique morphism $\langle w, a^\clubsuit\rangle$ in $\mathbb{C}$ making the diagram below commute:

$$\begin{tikzcd}
\Sigma(\mu\Sigma) \arrow[rr, "\iota"] \arrow[d, "\Sigma\langle w,\, a^\clubsuit\rangle"'] & & \mu\Sigma \arrow[d, "\langle w,\, a^\clubsuit\rangle"] \\
\Sigma(A \times B(A, A)) \arrow[r, "\langle a\cdot\Sigma\mathsf{fst},\, \varrho_{A,A}\rangle"] & A \times B(A, \Sigma^\star(A + A)) \arrow[r, "\mathsf{id}\times B(\mathsf{id}, \hat{a}\cdot\Sigma^\star\nabla)"] & A \times B(A, A)
\end{tikzcd} \tag{4.4}$$

Postcomposing this diagram with $\mathsf{fst}\colon A \times B(A, A) \to A$ shows that $w\colon \mu\Sigma \to (A, a)$ is a Σ-algebra morphism; hence $w = \mathsf{it}\, a$ and the diagram (4.4) can be rewritten as

$$\begin{tikzcd}
\Sigma(\mu\Sigma) \arrow[rr, "\iota"] \arrow[d, "\Sigma\langle \mathsf{it}\, a,\, a^\clubsuit\rangle"'] & & \mu\Sigma \arrow[d, "\langle \mathsf{it}\, a,\, a^\clubsuit\rangle"] \\
\Sigma(A \times B(A, A)) \arrow[r, "\langle a\cdot\Sigma\mathsf{fst},\, \varrho_{A,A}\rangle"] & A \times B(A, \Sigma^\star(A + A)) \arrow[r, "\mathsf{id}\times B(\mathsf{id}, \hat{a}\cdot\Sigma^\star\nabla)"] & A \times B(A, A)
\end{tikzcd} \tag{4.5}$$

By postcomposing with $\mathsf{snd}\colon A \times B(A, A) \to B(A, A)$ we see that (4.3) commutes.

Uniqueness of $a^\clubsuit$. Suppose that $a^\clubsuit$ is a morphism such that (4.3) commutes. Then (4.4) commutes with $w = \mathsf{it}\, a$, so uniqueness of $a^\clubsuit$ is entailed by the uniqueness of $\langle w, a^\clubsuit\rangle$. □

Definition 4.7. The *operational model* of a V-pointed higher-order GSOS law ϱ is given by the $B(\mu\Sigma, -)$-coalgebra

$$\iota^\clubsuit\colon \mu\Sigma \to B(\mu\Sigma, \mu\Sigma).$$

Example 4.8. (1) Consider the higher-order GSOS law ϱ corresponding to **xCL** (see Examples 3.5 and 3.9). Then the operational model $\iota^\clubsuit$ is precisely that transition system $\gamma\colon \Lambda_{\mathsf{CL}} \to \Lambda_{\mathsf{CL}} + \Lambda_{\mathsf{CL}}^{\Lambda_{\mathsf{CL}}}$ of (3.2) induced by the rules in Figure 1. Given a Σ-algebra (A, a), the morphism $a^\clubsuit$ is obtained by interpreting all those transitions in the algebra A. For instance, since there is a transition $K \xrightarrow{t} K'(t)$, we have $a^\clubsuit(K) \in A^A$ given by $a^\clubsuit(K)(u) = (K')^A(u)$, where $(K')^A\colon A \to A$ is the interpretation of the unary operation symbol $K' \in \Sigma$ in A.

(2) More generally, the operational model $\gamma\colon \mu\Sigma \to \mu\Sigma + \mu\Sigma^{\mu\Sigma}$ of an $\mathcal{HO}$ specification, which runs programs according to the given inductive $\mathcal{HO}$ rules, coincides with the operational model $\iota^\clubsuit$ of its corresponding higher-order GSOS law (Theorem 3.7).

Remark 4.9 (First-order vs. higher-order abstract GSOS)**.** Higher-order abstract GSOS is a conservative extension of first-order abstract GSOS (Section 2.2). In more detail, given endofunctors $\Sigma, B_0\colon \mathbb{C} \to \mathbb{C}$, every first-order GSOS law

$$\varrho^0_Y\colon \Sigma(Y \times B_0 Y) \to B_0\Sigma^\star Y \qquad (Y \in \mathbb{C})$$

of Σ over B_0 can be turned into a 0-pointed higher-order GSOS law

$$\varrho_{X,Y}\colon \Sigma(X \times B(X,Y)) \to B(X, \Sigma^\star(X+Y)) \qquad (X, Y \in \mathbb{C})$$

of Σ over $B\colon \mathbb{C}^{\mathsf{op}} \times \mathbb{C} \to \mathbb{C}$, where $B(X,Y) = B_0 Y$, whose components are given by

$$\varrho_{X,Y} \;=\; \big(\;\Sigma(X \times B_0 Y) \xrightarrow{\Sigma(\mathsf{inl}\times B_0\mathsf{inr})} \Sigma((X+Y)\times B_0(X+Y)) \xrightarrow{\varrho^0_{X+Y}} B_0\Sigma^\star(X+Y)\;\big).$$

Let us check that ϱ is indeed a higher-order GSOS law. Naturality of $\varrho_{X,-}$ is shown by the following commutative diagram for $f\colon Y \to Y'$. The left-hand part obviously commutes, and the right-hand one commutes by naturality of ϱ^0.

$$\begin{tikzcd}
\Sigma(X \times B(X,Y)) = \Sigma(X \times B_0 Y) \arrow[r, "\Sigma(\mathsf{inl}\times B_0\mathsf{inr})"] \arrow[d, "\Sigma(\mathsf{id}\times B_0 f)"'] & \Sigma((X+Y)\times B_0(X+Y)) \arrow[r, "\varrho^0_{X+Y}"] \arrow[d, "\Sigma((\mathsf{id}+f)\times B_0(\mathsf{id}+f))"] & B_0\Sigma^\star(X+Y) = B(X,\Sigma^\star(X+Y)) \arrow[d, "B_0\Sigma^\star(\mathsf{id}+f)"] \\
\Sigma(X \times B(X,Y')) = \Sigma(X \times B_0 Y') \arrow[r, "\Sigma(\mathsf{inl}\times B_0\mathsf{inr})"] & \Sigma((X+Y')\times B_0(X+Y')) \arrow[r, "\varrho^0_{X+Y'}"] & B_0\Sigma^\star(X+Y') = B(X,\Sigma^\star(X+Y'))
\end{tikzcd}$$

Since $B(X,Y) = B_0 Y$ does not depend on its contravariant component, dinaturality of $\varrho_{-,Y}$ is equivalent to naturality and is shown by the commutative diagram below for $g\colon X \to X'$:

$$\begin{tikzcd}
\Sigma(X \times B(X,Y)) = \Sigma(X \times B_0 Y) \arrow[r, "\Sigma(\mathsf{inl}\times B_0\mathsf{inr})"] \arrow[d, "\Sigma(g\times\mathsf{id})"'] & \Sigma((X+Y)\times B_0(X+Y)) \arrow[r, "\varrho^0_{X+Y}"] \arrow[d, "\Sigma((g+\mathsf{id})\times B_0(g+\mathsf{id}))"] & B_0\Sigma^\star(X+Y) = B(X,\Sigma^\star(X+Y)) \arrow[d, "B_0\Sigma^\star(g+\mathsf{id})"] \\
\Sigma(X' \times B(X',Y)) = \Sigma(X' \times B_0 Y) \arrow[r, "\Sigma(\mathsf{inl}\times B_0\mathsf{inr})"] & \Sigma((X'+Y)\times B_0(X'+Y)) \arrow[r, "\varrho^0_{X'+Y}"] & B_0\Sigma^\star(X'+Y) = B(X',\Sigma^\star(X'+Y))
\end{tikzcd}$$

The two laws ϱ^0 and ϱ are semantically equivalent in the sense that their operational models

$$\gamma\colon \mu\Sigma \to B_0(\mu\Sigma) \qquad \text{and} \qquad \iota^\clubsuit\colon \mu\Sigma \to B_0(\mu\Sigma) = B(\mu\Sigma,\mu\Sigma)$$

coincide. To see this, recall from Section 2.2 that the coalgebra structure γ is uniquely determined by the following commutative diagram:

$$\begin{tikzcd}
\Sigma(\mu\Sigma) \arrow[rr, "\iota"] \arrow[d, "\Sigma\langle\mathsf{id},\gamma\rangle"'] & & \mu\Sigma \arrow[d, "\gamma"] \\
\Sigma(\mu\Sigma \times B_0(\mu\Sigma)) \arrow[r, "\varrho^0_{\mu\Sigma}"] & B_0\Sigma^\star(\mu\Sigma) \arrow[r, "B_0\hat{\iota}"] & B_0(\mu\Sigma)
\end{tikzcd}$$

Thus, we only need to show that $\iota^\clubsuit$ is such a γ, which follows from the commutative diagram below. The upper cell commutes by definition of $\iota^\clubsuit$, the cell involving $\varrho^0_{\mu\Sigma}$ commutes by naturality of ϱ^0, and the remaining cells commute either trivially or by definition.

$$
\begin{tikzcd}
\Sigma(\mu\Sigma) \arrow[rrrrr, "\iota"] \arrow[dr, "\langle \mathsf{it}\,\iota,\, \iota^\clubsuit\rangle"] \arrow[ddd, "\Sigma\langle \mathsf{id},\, \iota^\clubsuit\rangle"'] & & & & & \mu\Sigma \arrow[dl, "\iota^\clubsuit"'] \arrow[ddd, "\iota^\clubsuit"] \\
& \Sigma(\mu\Sigma \times B(\mu\Sigma,\mu\Sigma)) \arrow[r, "\varrho_{\mu\Sigma,\mu\Sigma}"] \arrow[d, "\Sigma(\mathsf{inl}\times B_0\mathsf{inr})"] \arrow[ddl, equals, bend right] & B(\mu\Sigma,\Sigma^\star(\mu\Sigma+\mu\Sigma)) \arrow[r, "B(\mathsf{id},\Sigma^\star\nabla)"] \arrow[dd, "B_0\Sigma^\star\nabla"] & B(\mu\Sigma,\Sigma^\star(\mu\Sigma)) \arrow[r, "B(\mathsf{id},\hat\iota)"] & B(\mu\Sigma,\mu\Sigma) \arrow[ddr, equals] & \\
& \Sigma((\mu\Sigma+\mu\Sigma)\times B_0(\mu\Sigma+\mu\Sigma)) \arrow[ur, "\varrho^0_{\mu\Sigma+\mu\Sigma}"'] \arrow[dl, "\Sigma(\nabla\times B_0\nabla)"] & & & & \\
\Sigma(\mu\Sigma\times B_0(\mu\Sigma)) \arrow[rr, "\varrho^0_{\mu\Sigma}"] & & B_0\Sigma^\star(\mu\Sigma) \arrow[rrr, "B_0\hat\iota"] & & & B_0(\mu\Sigma)
\end{tikzcd}
$$

Remark 4.10. The construction of the operational model (Definition 4.7) only uses the component $\varrho_{\mu\Sigma,\mu\Sigma}$ of the given higher-order GSOS law, followed by a codiagonal that simply forgets which of the two copies of $\mu\Sigma$ the individual variables of the output term come from. It would thus be tempting to generalize higher-order GSOS laws (4.2) to dinatural transformations of type

$$\varrho'_X \colon \Sigma(X\times B(X,X)) \to B(X,\Sigma^\star X). \tag{4.6}$$

Note that they are indeed more general: every higher-order GSOS law (4.2) induces a dinatural transformation (4.6) by putting

$$\varrho'_X = \big(\ \Sigma(X\times B(X,X)) \xrightarrow{\varrho_{X,X}} B(X,\Sigma^\star(X+X)) \xrightarrow{B(\mathsf{id},\nabla)} B(X,\Sigma^\star X)\ \big).$$

Unfortunately, this format appears to be too permissive to guarantee congruence. Specifically, the proof of Lemma 4.16 below, which is the key to our congruence result, crucially rests on the 'two-variable' form $\varrho_{X,Y}$ of higher-order GSOS laws and does not carry over to the generalized format.

4.2 Compositionality

We now investigate when a higher-order GSOS law gives rise to a compositional semantics. Recall from Section 2.2 that in first-order abstract GSOS, compositionality comes for free and is an immediate consequence of $\mu\Sigma$ extending to an initial bialgebra and νB extending to a final bialgebra for a given GSOS law. We shall see in Section 4.3 that the latter does not carry over to the higher-order setting. Therefore, we take a different route to compositionality, working in a framework of regular categories (Section 2.1).

Assuming that the final $B(\mu\Sigma,-)$-coalgebra

$$\zeta\colon Z \to B(\mu\Sigma, Z)$$

exists, we think of the unique coalgebra morphism $\mathsf{coit}(\iota^\clubsuit)\colon \mu\Sigma \to Z$ from the operational model $\iota^\clubsuit\colon \mu\Sigma \to B(\mu\Sigma,\mu\Sigma)$ (Definition 4.7) to (Z,ζ) as the map assigning to each program in $\mu\Sigma$ its abstract behaviour. The ensuing notion of *behavioural equivalence* is then expressed categorically by the kernel pair of $\mathsf{coit}(\iota^\clubsuit)$, i.e. the pullback

$$
\begin{tikzcd}
E \arrow[r, "p_1"] \arrow[d, "p_2"'] \arrow[dr, phantom, "\lrcorner", very near start] & \mu\Sigma \arrow[d, "\mathsf{coit}\,\iota^\clubsuit"] \\
\mu\Sigma \arrow[r, "\mathsf{coit}\,\iota^\clubsuit"] & Z
\end{tikzcd}
\tag{4.7}
$$

Definition 4.11. The kernel pair E is a *congruence* if it forms a subalgebra of the product algebra $\mu\Sigma \times \mu\Sigma$; that is, E can be equipped with a (necessarily unique) Σ-algebra structure (E, e) such that $p_1, p_2 \colon (E, e) \to (\mu\Sigma, \iota)$ are Σ-algebra homomorphisms.

Remark 4.12. For $\mathbb{C} = \mathbf{Set}$ and Σ a polynomial functor, the kernel E is the equivalence relation on $\mu\Sigma$ defined by

$$E = \{\, (s,t) \in \mu\Sigma \times \mu\Sigma \colon (\mathsf{coit}\, \iota^{\clubsuit})(s) = (\mathsf{coit}\, \iota^{\clubsuit})(t) \,\}$$

with the two projection maps $p_1(s,t) = s$ and $p_2(s,t) = t$, and E forms a congruence in the above categorical sense if and only if it forms a congruence in the usual algebraic sense as recalled in Section 2.1, i.e. it is compatible with the Σ-algebra structure of $\mu\Sigma$.

Our main compositionality result asserts that for a regular base category $\mathbb{C}$ and under mild conditions on the functors Σ and B, behavioural equivalence is always a congruence.

Remark 4.13. Recall that a parallel pair $f, g \colon X \rightrightarrows Y$ is *reflexive* if there exists a common splitting, viz. a morphism $s \colon Y \to X$ such that $f \cdot s = \mathsf{id}_Y = g \cdot s$. A *reflexive coequalizer* is a coequalizer of a reflexive pair. Preservation of reflexive coequalizers is a relatively mild condition for set functors. For instance, every polynomial set functor Σ and, more generally, every finitary set functor preserves reflexive coequalizers [6, Cor. 6.30].

Theorem 4.14 (Compositionality)**.** *Suppose that Assumptions 4.2 hold, and let ϱ be a V-pointed higher-order GSOS law of $\Sigma \colon \mathbb{C} \to \mathbb{C}$ over $B \colon \mathbb{C}^{\mathsf{op}} \times \mathbb{C} \to \mathbb{C}$. Suppose that the category $\mathbb{C}$ is regular, that Σ preserves reflexive coequalizers, and that B preserves monomorphisms. Then the kernel pair of $\mathsf{coit}(\iota^{\clubsuit}) \colon \mu\Sigma \to Z$ is a congruence.*

(Note that a morphism (f, g) in $\mathbb{C}^{\mathsf{op}} \times \mathbb{C}$ is monic iff g is monic in $\mathbb{C}$ and f is epic in $\mathbb{C}$.)

Remark 4.15. We will show that the coequalizer $q \colon \mu\Sigma \twoheadrightarrow Q$ of p_1, p_2 in $\mathbb{C}$ can be equipped with a Σ-algebra structure $a \colon \Sigma Q \to Q$ such that q is a Σ-algebra morphism from $\mu\Sigma$ to (Q, a), that is, $q = \mathsf{it}(a)$. This immediately implies the theorem: since p_1, p_2 is the kernel pair of q in $\mathbb{C}$ and the forgetful functor from $\mathbf{Alg}(\Sigma)$ to $\mathbb{C}$ creates limits, in particular kernel pairs, there exists a unique Σ-algebra structure e on E such that $p_1, p_2 \colon (E, e) \to (\mu\Sigma, \iota)$ are Σ-algebra homomorphisms (and this makes p_1, p_2 the kernel pair of q in $\mathbf{Alg}(\Sigma)$).

We make use of the following two lemmas, which are of independent interest, en route to proving Theorem 4.14. First, we establish a crucial connection between $\iota^{\clubsuit}$ and $a^{\clubsuit}$ (for general $a \colon \Sigma A \to A$), showing that they can be unified by running the unique morphism $\mathsf{it}(a) \colon \mu\Sigma \to A$ at the covariant and the contravariant positions of B correspondingly. This critically relies on (di)naturality of the given law ϱ.

Lemma 4.16. *Let (A, a) be a Σ-algebra. Then the following diagram commutes:*

$$\begin{tikzcd} \mu\Sigma \arrow[r, "\iota^{\clubsuit}"] \arrow[d, "a^{\clubsuit}"'] & B(\mu\Sigma, \mu\Sigma) \arrow[d, "B(\mathsf{id}, \mathsf{it}\, a)"] \\ B(A, A) \arrow[r, "B(\mathsf{it}\, a, \mathsf{id})"] & B(\mu\Sigma, A) \end{tikzcd}$$

Example 4.17 (**xCL**, cf. Example 4.8)**.** The two legs of the diagram send the term $K \in \mu\Sigma$ to the function $f \in A^{\mu\Sigma}$ given by $f(t) = (K')^A((\mathsf{it}(a))(t))$, equivalently $f(t) = (\mathsf{it}(a))(K'(t))$ since $\mathsf{it}(a)$ is a Σ-algebra morphism.

Proof of Lemma 4.16. We strengthen the claim of the lemma a bit and show that the outside of the following diagram commutes:

$$\begin{tikzcd}
\mu\Sigma \arrow[rr, "{\langle \mathsf{id},\, \iota^{\clubsuit} \rangle}"] \arrow[d, "{\langle \mathsf{id},\, a^{\clubsuit} \rangle}"'] \arrow[drr, "\mathsf{it}\, b"] & & \mu\Sigma \times B(\mu\Sigma, \mu\Sigma) \arrow[d, "{\mathsf{id}\times B(\mathsf{id},\mathsf{it}\, a)}"] \\
\mu\Sigma \times B(A, A) \arrow[rr, "{\mathsf{id}\times B(\mathsf{it}\, a,\mathsf{id})}"] & & \mu\Sigma \times B(\mu\Sigma, A)
\end{tikzcd}$$

From this, the goal easily follows by applying the right projection snd. To prove commutativity of the diagram, we consider the Σ-algebra structure b on $\mu\Sigma\times B(\mu\Sigma, A)$ given by the following composite:

$$\begin{tikzcd}
\Sigma(\mu\Sigma \times B(\mu\Sigma, A)) \arrow[r, "{\langle \iota\cdot\Sigma\mathsf{fst},\, \varrho_{\mu\Sigma,A} \rangle}"] & \mu\Sigma \times B(\mu\Sigma, \Sigma^{\star}(\mu\Sigma + A)) \arrow[dl, "{\mathsf{id}\times B(\mathsf{id},\Sigma^{\star}(\mathsf{it}\, a+\mathsf{id}))}"'] & \\
\mu\Sigma \times B(\mu\Sigma, \Sigma^{\star}(A + A)) \arrow[r, "{\mathsf{id}\times B(\mathsf{id},\Sigma^{\star}\nabla)}"] & \mu\Sigma \times B(\mu\Sigma, \Sigma^{\star} A) \arrow[r, "{\mathsf{id}\times B(\mathsf{id},\hat{a})}"] & \mu\Sigma \times B(\mu\Sigma, A).
\end{tikzcd}$$

We will show that the composite morphisms

$$\mu\Sigma \xrightarrow{\langle \mathsf{id},\ \iota^{\clubsuit}\rangle} \mu\Sigma \times B(\mu\Sigma, \mu\Sigma) \xrightarrow{\mathsf{id}\times B(\mathsf{id},\,\mathsf{it}\, a)} \mu\Sigma \times B(\mu\Sigma, A) \tag{4.8}$$

$$\mu\Sigma \xrightarrow{\langle \mathsf{id},\ a^{\clubsuit}\rangle} \mu\Sigma \times B(A, A) \xrightarrow{\mathsf{id}\times B(\mathsf{it}\, a,\,\mathsf{id})} \mu\Sigma \times B(\mu\Sigma, A), \tag{4.9}$$

are both Σ-algebra morphisms from $\mu\Sigma$ to $(\mu\Sigma \times B(\mu\Sigma, A), b)$; hence they are equal to $\mathsf{it}\, b$ by initiality of $\mu\Sigma$.

The morphism (4.8) is a composition of Σ-algebra morphisms: $\langle \mathsf{id},\ \iota^{\clubsuit}\rangle$ is so by definition, and $\mathsf{id}\times B(\mathsf{id}, \mathsf{it}\, a)$ is so by commutativity of the following diagram:

$$\begin{tikzcd}
\Sigma(\mu\Sigma \times B(\mu\Sigma, \mu\Sigma)) \arrow[r, "{\Sigma(\mathsf{id}\times B(\mathsf{id},\mathsf{it}\, a))}"] \arrow[dd, "{\langle \iota\cdot\Sigma\mathsf{fst},\, \varrho_{\mu\Sigma,\mu\Sigma} \rangle}"'] & \Sigma(\mu\Sigma \times B(\mu\Sigma, A)) \arrow[d, "{\langle \iota\cdot\Sigma\mathsf{fst},\, \varrho_{\mu\Sigma,A} \rangle}"] \\
& \mu\Sigma \times B(\mu\Sigma, \Sigma^{\star}(\mu\Sigma + A)) \arrow[d, "{\mathsf{id}\times B(\mathsf{id},\Sigma^{\star}(\mathsf{it}\, a+\mathsf{id}))}"] \\
\mu\Sigma \times B(\mu\Sigma, \Sigma^{\star}(\mu\Sigma + \mu\Sigma)) \arrow[r, "{\mathsf{id}\times B(\mathsf{id},\Sigma^{\star}(\mathsf{it}\, a+\mathsf{it}\, a))}"] \arrow[ur, bend left, "{\mathsf{id}\times B(\mathsf{id},\Sigma^{\star}(\mathsf{id}+\mathsf{it}\, a))}"] \arrow[d, "{\mathsf{id}\times B(\mathsf{id},\Sigma^{\star}\nabla)}"'] & \mu\Sigma \times B(\mu\Sigma, \Sigma^{\star}(A + A)) \arrow[d, "{\mathsf{id}\times B(\mathsf{id},\Sigma^{\star}\nabla)}"] \\
\mu\Sigma \times B(\mu\Sigma, \Sigma^{\star}(\mu\Sigma)) \arrow[r, "{\mathsf{id}\times B(\mathsf{id},\Sigma^{\star}(\mathsf{it}\, a))}"] \arrow[d, "{\mathsf{id}\times B(\mathsf{id},\hat{\iota})}"'] & \mu\Sigma \times B(\mu\Sigma, \Sigma^{\star} A) \arrow[d, "{\mathsf{id}\times B(\mathsf{id},\hat{a})}"] \\
\mu\Sigma \times B(\mu\Sigma, \mu\Sigma) \arrow[r, "{\mathsf{id}\times B(\mathsf{id},\mathsf{it}\, a)}"] & \mu\Sigma \times B(\mu\Sigma, A)
\end{tikzcd}$$

The three upper cells commute by naturality of ϱ and by functoriality of B in the second argument; the bottom cell commutes because $\mathsf{it}(a)\colon \mu\Sigma \to A$ is a Σ-algebra morphism.

That the morphism (4.9) is a Σ-algebra morphism is shown from the following diagram:

$$\begin{tikzcd}
\Sigma(\mu\Sigma) \arrow[r, "\Sigma\langle \mathsf{id},\, a^{\clubsuit}\rangle"] \arrow[dddd, "\iota"'] & \Sigma(\mu\Sigma \times B(A,A)) \arrow[r, "\Sigma(\mathsf{id}\times B(\mathsf{it}\,a,\,\mathsf{id}))"] \arrow[d, "\langle \iota\cdot\Sigma\mathsf{fst},\, \Sigma(\mathsf{it}\,a\times\mathsf{id})\rangle"'] & \Sigma(\mu\Sigma \times B(\mu\Sigma,A)) \arrow[d, "\langle \iota\cdot\Sigma\mathsf{fst},\, \varrho_{\mu\Sigma,A}\rangle"] \\
 & \mu\Sigma \times \Sigma(A \times B(A,A)) \arrow[d, "\mathsf{id}\times\varrho_{A,A}"'] & \mu\Sigma \times B(\mu\Sigma, \Sigma^\star(\mu\Sigma + A)) \arrow[d, "\mathsf{id}\times B(\mathsf{id},\Sigma^\star(\mathsf{it}\,a+\mathsf{id}))"] \\
 & \mu\Sigma \times B(A, \Sigma^\star(A+A)) \arrow[r, "\mathsf{id}\times B(\mathsf{it}\,a,\,\mathsf{id})"] \arrow[d, "\mathsf{id}\times B(\mathsf{id},\Sigma^\star\nabla)"'] & \mu\Sigma \times B(\mu\Sigma, \Sigma^\star(A+A)) \arrow[d, "\mathsf{id}\times B(\mathsf{id},\Sigma^\star\nabla)"] \\
 & \mu\Sigma \times B(A, \Sigma^\star A) \arrow[r, "\mathsf{id}\times B(\mathsf{it}\,a,\,\mathsf{id})"] \arrow[d, "\mathsf{id}\times B(\mathsf{id},\hat{a})"'] & \mu\Sigma \times B(\mu\Sigma, \Sigma^\star A) \arrow[d, "\mathsf{id}\times B(\mathsf{id},\hat{a})"] \\
\mu\Sigma \arrow[r, "\langle \mathsf{id},\, a^{\clubsuit}\rangle"] & \mu\Sigma \times B(A,A) \arrow[r, "\mathsf{id}\times B(\mathsf{it}\,a,\,\mathsf{id})"] & \mu\Sigma \times B(\mu\Sigma, A)
\end{tikzcd}$$

The left cell commutes by definition of $a^{\clubsuit}$. The two lower right cells commute by functoriality of B, and the right upper cell commutes by an instance of dinaturality for ϱ:

$$\begin{tikzcd}
 & \Sigma(\mu\Sigma \times B(\mu\Sigma,A)) \arrow[r, "\varrho_{\mu\Sigma,A}"] & B(\mu\Sigma, \Sigma^\star(\mu\Sigma + A)) \arrow[dr, "B(\mathsf{id},\Sigma^\star(\mathsf{it}\,a+\mathsf{id}))"] & \\
\Sigma(\mu\Sigma \times B(A,A)) \arrow[ur, "\Sigma(\mathsf{id}\times B(\mathsf{it}\,a,\mathsf{id}))"] \arrow[dr, "\Sigma(\mathsf{it}\,a\times\mathsf{id})"'] & & & B(\mu\Sigma, \Sigma^\star(A+A)) \\
 & \Sigma(A \times B(A,A)) \arrow[r, "\varrho_{A,A}"] & B(A, \Sigma^\star(A+A)) \arrow[ur, "B(\mathsf{it}\,a,\mathsf{id})"'] &
\end{tikzcd}$$

□

Using the universal property of the pullback (4.7), we obtain a morphism $d\colon \mu\Sigma \to E$ such that $p_1 \cdot d = \mathsf{id}$ and $p_2 \cdot d = \mathsf{id}$. (This only needs the definition of the pullback, not the previous lemma.) It follows that $p_1^\star, p_2^\star\colon \Sigma^\star E \rightrightarrows \mu\Sigma$ is a reflexive pair in $\mathbb{C}$ with common section $\eta_E \cdot d$, where η is the unit of the monad $\Sigma^\star$ and $p_i^\star$ is the free extension of $p_i\colon E \to \mu\Sigma$. By our assumptions, the coequalizer of $p_1^\star$ and $p_2^\star$ is preserved by the functor Σ. Hence, there exists a Σ-algebra structure $\iota_\sim\colon \Sigma(\mu\Sigma_\sim) \to \mu\Sigma_\sim$, obtained using the universal property of the coequalizer $\Sigma(\mathsf{it}(\iota_\sim))$ from the diagram

$$\begin{tikzcd}
\Sigma\Sigma^\star E \arrow[r, shift left, "\Sigma p_1^\star"] \arrow[r, shift right, "\Sigma p_2^\star"'] \arrow[d, "\iota_E"'] & \Sigma(\mu\Sigma) \arrow[r, "\Sigma(\mathsf{it}\,\iota_\sim)"] \arrow[d, "\iota"] & \Sigma(\mu\Sigma_\sim) \arrow[d, dashed, "\iota_\sim"] \\
\Sigma^\star E \arrow[r, shift left, "p_1^\star"] \arrow[r, shift right, "p_2^\star"'] & \mu\Sigma \arrow[r, "\mathsf{it}\,\iota_\sim"] & \mu\Sigma_\sim .
\end{tikzcd} \tag{4.10}$$

Here ι_E denotes the Σ-algebra structure of the free algebra $\Sigma^\star E$, and we already denote the coequalizer of $p_1^\star$ and $p_2^\star$ by $\mathsf{it}(\iota_\sim)$, as commutation of the right-hand side identifies it as the unique Σ-algebra morphism induced by $\iota_\sim$.

Lemma 4.18. *Under the conditions of Theorem 4.14, there exists a coalgebra structure $\varsigma\colon \mu\Sigma_\sim \to B(\mu\Sigma_\sim, \mu\Sigma_\sim)$ making the triangle below commute, where $\iota_\sim^{\clubsuit} = (\iota_\sim)^{\clubsuit}$:*

$$\begin{tikzcd}
 & \mu\Sigma \arrow[dl, "\mathsf{it}\,\iota_\sim"'] \arrow[dr, "\iota_\sim^{\clubsuit}"] & \\
\mu\Sigma_\sim \arrow[rr, "\varsigma"] & & B(\mu\Sigma_\sim, \mu\Sigma_\sim)
\end{tikzcd}$$

Proof. By definition of $\mathsf{it}(\iota_\sim)$ as a coequalizer of $p_1^\star$ and $p_2^\star$, it suffices to show that $\iota^\clubsuit_\sim$ also coequalizes $p_1^\star$ and $p_2^\star$, which we strengthen to $\langle \mathsf{it}\,\iota_\sim,\, \iota^\clubsuit_\sim\rangle \cdot p_1^\star = \langle \mathsf{it}\,\iota_\sim, \iota^\clubsuit_\sim\rangle \cdot p_2^\star$. Since $\langle \mathsf{it}\,\iota_\sim,\, \iota^\clubsuit_\sim\rangle \cdot p_1^\star$ and $\langle \mathsf{it}\,\iota_\sim,\, \iota^\clubsuit_\sim\rangle \cdot p_2^\star$ are Σ-algebra morphisms, see diagram (4.5), whose domain is the free Σ-algebra $\Sigma^\star E$, it suffices to show that the desired equation holds when precomposed with $\eta_E \colon E \to \Sigma^\star E$. Thus, it remains to show that $\langle \mathsf{it}\,\iota_\sim,\, \iota^\clubsuit_\sim\rangle \cdot p_1 = \langle \mathsf{it}\,\iota_\sim,\, \iota^\clubsuit_\sim\rangle \cdot p_2$, which, in turn, reduces to $\iota^\clubsuit_\sim \cdot p_1 = \iota^\clubsuit_\sim \cdot p_2$. We have

$$
\begin{aligned}
& \iota^\clubsuit_\sim \cdot p_1 = \iota^\clubsuit_\sim \cdot p_2 & \\
\iff\ & B(\mathsf{it}\,\iota_\sim, \mathsf{id}) \cdot \iota^\clubsuit_\sim \cdot p_1 = B(\mathsf{it}\,\iota_\sim, \mathsf{id}) \cdot \iota^\clubsuit_\sim \cdot p_2 & \quad B(\mathsf{it}\,\iota_\sim, \mathsf{id}) \text{ is mono} \\
\iff\ & B(\mathsf{id}, \mathsf{it}\,\iota_\sim) \cdot \iota^\clubsuit \cdot p_1 = B(\mathsf{id}, \mathsf{it}\,\iota_\sim) \cdot \iota^\clubsuit \cdot p_2. & \quad \text{Lemma 4.16}
\end{aligned}
$$

Let us denote the coequalizer of p_1, p_2 by $q \colon \mu\Sigma \to Q$. Since $\mathsf{it}(\iota_\sim)$ coequalizes p_1 and p_2, it factorizes through q. It thus suffices to show that

$$B(\mathsf{id}, q) \cdot \iota^\clubsuit \cdot p_1 = B(\mathsf{id}, q) \cdot \iota^\clubsuit \cdot p_2.$$

By regularity of the base category $\mathbb{C}$, the unique morphism $m \colon Q \to Z$ such that $\mathsf{coit}(\iota^\clubsuit) = m \cdot q$ is monic. Since $B(\mathsf{id}, -)$ preserves monomorphisms, it suffices to show that

$$B(\mathsf{id}, \mathsf{coit}\,\iota^\clubsuit) \cdot \iota^\clubsuit \cdot p_1 = B(\mathsf{id}, \mathsf{coit}\,\iota^\clubsuit) \cdot \iota^\clubsuit \cdot p_2.$$

Note that $B(\mathsf{id}, \mathsf{coit}\,\iota^\clubsuit) \cdot \iota^\clubsuit = \zeta \cdot \mathsf{coit}\,\iota^\clubsuit$ since $\mathsf{coit}(\iota^\clubsuit)$ is a coalgebra morphism from $(\mu\Sigma, \iota^\clubsuit)$ to (Z, ζ). Hence the above equation follows from $\mathsf{coit}\,\iota^\clubsuit \cdot p_1 = \mathsf{coit}\,\iota^\clubsuit \cdot p_2$, which holds by (4.7). □

These preparations in hand, we can proceed with the proof of the main result.

Proof of Theorem 4.14. Using the coalgebra $\varsigma \colon \mu\Sigma_\sim \to B(\mu\Sigma_\sim, \mu\Sigma_\sim)$ from Lemma 4.18 together with Lemma 4.16, the upper rectangular cell of the following diagram commutes:

$$
\begin{tikzcd}
\mu\Sigma \arrow[rr, "\iota^\clubsuit"] \arrow[d, "\mathsf{it}\,\iota_\sim"'] \arrow[dr, "\iota^\clubsuit_\sim"] \arrow[dd, bend right=60, "\mathsf{coit}\,\iota^\clubsuit"'] & & B(\mu\Sigma, \mu\Sigma) \arrow[d, "B(\mathsf{id}, \mathsf{it}\,\iota_\sim)"] \arrow[dd, bend left=60, "B(\mathsf{id}, \mathsf{coit}\,\iota^\clubsuit)"] \\
\mu\Sigma_\sim \arrow[r, "\varsigma"] \arrow[d, "m"'] & B(\mu\Sigma_\sim, \mu\Sigma_\sim) \arrow[r, "B(\mathsf{it}\,\iota_\sim, \mathsf{id})"] & B(\mu\Sigma, \mu\Sigma_\sim) \arrow[d, "B(\mathsf{id}, m)"] \\
Z \arrow[rr, "\zeta"] & & B(\mu\Sigma, Z)
\end{tikzcd}
\tag{4.11}
$$

By finality of (Z, ζ), we also have a morphism m such that the lower rectangular cell commutes. Therefore, $\mathsf{coit}(\iota^\clubsuit) = m \cdot \mathsf{it}\,\iota_\sim$ by uniqueness of $\mathsf{coit}(\iota^\clubsuit)$. From this we derive the desired result as follows. First, we obtain a $\Sigma^\star$-algebra structure $e \colon \Sigma^\star E \to E$ such that $p_1 \cdot e = p_1^\star$ and $p_2 \cdot e = p_2^\star$ by the universal property of E as the pullback (4.7), using that

$$\mathsf{coit}(\iota^\clubsuit) \cdot p_1^\star = m \cdot (\mathsf{it}\,\iota_\sim) \cdot p_1^\star = m \cdot (\mathsf{it}\,\iota_\sim) \cdot p_2^\star = \mathsf{coit}(\iota^\clubsuit) \cdot p_2^\star$$

by (4.10). This entails that the pair p_1, p_2 has the same coequalizer as $p_1^\star, p_2^\star$, i.e. $\mathsf{it}(\iota_\sim)$. Indeed, $\mathsf{it}(\iota_\sim)$ coequalizes $p_1 = p_1^\star \cdot \eta_E$ and $p_2 = p_2^\star \cdot \eta_E$ by definition, and every morphism that coequalizes p_1 and p_2 must coequalize $p_1 \cdot e = p_1^\star$ and $p_2 \cdot e = p_2^\star$, and hence factor uniquely through $\mathsf{it}(\iota_\sim)$. Since $\mathsf{it}(\iota_\sim)$ is the coequalizer of p_1, p_2 in $\mathbb{C}$, and moreover $\mathsf{it}(\iota_\sim)$ is a Σ-algebra morphism by (4.10), we can use Remark 4.15 to conclude the proof. □

Example 4.19. The **xCL** calculus (Section 3.1) satisfies the assumptions of Theorem 4.14: **Set** is a regular category, every polynomial functor Σ preserves reflexive coequalizers (see Remark 4.13), and the behaviour functor $B(X,Y) = Y + Y^X$ maps surjections to injections in the contravariant argument and preserves injections in the covariant one. Consequently, compositionality of **xCL** (Proposition 3.3) is an instance of Theorem 4.14. More generally, the compositionality of $\mathcal{HO}$ specifications (Proposition 3.6) follows from Theorem 4.14.

Example 4.20. The nondeterministic **xCL** calculus (Section 3.3) is handled analogously; just observe that the finite power set functor $\mathscr{P}_\omega$ preserves both surjections and injections. Thus, compositionality (Proposition 3.10) again follows from Theorem 4.14.

4.3 Higher-order bialgebras

We conclude this section with a bialgebraic perspective on higher-order GSOS laws. First, we isolate the underlying higher-order notion of coalgebra:

Definition 4.21 (Higher-Order Coalgebra)**.** A *higher-order coalgebra* for a mixed variance bifunctor $B\colon \mathbb{C}^{\mathsf{op}}\times\mathbb{C} \to \mathbb{C}$ is a pair (C,c) of an object $C \in \mathbb{C}$ and a morphism $c\colon C \to B(C,C)$. A *morphism* from (C,c) to a higher-order coalgebra (C',c') is a morphism $h\colon C \to C'$ of $\mathbb{C}$ such that the following diagram commutes:

$$
\begin{tikzcd}
C \arrow[rr, "c"] \arrow[d, "h"'] & & B(C,C) \arrow[d, "B(\mathsf{id},h)"] \\
C' \arrow[r, "c'"] & B(C',C') \arrow[r, "B(h,\mathsf{id})"] & B(C,C')
\end{tikzcd}
$$

Proposition 4.22. *Higher-order coalgebras for B and their morphisms form a category.*

Proof. Clearly id_C is a morphism from (C,c) to (C,c). Moreover, the composite $h\cdot g$ of two morphisms $g\colon (C,c) \to (C',c')$ and $h\colon (C',c') \to (C'',c'')$ of higher-order coalgebras is again a morphism of higher-order coalgebras by the commutative diagram below:

$$
\begin{tikzcd}
C \arrow[rrr, "c"] \arrow[d, "g"'] & & & B(C,C) \arrow[d, "B(\mathsf{id},g)"] \\
C' \arrow[rr, "c'"] \arrow[d, "h"'] & & B(C',C') \arrow[r, "B(g,\mathsf{id})"] \arrow[d, "B(\mathsf{id},h)"'] & B(C,C') \arrow[d, "B(\mathsf{id},h)"] \\
C'' \arrow[r, "c''"] & B(C'',C'') \arrow[r, "B(h,\mathsf{id})"] & B(C',C'') \arrow[r, "B(g,\mathsf{id})"] & B(C,C'')
\end{tikzcd}
$$
□

A higher-order bialgebra for a higher-order GSOS law ϱ combines an algebra structure with a higher-order coalgebra structure compatible with ϱ:

Definition 4.23 (Higher-Order Bialgebra)**.** Given a V-pointed higher-order GSOS law ϱ, a *ϱ-bialgebra* is a triple (A,a,c) such that (A,a) is a Σ-algebra and (A,c) is a higher-order B-coalgebra making the following diagram commute:

$$
\begin{tikzcd}
\Sigma A \arrow[r, "a"] \arrow[d, "\Sigma\langle\mathsf{id},c\rangle"'] & A \arrow[r, "c"] & B(A,A) \\
\Sigma(A\times B(A,A)) \arrow[r, "\varrho_{A,A}"] & B(A,\Sigma^\star(A+A)) \arrow[r, "B(\mathsf{id},\Sigma^\star\nabla)"] & B(A,\Sigma^\star A) \arrow[u, "B(\mathsf{id},\hat{a})"']
\end{tikzcd}
$$

A *morphism* from (A, a, c) to a ϱ-bialgebra (A', a', c') is a morphism $h\colon A \to A'$ of $\mathbb{C}$ that is both a morphism of Σ-algebras and of higher-order B-coalgebras.

An *initial* (*final*) ϱ-bialgebra is simply an initial (final) object of the category of ϱ-bialgebras and their morphisms. Similar to the first-order case, the initial ϱ-bialgebra extends the initial Σ-algebra:

Proposition 4.24. *The triple $(\mu\Sigma, \iota, \iota^{\clubsuit})$ is the initial ϱ-bialgebra.*

Proof. It follows directly by definition of $\iota^{\clubsuit}$ in (4.3), and by observing that $\mathsf{it}(\iota) = \mathsf{id}$, that $(\mu\Sigma, \iota, \iota^{\clubsuit})$ is a ϱ-bialgebra. To prove initiality, suppose that (A, a, c) is a ϱ-bialgebra. We show that $\mathsf{it}/a)\colon \mu\Sigma \to A$ is the unique ϱ-bialgebra morphism from $(\mu\Sigma, \iota, \iota^{\clubsuit})$ to (A, a, c). To show that $\mathsf{it}(a)$ is a ϱ-bialgebra morphism, we need to verify that the diagram

$$
\begin{tikzcd}
\mu\Sigma \arrow[rrr, "\iota^{\clubsuit}"] \arrow[d, "\mathsf{it}\,a"'] \arrow[drr, "a^{\clubsuit}"] & & & B(\mu\Sigma, \mu\Sigma) \arrow[d, "B(\mathsf{id},\mathsf{it}\,a)"] \\
A \arrow[rr, "c"] & & B(A, A) \arrow[r, "B(\mathsf{it}\,a,\mathsf{id})"] & B(\mu\Sigma, A)
\end{tikzcd}
$$

commutes. The quadrangular cell commutes by Lemma 4.16, and we are left to show that $c \cdot (\mathsf{it}\,a) = a^{\clubsuit}$. This follows from the fact that $c \cdot (\mathsf{it}\,a)$ satisfies the characteristic property of $a^{\clubsuit}$ given by (4.3). Indeed, the diagram

$$
\begin{tikzcd}
\Sigma(\mu\Sigma) \arrow[rrr, "\iota"] \arrow[d, "\Sigma(\mathsf{it}\,a)"'] \arrow[dd, bend right=60, "\Sigma\langle \mathsf{it}\,a,\, c\cdot\mathsf{it}\,a\rangle"'] & & & \mu\Sigma \arrow[d, "\mathsf{it}\,a"] \arrow[dd, bend left=60, "c\cdot\mathsf{it}\,a"] \\
\Sigma A \arrow[rrr, "a"] \arrow[d, "\Sigma\langle \mathsf{id},\, c\rangle"'] & & & A \arrow[d, "c"] \\
\Sigma(A \times B(A, A)) \arrow[r, "\varrho_{A,A}"] & B(A, \Sigma^{\star} A) \arrow[r, "B(\mathsf{id},\Sigma^{\star}\nabla)"] & B(A, \Sigma^{\star}(A + A)) \arrow[r, "B(\mathsf{id},\hat{a})"] & B(A, A)
\end{tikzcd}
$$

commutes: the top cell commutes by definition of $\mathsf{it}(a)$, and the bottom one commutes by the assumption that (A, a, c) is a ϱ-bialgebra.

Uniqueness of the bialgebra morphism $\mathsf{it}(a)\colon \mu\Sigma \to A$ is by initiality of $\mu\Sigma$ as a Σ-algebra, since every bialgebra morphism is, by definition, in particular a Σ-algebra morphism. $\square$

In first-order abstract GSOS, a final bialgebra for a GSOS law can be derived from the final coalgebra νB. In the higher-order setting, the intended semantic domain for a higher-order GSOS law ϱ is $\nu B(\mu\Sigma, -)$, the final coalgebra for the endofunctor $B(\mu\Sigma, -)$. However, this object generally does not extend to a final ϱ-bialgebra. In fact, a final ϱ-bialgebra usually fails to exist, even for simple 'deterministic' behaviour bifunctors:

Example 4.25. Consider the bifunctor $B(X, Y) = 2^X\colon \mathbf{Set}^{\mathsf{op}} \times \mathbf{Set} \to \mathbf{Set}$, the empty signature $\Sigma = \emptyset$, and the unique ($\emptyset$-pointed) higher-order GSOS law ϱ of Σ over B. A ϱ-bialgebra is just a map $z\colon Z \to 2^Z$, and a morphism from a ϱ-bialgebra (W, w) to (Z, z) is a map $h\colon W \to Z$ making the diagram

$$
\begin{tikzcd}
W \arrow[rr, "w"] \arrow[d, "h"'] & & 2^W \arrow[d, equal] \\
Z \arrow[r, "z"] & 2^Z \arrow[r, "2^h"] & 2^W
\end{tikzcd}
\tag{4.12}
$$

commute. We claim that no final ϱ-bialgebra exists, despite the endofunctor $B(\mu\Sigma, -) = 2^{\mu\Sigma} \cong 1$ having a final coalgebra. Suppose for a contradiction that (Z, z) is a final ϱ-bialgebra.

$$\mathsf{app1}\,\frac{p \to p'}{p\,q \to p'\,q} \qquad \mathsf{app2}\,\frac{}{(\lambda x.p)\,q \to p[q/x]}$$

Fig. 2. Small-step operational semantics of the call-by-name λ-calculus.

Choose an arbitrary ϱ-bialgebra (W, w) such that $|W| > |Z|$ and $w\colon W \to 2^W$ is injective. Then no map $h\colon W \to Z$ makes (4.12) commute, since w is injective but h is not.

On the positive side, we have an algebra structure $\iota_\sim\colon \Sigma(\mu\Sigma_\sim) \to \mu\Sigma_\sim$ by Theorem 4.14 and a coalgebra structure $\varsigma\colon \mu\Sigma_\sim \to B(\mu\Sigma_\sim, \mu\Sigma_\sim)$ by Lemma 4.18, and these combine to a ϱ-bialgebra:

Proposition 4.26. *In the setting of Theorem 4.14, the triple $(\mu\Sigma_\sim, \iota_\sim, \varsigma)$ is a ϱ-bialgebra.*

Proof. The outside of the diagram

$$\begin{tikzcd}
\Sigma(\mu\Sigma) \arrow[rr, "\iota"] \arrow[d, "\Sigma(\mathsf{it}\,\iota_\sim)"'] \arrow[dd, bend right=60, "\Sigma\langle \mathsf{it}\,\iota_\sim, \iota_\sim^{\clubsuit}\rangle"'] & & \mu\Sigma \arrow[d, "\mathsf{it}\,\iota_\sim"] \arrow[dd, bend left=60, "\iota_\sim^{\clubsuit}"] \\
\Sigma(\mu\Sigma_\sim) \arrow[rr, "\iota_\sim"] \arrow[d, "\Sigma\langle \mathsf{id}, \varsigma\rangle"'] & & \mu\Sigma_\sim \arrow[d, "\varsigma"] \\
\Sigma(\mu\Sigma_\sim \times B(\mu\Sigma_\sim, \mu\Sigma_\sim)) \arrow[r, "\varrho_{\mu\Sigma_\sim,\mu\Sigma_\sim}"] & B(\mu\Sigma_\sim, \Sigma^\star(\mu\Sigma_\sim + \mu\Sigma_\sim)) \arrow[r, "B(\mathsf{id}, \widehat{\iota_\sim}\cdot\Sigma^\star\nabla)"] & B(\mu\Sigma_\sim, \mu\Sigma_\sim)
\end{tikzcd}$$

commutes by definition of $\iota_\sim^{\clubsuit}$. The side cells commute by Lemma 4.18, and the top middle cell commutes by definition of $\mathsf{it}(\iota_\sim)$. Note that $\Sigma(\mathsf{it}\,\iota_\sim)$ is a coequalizer, since $\mathsf{it}(\iota_\sim)$ is a reflexive coequalizer and Σ preserves it. Hence $\Sigma(\mathsf{it}\,\iota_\sim)$ is epic, and therefore the bottom middle cell commutes, which is the ϱ-bialgebra law in question. □

The ϱ-bialgebra $(\mu\Sigma_\sim, \iota_\sim, \varsigma)$ can thus be regarded as a suitable candidate for a denotational domain in the higher-order setting, despite not being characterized by a universal property.

In conclusion, the above results firmly indicate that bialgebras remain a meaningful concept in higher-order abstract GSOS, and a potentially useful tool for deriving congruence results. For instance, in Urbat et al. [58] we have established a general congruence result with respect to *weak* (bi)similarity on operational models of higher-order GSOS laws, where the notion of *lax* higher-order bialgebra figures prominently. Additionally, recent work [24] reveals that it is possible to construct a denotational ϱ-bialgebra in a manner that parallels the first-order case.

5 The λ-calculus

We now depart from combinatory calculus and move to languages with variable binding, starting with the all-important (untyped) λ-calculus. The λ-calculus comes in various flavours, such as *call-by-name* or *call-by-value*, and the respective operational semantics can be formulated in either big-step or small-step style. For the purposes of our work, we are going to give a categorical treatment of the small-step call-by-name and the small-step call-by-value λ-calculus. We start with the former, whose operational semantics is presented in Figure 2. Here, p, p', q range over possibly open λ-terms and $[q/x]$ denotes capture-avoiding substitution of the term q for the variable x.

The operational semantics of the call-by-name λ-calculus induces a deterministic transition relation $\to$ on the set of (possibly open) λ-terms modulo α-equivalence. Note that every λ-term t either *reduces* ($t \to t'$ for some t') or is in *weak head normal form*, that is, t is a

λ-abstraction $\lambda x.t'$ or of the form $x\, s_1\, s_2\, \cdots\, s_k$ for a variable x and terms $s_1, \ldots, s_k$ ($k \geq 0$). In particular, a closed term either reduces or is a λ-abstraction. As usual, we let application associate to the left: $t_1\, t_2\, t_3\, \cdots\, t_n$ means $(\cdots((t_1\, t_2)\, t_3)\cdots)\, t_n$.

On the side of program equivalences, λ-calculus semantics can be roughly divided into three kinds: *applicative bisimilarity* [3], *normal form bisimilarity* [36] and *environmental bisimilarity* [51]. We are looking to give a coalgebraic account of *strong* versions of applicative bisimilarity, see Definition 5.15 and Proposition 5.16.

5.1 The presheaf approach to higher-order languages

Fiore et al. [17] propose the presheaf category $\mathbf{Set}^{\mathbb{F}}$ as a setting for algebraic signatures with variable binding, such as the λ-calculus and the π-calculus. We review some of the core ideas from their work as well as follow-up work by Fiore and Turi [19].

Let $\mathbb{F}$ be the category of finite cardinals, a skeleton of the category of finite sets. The objects of $\mathbb{F}$ are the sets $n = \{0, \ldots, n-1\}$ ($n \in \mathbb{N}$), and morphisms $n \to m$ are functions. The category $\mathbb{F}$ has a canonical coproduct structure

$$n \xrightarrow{\mathsf{old}_n} n+1 \xleftarrow{\mathsf{new}_n} 1 \tag{5.1}$$

where $\mathsf{old}_n(i) = i$ and $\mathsf{new}_n(0) = n$. Notice the appropriate naming of the coproduct injections: The idea is that each object $n \in \mathbb{F}$ is an untyped context of n free variables, while morphisms $n \to m$ are variable *renamings*. When extending a context along $\mathsf{old}_n(i) = i$, we understand the pre-existing elements of n as the "old" variables, and the added element $\mathsf{new}_n(0)$ as the "new" variable. The coproduct structure of $\mathbb{F}$ gives rise to three fundamental operations on contexts, namely *exchanging*, *weakening* and *contraction*:

$$\begin{aligned} \mathsf{s} &= [\mathsf{new}_1, \mathsf{old}_1] \colon 2 \to 2, \\ \mathsf{w} &= \mathsf{old}_0 \colon 0 \to 1, \\ \mathsf{c} &= [\mathsf{id}_1, \mathsf{id}_1] \colon 2 \to 1. \end{aligned} \tag{5.2}$$

We think of a presheaf $X \in \mathbf{Set}^{\mathbb{F}}$ as a collection of terms: elements of $X(n)$ are "X-terms" with free variables from the set $n = \{0, \ldots, n-1\}$, and for each $r \colon n \to m$ the map $X(r) \colon X(n) \to X(m)$ sends a term $t \in X(n)$ to the term $X(r)(t) \in X(m)$ obtained by renaming the free variables of t according to r.

Example 5.1. (1) The simplest example is the presheaf $V \in \mathbf{Set}^{\mathbb{F}}$ of variables, defined by

$$V(n) = n \qquad \text{and} \qquad V(r) = r.$$

Thus, a V-term at stage n is simply a choice of a variable $i \in n$.

(2) For every algebraic signature Σ, the presheaf $\Sigma^\star \in \mathbf{Set}^{\mathbb{F}}$ of Σ-terms is given by the domain restriction of the free monad on Σ to $\mathbb{F}$. Thus $\Sigma^\star(n)$ is the set of Σ-terms in variables from n.

(3) The presheaf $\Lambda \in \mathbf{Set}^{\mathbb{F}}$ of λ-terms is given by

$$\begin{aligned} \Lambda(n) &= \lambda\text{-terms modulo } \alpha\text{-equivalence with free variables from } n, \\ \Lambda(r)(t) &= t[r(0)/0, \ldots, r(n-1)/n-1] \qquad \text{for } r \colon n \to m, \end{aligned}$$

where $t[-/-]$ denotes simultaneous substitution in the term t.

The process of substituting terms for variables can be treated at the abstract level of presheaves as follows. For every presheaf $Y \in \mathbf{Set}^{\mathbb{F}}$, there is a functor

$$- \bullet Y \colon \mathbf{Set}^{\mathbb{F}} \to \mathbf{Set}^{\mathbb{F}}$$

given by

$$(X \bullet Y)(m) = \int^{n \in \mathbb{F}} X(n) \times (Y(m))^n = \big(\textstyle\coprod_{n \in \mathbb{F}} X(n) \times (Y(m))^n\big)/{\approx}, \tag{5.3}$$

where $\approx$ is the equivalence relation generated by all pairs

$$(x, y_0, \dots, y_{n-1}) \approx (x', y_0', \dots, y_{k-1}')$$

such that $(x, y_0, \dots, y_{n-1}) \in X(n) \times Y(m)^n$, $(x', y_0', \dots, y_{k-1}') \in X(k) \times Y(m)^k$ and there exists $r \colon n \to k$ satisfying $x' = X(r)(x)$ and $y_i = y_{r(i)}'$ for $i = 0, \dots, n-1$. An equivalence class in $(X \bullet Y)(m)$ can be thought of as a term $x \in X(n)$ with n free variables, together with n terms $y_0, \dots, y_{n-1} \in Y(m)$ to be substituted for them. The above equivalence relation then says that the choice of a term $x \in X(n)$ and a tuple $y_0, \dots, y_{n-1} \in Y(m)^n$ should be compatible with renamings: for instance, if $X = Y = \Lambda$ and $s \in \Lambda(1) \subseteq \Lambda(2)$, then for all $t, u \in \Lambda(0)$, the renaming $r \colon 1 \to 2$ with $r(0) = 1$ witnesses that $(s\,s, t) \approx (s\,s, u, t)$.

Varying Y, one obtains the *substitution tensor*

$$- \bullet - \colon \mathbf{Set}^{\mathbb{F}} \times \mathbf{Set}^{\mathbb{F}} \to \mathbf{Set}^{\mathbb{F}},$$

which makes $\mathbf{Set}^{\mathbb{F}}$ into a (non-symmetric) closed monoidal category with unit V, the presheaf of variables. Monoids in $(\mathbf{Set}^{\mathbb{F}}, \bullet, V)$ can be seen as collections of terms equipped with a substitution structure. Closure of $(\mathbf{Set}^{\mathbb{F}}, \bullet, V)$ is witnessed by the fact that for every $Y \in \mathbf{Set}^{\mathbb{F}}$ the functor $- \bullet Y \colon \mathbf{Set}^{\mathbb{F}} \to \mathbf{Set}^{\mathbb{F}}$ has a right adjoint[1] given by

$$\langle\!\langle Y, - \rangle\!\rangle \colon \mathbf{Set}^{\mathbb{F}} \to \mathbf{Set}^{\mathbb{F}}, \qquad \langle\!\langle Y, W \rangle\!\rangle(n) = \int_{m \in \mathbb{F}} [(Y(m))^n, W(m)] = \mathsf{Nat}(Y^n, W).$$

An element of $\langle\!\langle Y, W \rangle\!\rangle(n)$, viz. a natural family of maps $Y(m)^n \to W(m)$ ($m \in \mathbb{F}$), is thought of as describing the substitution of Y-terms in m variables for the n variables of some fixed ambient term, resulting in a W-term in m variables.

5.2 Syntax

Variable binding is captured by the *context extension* endofunctor

$$\delta \colon \mathbf{Set}^{\mathbb{F}} \to \mathbf{Set}^{\mathbb{F}}$$

defined on objects by

$$\delta X(n) = X(n+1) \qquad \text{and} \qquad \delta X(h) = X(h + \mathsf{id}_1)$$

and on morphisms $h \colon X \to Y$ by

$$(\delta h)_n = \big(X(n+1) \xrightarrow{h_{n+1}} Y(n+1)\big).$$

[1] Fiore et al. [17] denote the right adjoint by $\langle Y, - \rangle$; we employ the double bracket notation $\langle\!\langle Y, - \rangle\!\rangle$ instead for distinction with morphisms into products.

Informally, the elements of $\delta X(n)$ are terms arising by binding the last variable in an X-term with $n+1$ free variables. The operations $\mathsf{s}, \mathsf{w}, \mathsf{c}$ on contexts, see (5.2), give rise to natural transformations

$$\mathsf{swap}\colon \delta^2 \to \delta^2, \qquad \mathsf{up}\colon \mathrm{Id} \to \delta \qquad \text{and} \qquad \mathsf{contract}\colon \delta^2 \to \delta$$

in $\mathbf{Set}^{\mathbb{F}}$, which correspond respectively to the actions of swapping the two "newest" variables in a term, weakening and contraction. Their components are defined by

$$\begin{aligned} \mathsf{swap}_{X,n} &= \big(X(n+2) \xrightarrow{X(\mathsf{id}_n + \mathsf{s})} X(n+2) \big), \\ \mathsf{up}_{X,n} &= \big(X(n) \xrightarrow{X(\mathsf{id}_n + \mathsf{w})} X(n+1) \big), \\ \mathsf{contract}_{X,n} &= \big(X(n+2) \xrightarrow{X(\mathsf{id}_n + \mathsf{c})} X(n+1) \big). \end{aligned} \tag{5.4}$$

The presheaf V of variables (Example 5.1(1)) and the endofunctor δ are the two main constructs that enable the categorical modeling of syntax with variable binding. For example, the binding signature of the λ-calculus corresponds to the endofunctor

$$\Sigma\colon \mathbf{Set}^{\mathbb{F}} \to \mathbf{Set}^{\mathbb{F}}, \qquad \Sigma X = V + \delta X + X \times X. \tag{5.5}$$

This is analogous to algebraic signatures determining (polynomial) endofunctors on $\mathbf{Set}$. For Σ as in (5.5), the forgetful functor $\mathbf{Alg}(\Sigma) \to \mathbf{Set}^{\mathbb{F}}$ has a left adjoint that takes a presheaf $X \in \mathbf{Set}^{\mathbb{F}}$ to the free Σ-algebra $\Sigma^{\star} X$. In particular, the initial algebra $\mu\Sigma$ is given by the presheaf Λ of λ-terms; more precisely:

Proposition 5.2. *The initial algebra for Σ is given by*

$$V + \delta\Lambda + \Lambda \times \Lambda \xrightarrow{[\mathsf{var}, \lambda.(-), \mathsf{app}]} \Lambda$$

where $\mathsf{var}\colon V \to \Lambda$ is the inclusion of variables, $\lambda.(-)$ sends $t \in \delta\Lambda(n) = \Lambda(n+1)$ to $\lambda n+1.t$, and app sends $(t,s) \in \Lambda(n) \times \Lambda(n)$ to $t\,s$.

The substitution tensor $\bullet$ gives rise to the expected substitution structure on λ-terms:

Proposition 5.3. *The presheaf $\Lambda = \mu\Sigma$ of λ-terms admits the structure of a monoid (Λ, μ, η) in $(\mathbf{Set}^{\mathbb{F}}, \bullet, V)$ whose unit $\eta\colon V \to \Lambda$ is the inclusion of variables and whose multiplication $\mu\colon \Lambda \bullet \Lambda \to \Lambda$ is the uncurried natural transformation $\bar{\mu}\colon \Lambda \to \langle\!\langle \Lambda, \Lambda \rangle\!\rangle$ given by*

$$\bar{\mu}_n\colon \Lambda(n) \to \langle\!\langle \Lambda, \Lambda \rangle\!\rangle(n) = \mathsf{Nat}(\Lambda^n, \Lambda), \qquad t \mapsto \lambda \vec{u} \in \Lambda(m)^n.\, t[\vec{u}].$$

Here, $t[\vec{u}]$ denotes the simultaneous substitution $t[u_0/0, \dots, u_{n-1}/n-1]$.

5.3 Behaviour

To capture the λ-calculus in the abstract categorical setting of higher-order GSOS laws developed in Section 4, we consider the behaviour bifunctor

$$B\colon (\mathbf{Set}^{\mathbb{F}})^{\mathsf{op}} \times \mathbf{Set}^{\mathbb{F}} \to \mathbf{Set}^{\mathbb{F}}, \qquad B(X,Y) = \langle\!\langle X, Y \rangle\!\rangle \times (Y + Y^X + 1), \tag{5.6}$$

where Y^X denotes the exponential object in the topos $\mathbf{Set}^{\mathbb{F}}$.

Remark 5.4. Exponentials in presheaf categories have a simple explicit description [38, Sec. I.6]. The exponential Y^X in $\mathbf{Set}^{\mathbb{F}}$ and its evaluation morphism $\mathsf{ev}\colon Y^X \times X \to Y$ are, respectively, given by

$$Y^X(n) = \mathsf{Nat}((-)^n \times X, Y) \qquad \text{and} \qquad \mathsf{ev}_n(f, x) = f_n(\mathsf{id}_n, x) \in Y(n)$$

for every natural transformation $f\colon (-)^n \times X \to Y$ and $x \in X(n)$. In the following we put

$$f(x) \;:=\; \mathsf{ev}_n(f,x).$$

Our intended operational model of the λ-calculus is a $B(\Lambda,-)$-coalgebra

$$\langle \gamma_1, \gamma_2\rangle\colon \Lambda \to \langle\!\langle \Lambda,\Lambda\rangle\!\rangle \times (\Lambda + \Lambda^\Lambda + 1) \tag{5.7}$$

on the presheaf of λ-terms. For each term $t \in \Lambda(n)$, the natural transformation $\gamma_1(t)\colon \Lambda^n \to \Lambda$ exposes the simultaneous substitution structure, that is, $\gamma_1(t)$ is equal to $\bar{\mu}(t)$ from Proposition 5.3. Similarly, $\gamma_2(t)$ is an element of the coproduct $\Lambda(n) + \Lambda^\Lambda(n) + 1$, representing either a reduction step, a λ-abstraction seen as a function on terms, or that t is stuck. To apply the higher-order abstract GSOS framework, let us first note that one of its key assumptions holds:

Lemma 5.5. *For every $X \in \mathbf{Set}^{\mathbb{F}}$ the functor $B(X,-)$ has a final coalgebra.*

Proof. The functor $B(X,-)$ preserves limits of ω^{op}-chains: the right adjoints $\langle\!\langle X,-\rangle\!\rangle$ and $(-)^X$ preserve all limits, and limits of ω^{op}-chains commute with products and coproducts in $\mathbf{Set}$ and thus in $\mathbf{Set}^{\mathbb{F}}$ (using that limits and colimits in presheaf categories are formed pointwise). Therefore, dually to the classic result by Adámek [5], the functor $B(X,-)$ has a final coalgebra computed as the limit of the final ω^{op}-chain

$$1 \leftarrow B(X,1) \leftarrow B(X,B(X,1)) \leftarrow B(X,B(X,B(X,1))) \leftarrow \cdots \qquad \square$$

Since $B(X,-)$ preserves pullbacks, and hence weakly preserves pullbacks, behavioural equivalence (4.7) on $B(X,-)$-coalgebras coincides with coalgebraic bisimilarity (see [47]). Recall that a *bisimulation* between $B(X,-)$-coalgebras $W \to B(X,W)$ and $Z \to B(X,Z)$ is a relation between W and Z, i.e. a sub-presheaf $R \subseteq W \times Z$, that can be equipped with a coalgebra structure $r\colon R \to B(X,R)$ such that the projections $p_1\colon R \to W$ and $p_2\colon R \to Z$ are $B(X,-)$-coalgebra morphisms.

$$\begin{tikzcd}
W \arrow[d, "w"'] & R \arrow[l, "p_1"'] \arrow[r, "p_2"] \arrow[d, dashed, "r"] & Z \arrow[d, "z"] \\
B(X,W) & B(X,R) \arrow[l, "B(X,p_1)"'] \arrow[r, "B(X,p_2)"] & B(X,Z)
\end{tikzcd}$$

The next proposition gives an elementary characterization of bisimulations.

Proposition 5.6. *Given $X \in \mathbf{Set}^{\mathbb{F}}$ and two $B(X,-)$-coalgebras*

$$\langle c_1, c_2\rangle\colon W \to \langle\!\langle X,W\rangle\!\rangle \times (W + W^X + 1) \quad \text{and} \quad \langle d_1, d_2\rangle\colon Z \to \langle\!\langle X,Z\rangle\!\rangle \times (Z + Z^X + 1),$$

a family of relations $R(n) \subseteq W(n) \times Z(n)$, $n \in \mathbb{F}$, is a bisimulation if and only if for all $n \in \mathbb{F}$ and $(w,z) \in R(n)$ the following conditions hold (omitting subscripts of components of the natural transformations c_i, d_i):

(1) *$(W(r)(w), Z(r)(z)) \in R(m)$ for all $r\colon n \to m$;*

(2) *$(c_1(w)(\vec{u}), d_1(z)(\vec{u})) \in R(m)$ for all $m \in \mathbb{F}$ and $\vec{u} \in X(m)^n$;*

(3) *$c_2(w) = w' \in W(n) \implies d_2(z) = z' \in Z(n) \wedge (w',z') \in R(n)$;*

(4) *$c_2(w) = f \in W^X(n) \implies d_2(z) = g \in Z^X(n) \wedge \forall e \in X(n).\,(f(e), g(e)) \in R(n)$;*

(5) *$c_2(w) = * \implies d_2(z) = *$;*

(6) *$d_2(z) = z' \in Z(n) \implies c_2(w) = w' \in W(n) \wedge (w',z') \in R(n)$;*

(7) $d_2(z) = g \in Z^X(n) \implies c_2(w) = f \in W^X(n) \wedge \forall e \in X(n).\ (f(e), g(e)) \in R(n)$;

(8) $d_2(z) = * \implies c_2(w) = *$.

Before proving the proposition, let us elaborate on the conditions (1)–(8). Condition (1) states that $B(X,-)$-bisimulations are compatible with the renaming of free variables: given a renaming $r\colon n \to m$, the renamed terms $W(r)(w)$ and $Z(r)(z)$ are related by $R(m)$. Similarly, condition (2) states that $B(X,-)$-bisimulations are compatible with substitutions: given a substitution $\vec{u} \in X(m)^n$, the resulting terms $c_1(w)(\vec{u})$ and $d_1(z)(\vec{u})$ are related by $R(m)$. Conditions (6)–(8) are symmetric to (3)–(5); in fact, since $B(X,-)$-coalgebras are deterministic transition systems, conditions (6)–(8) are implied by (3)–(5) and hence could be dropped. We opted to state (6)–(8) explicitly, as these conditions become relevant in nondeterministic extensions of the λ-calculus.

Proof of Proposition 5.6. For the $\implies$ direction, suppose that R is a bisimulation, i.e. there exists a coalgebra structure $\langle r_1, r_2 \rangle$ on R making the diagram below commute, where p_1, p_2 are the projections:

$$\begin{tikzcd}
W \arrow[d, "\langle c_1, c_2 \rangle"'] & R \arrow[l, "p_1"'] \arrow[r, "p_2"] \arrow[d, "\langle r_1, r_2 \rangle"] & Z \arrow[d, "\langle d_1, d_2 \rangle"] \\
\langle\!\langle X, W \rangle\!\rangle \times (W + W^X + 1) & \langle\!\langle X, R \rangle\!\rangle \times (R + R^X + 1) \arrow[l, "\langle\!\langle X, p_1 \rangle\!\rangle \times (p_1 + p_1^X + 1)"'] \arrow[r, "\langle\!\langle X, p_2 \rangle\!\rangle \times (p_2 + p_2^X + 1)"] & \langle\!\langle X, Z \rangle\!\rangle \times (Z + Z^X + 1)
\end{tikzcd}$$

Then (1) holds because R is a sub-presheaf of $W \times Z$, and (2), (3) and (5) are immediate from the above diagram. Concerning (4), let $w\, R(n)\, z$ and suppose that $c_2(w) = f \in W^X(n)$. Then the above diagram implies that $r_2(w, z) = h \in R^X(n)$ and $d_2(z) = g \in Z^X(n)$. Moreover, for all $e \in X(n)$, using infix notation for the binary relation $R(n)$, we have that

$$f(e) = \mathsf{ev}(f, e) = p_1(\mathsf{ev}(h, e))\ \ R(n)\ \ p_2(\mathsf{ev}(h, e)) = \mathsf{ev}(g, e) = g(e)$$

where the second and the penultimate equality follow via naturality of ev.

For the $\Longleftarrow$ direction, suppose that $R(n) \subseteq W(n) \times Z(n)$, $n \in \mathbb{F}$, is a family of relations satisfying (1)–(8) for all $w\, R(n)\, z$. Condition (1) asserts that R is a sub-presheaf of $W \times Z$; thus it remains to define a coalgebra structure $\langle r_1, r_2 \rangle$ on R making the diagram above commute. It suffices to define the components

$$\langle r_{1,n}, r_{2,n} \rangle \colon R(n) \to \langle\!\langle X, R \rangle\!\rangle(n) \times (R(n) + R^X(n) + 1)$$

and prove that the diagram commutes pointwise at every $n \in \mathbb{F}$; the naturality of r_1, r_2 then follows since the two lower horizontal maps in the diagram are jointly monomorphic. Indeed, this is easy to see using the following diagram, where $f\colon n \to m$ is a morphism of $\mathbb{F}$ and we abbreviate $F = B(X, -)$, $r = \langle r_1, r_2 \rangle$ and $c = \langle c_1, c_2 \rangle$:

$$
\begin{tikzcd}
W(n) \arrow[rrr, "c_n"] \arrow[ddd, "Wf"'] & & & FW(n) \arrow[ddd, "FW(f)"] \\
& R(n) \arrow[ul, "p_{1,n}"'] \arrow[r, "r_n"] \arrow[d, "R(f)"'] & FR(n) \arrow[ur, "Fp_{1,n}"] \arrow[d, "FR(f)"] & \\
& R(m) \arrow[dl, "p_{1,m}"'] \arrow[r, "r_m"] & FR(m) \arrow[dr, "Fp_{1,m}"] & \\
W(m) \arrow[rrr, "c_m"] & & & FW(m)
\end{tikzcd}
$$

Since its outside and the upper, lower, right and left-hand parts commute due to the naturality of c and p_1 so does the desired inner square when postcomposed by $Fp_{1,m}$. A similar diagram using $(Z, d = \langle d_1, d_2 \rangle)$ and p_2 in lieu of (W, c) and p_1 shows that the desired square commutes when postcomposed by $Fp_{2,m}$. So since $Fp_{1,m}, Fp_{2,m}$ is jointly monic, the desired square commutes.

We define

$$r_{1,n} \colon R(n) \to \langle\!\langle X, R \rangle\!\rangle(n) = \mathsf{Nat}(X^n, R) \qquad \text{by} \qquad r_{1,n}(w, z) = \langle c_{1,n}(w), d_{1,n}(z) \rangle.$$

Condition (2) shows that this map is well-typed and that it makes the first component of the diagram commute.

To define $r_{2,n}$, using extensivity of the presheaf topos $\mathbf{Set}^{\mathbb{F}}$ we express R as a coproduct $R = R_0 + R_1 + R_2$ of the sub-presheaves given by

$$
\begin{aligned}
R_0(n) &= \{ (w, z) \in R(n) : c_2(w) \in W(n),\ d_2(z) \in Z(n) \}, \\
R_1(n) &= \{ (w, z) \in R(n) : c_2(w) \in W^X(n),\ d_2(z) \in Z^X(n) \}, \\
R_2(n) &= \{ (w, z) \in R(n) : c_2(w) = *,\ d_2(z) = * \}.
\end{aligned}
$$

Thus, it suffices to define $r_{2,n} \colon R_0(n) + R_1(n) + R_2(n) \to R(n) + R^X(n) + 1$ separately for each summand of its domain. Given $(w, z) \in R_0(n)$, we put

$$r_{2,n}(w, z) = (c_2(w), d_2(z)) \in R(n).$$

By condition (3), this is well-typed and makes the second component of the diagram (with domain R restricted to R_0) commute. Similarly, for $(w, z) \in R_2(n)$ we put

$$r_{2,n}(w, z) = *;$$

the second component of the diagram (with domain R restricted to R_2) then commutes by condition (5). Finally, for $(w, z) \in R_1(n)$ we put

$$r_{2,n}(w, z) = \mathsf{curry}\ h(w, z) \in R^X(n)$$

where $h \colon R_1 \times X \to R$ is the natural transformation whose component at $m \in \mathbb{F}$ is given by

$$h_m((w', z'), e) = (c_2(w')(e), d_2(z')(e)) \in R(m).$$

Condition (4) asserts that h_m is well-typed and that the second component of the diagram (with domain R restricted to R_1) commutes. $\square$

5.4 Semantics

As explained above, in our intended operational model $\langle\gamma_1,\gamma_2\rangle\colon \Lambda \to B(\Lambda,\Lambda)$ for the λ-calculus, the component γ_1 should be the transpose of the monoid multiplication $\mu\colon \Lambda\bullet\Lambda \to \Lambda$ from Proposition 5.3 under the adjunction $-\bullet\Lambda \dashv \langle\!\langle \Lambda, -\rangle\!\rangle$. As an interesting technical subtlety, for this model to be induced by a higher-order GSOS law $\varrho = (\varrho_{X,Y})$ of some sort, the argument X is required to be equipped with a *point* $\mathsf{var}\colon V \to X$. The importance of points for defining substitution was first identified by [17] (see also [16]) and is worth recalling from its original source.

Fiore et al. argued that, given an endofunctor $F\colon \mathbf{Set}^{\mathbb{F}} \to \mathbf{Set}^{\mathbb{F}}$, in order to define a substitution structure $F^\star V \bullet F^\star V \to F^\star V$ on the free F-algebra over V, it is necessary for F to be *tensorially strong*, in that there is a natural transformation $\mathsf{st}_{X,Y}\colon FX \bullet Y \to F(X \bullet Y)$ satisfying the expected coherence laws [17, §3]. For the special case of F being the context extension endofunctor δ, this requires the presheaf Y to be equipped with a *point* $\mathsf{var}\colon V \to Y$: the strength map $\mathsf{st}_{X,Y}\colon \delta X \bullet Y \to \delta(X \bullet Y)$ is given at $m \in \mathbb{F}$ by

$$[t \in X(n+1), \vec{u} \in Y(m)^n] \quad \xmapsto{\mathsf{st}_{X,Y}} \quad [t, (\mathsf{up}_{Y,m}(\vec{u}), \mathsf{var}_{m+1}(\mathsf{new}_m)) \in Y(m+1)^{n+1}];$$

where $[-]$ are equivalence classes of the equivalence relation $\approx$ appearing in the definition (5.3) of $\bullet$. Intuitively, given a substitution of length n on a term with $n+1$ free variables, a fresh variable in Y should be used to (sensibly) produce a substitution of length $n+1$. This situation is relevant in the context of higher-order GSOS laws of binding signatures over $B(X,Y)$ where, e.g. in the case of the λ-calculus, one is asked to define a map of the form (factoring out the unnecessary parts)

$$\varrho_1\colon \delta\langle\!\langle X,Y\rangle\!\rangle \to \langle\!\langle X, \delta Y\rangle\!\rangle. \tag{5.8}$$

Writing $\overline{\mathsf{st}}$ for the transpose of st under $-\bullet X \dashv \langle\!\langle X,-\rangle\!\rangle$, we obtain ϱ_1 as the composite

$$\delta\langle\!\langle X,Y\rangle\!\rangle \xrightarrow{\overline{\mathsf{st}}_{\langle\!\langle X,Y\rangle\!\rangle,X}} \langle\!\langle X, \delta(\langle\!\langle X,Y\rangle\!\rangle \bullet X)\rangle\!\rangle \xrightarrow{\langle\!\langle X,\delta(\varepsilon)\rangle\!\rangle} \langle\!\langle X,\delta Y\rangle\!\rangle,$$

where $\varepsilon\colon \langle\!\langle X,Y\rangle\!\rangle \bullet X \to Y$ is the evaluation morphism for the hom-object $\langle\!\langle X,Y\rangle\!\rangle$. In elementary terms, the map ϱ_1 takes a natural transformation $f\colon X^{n+1} \to Y$ to the natural transformation $\varrho_1(f)\colon X^n \to \delta Y$ given by

$$\vec{u} \in X(m)^n \quad \mapsto \quad f_{m+1}(\mathsf{up}_{X,m}(\vec{u}), \mathsf{var}_{m+1}(\mathsf{new}_m)) \in Y(m+1). \tag{5.9}$$

Thus ϱ_1 represents capture-avoiding simultaneous substitution in which the freshest variable is bound, hence it should not be substituted.

At the same time, a higher-order GSOS law for the λ-calculus needs to turn a λ-abstraction into a function on potentially open terms precisely by only substituting the bound variable. This implies that we need a natural transformation of the form

$$\varrho_2\colon \delta\langle\!\langle X,Y\rangle\!\rangle \to Y^X. \tag{5.10}$$

Again, we make use of the point $\mathsf{var}\colon V \to X$ to produce ϱ_2:

$$\delta\langle\!\langle X,Y\rangle\!\rangle \xrightarrow{\cong} \langle\!\langle X, Y^X\rangle\!\rangle \xrightarrow{\langle\!\langle \mathsf{var}, Y^X\rangle\!\rangle} \langle\!\langle V, Y^X\rangle\!\rangle \xrightarrow{\cong} Y^X.$$

Here, the first isomorphism is given at $n \in \mathbb{F}$ by

$$\delta\langle\!\langle X,Y\rangle\!\rangle(n) = \mathsf{Nat}(X^{n+1}, Y) \cong \mathsf{Nat}(X^n, Y^X) = \langle\!\langle X, Y^X\rangle\!\rangle(n).$$

Thus, in elementary terms, the adjoint transpose $\overline{\varrho_2}$ of ϱ_2 acts as follows:

$$\overline{\varrho_2}(f) = \lambda e \in X(n).\, f_n(\mathsf{var}_n(0), \dots, \mathsf{var}_n(n-1), e) \qquad \text{for } f\colon X^{n+1} \to Y.$$

With these preparations at hand, we are now ready to define the small-step operational semantics of the call-by-name λ-calculus in terms of a V-pointed higher-order GSOS law of the syntax endofunctor $\Sigma X = V + \delta X + X \times X$ over the behaviour bifunctor $B(X,Y) = \langle\!\langle X, Y \rangle\!\rangle \times (Y + Y^X + 1)$. A law of this type is given by a family of presheaf maps

$$\begin{array}{c} V + \delta(X \times \langle\!\langle X,Y \rangle\!\rangle \times (Y + Y^X + 1)) + (X \times \langle\!\langle X,Y \rangle\!\rangle \times (Y + Y^X + 1))^2 \\ \big\downarrow \varrho_{X,Y} \\ \langle\!\langle X, \Sigma^\star(X+Y) \rangle\!\rangle \times (\Sigma^\star(X+Y) + (\Sigma^\star(X+Y))^X + 1) \end{array}$$

dinatural in $(X, \mathsf{var}_X) \in V/\mathbf{Set}^{\mathbb{F}}$ and natural in $Y \in \mathbf{Set}^{\mathbb{F}}$. We let $\varrho_{X,Y,n}$ denote the component of $\varrho_{X,Y}$ at $n \in \mathbb{F}$. For the definition of ϱ we set up some notation:

Notation 5.7. We write

$$\lambda.(-)\colon \delta\Sigma^\star \to \Sigma^\star \qquad \text{and} \qquad \mathsf{app}\colon \Sigma^\star \times \Sigma^\star \to \Sigma^\star$$

for the natural transformations whose components come from the Σ-algebra structure on free Σ-algebras; here app denotes application. In the following we will consider free algebras of the form $\Sigma^\star(X+Y)$. For simplicity, we usually keep inclusion maps implicit: Given $t_1, t_2 \in X(n)$ and $t_1' \in Y(n)$ we write $t_1\, t_2$ for $\mathsf{app}([\eta \cdot \mathsf{inl}(t_1)], [\eta \cdot \mathsf{inl}(t_2)])$, and similarly $t_1\, t_1'$ for $\mathsf{app}([\eta \cdot \mathsf{inl}(t_1)], [\eta \cdot \mathsf{inr}(t_1')])$ etc., where inl and inr are the coproduct injections and $\eta\colon \mathrm{Id} \to \Sigma^\star$ is the unit of the free monad $\Sigma^\star$.

Notation 5.8. Let

$$\pi\colon V \to \langle\!\langle X, \Sigma^\star(X+Y) \rangle\!\rangle$$

be the adjoint transpose of

$$V \bullet X \xrightarrow{\cong} X \xrightarrow{\mathsf{inl}} X + Y \xrightarrow{\eta} \Sigma^\star(X+Y).$$

Thus, for $v \in V(n) = n$, the natural transformation $\pi(v)(n)\colon X^n \to \Sigma^\star(X+Y)$ is the v-th projection $X^n \to X$ followed by $\eta \cdot \mathsf{inl}$. Further, recall that $j\colon V/\mathbf{Set}^{\mathbb{F}} \to \mathbf{Set}^{\mathbb{F}}$ denotes the forgetful functor.

Definition 5.9 (V-pointed higher-order GSOS law for the call-by-name λ-calculus)**.**

$$\varrho^{\mathrm{cn}}_{X,Y}\colon \Sigma(jX \times B(jX,Y)) \to B(jX, \Sigma^\star(jX+Y))$$

is given by

$$\begin{array}{lll} \multicolumn{3}{l}{\varrho^{\mathrm{cn}}_{X,Y,n}(tr) = \mathtt{case}\ tr\ \mathtt{of}} \\ v \in V(n) & \mapsto & \pi(v), * \\ \lambda.(t, f, \_) & \mapsto & \langle\!\langle X, \lambda.(-) \cdot \eta \cdot \mathsf{inr} \rangle\!\rangle(\varrho_1(f)), (\eta \cdot \mathsf{inr})^X(\varrho_2(f)) \\ (t_1, g, t_1')\,(t_2, h, \_) & \mapsto & \lambda m, \vec{u} \in X(m)^n.\, (g_m(\vec{u})\, h_m(\vec{u})), t_1'\, t_2 \\ (t_1, g, k)\,(t_2, h, \_) & \mapsto & \lambda m, \vec{u} \in X(m)^n.\, (g_m(\vec{u})\, h_m(\vec{u})), \eta \cdot \mathsf{inr} \cdot k(t_2) \\ (t_1, g, *)\,(t_2, h, \_) & \mapsto & \lambda m, \vec{u} \in X(m)^n.\, (g_m(\vec{u})\, h_m(\vec{u})), * \end{array}$$

where $t \in \delta X(n)$, $f \in \delta\langle\!\langle X, Y \rangle\!\rangle(n)$, $g, h \in \langle\!\langle X, Y \rangle\!\rangle(n)$, $k \in Y^X(n)$, $t_1, t_2 \in X(n)$ and $t_1' \in Y(n)$.

$$
\texttt{var}\,\frac{}{v \not\to} \qquad \texttt{lam}\,\frac{\vec{u} = (\mathsf{var}_n(0), \ldots, \mathsf{var}_n(n-1), e \in X(n)) \qquad t[\vec{u}] = t'}{\lambda.t \xrightarrow{e} t'}
$$

$$
\texttt{app1}\,\frac{t_1 \to t_1'}{t_1\, t_2 \to t_1'\, t_2} \qquad \texttt{app2}\,\frac{t_1 \xrightarrow{t_2} t_1'}{t_1\, t_2 \to t_1'} \qquad \texttt{app3}\,\frac{t_1 \not\to}{t_1\, t_2 \not\to}
$$

$$
\texttt{varSub}\,\frac{}{v[\vec{u}] = \vec{u}(v)} \qquad \texttt{lamSub}\,\frac{\vec{w} = (\mathsf{up}_{X,m}(\vec{u}), \mathsf{var}_{m+1}(\mathsf{new}_m)) \qquad t[\vec{w}] = t''}{(\lambda.t)[\vec{u}] = \lambda.t''}
$$

$$
\texttt{appSub}\,\frac{t_1[\vec{u}] = t_1' \qquad t_2[\vec{u}] = t_2'}{(t_1\, t_2)[\vec{u}] = t_1'\, t_2'}
$$

Fig. 3. Law ϱ^{cn} in the form of inference rules.

Remark 5.10. We have omitted brackets around the pairs on the right. The last three clauses refer to the application operator $p\, q = \mathsf{app}(p, q)$ and could also be written as, e.g.,

$$
\mathsf{app}((t_1, g, t_1'), (t_2, h, \_)) \quad \mapsto \quad \lambda m,\, \vec{u} \in X(m)^n.\, \mathsf{app}(g_m(\vec{u}), h_m(\vec{u})), \mathsf{app}(t_1', t_2).
$$

In Figure 3 the definition of ϱ^{cn} is rephrased in the style of inference rules. Here, $[\ldots]$ corresponds to the first component of $B(jX, Y)$, and $\to$, $\xrightarrow{t}$, $\not\to$ correspond to the slots in the second component of $B(jX, Y)$. For instance, the rule `lam` expresses that for every $t \in \delta X(n)$, $f \in \delta\langle\!\langle X, Y\rangle\!\rangle$ and $e \in X(n)$, putting $\vec{u} = (\mathsf{var}_n(0), \ldots, \mathsf{var}_n(n-1), e)$ and $t' = f(\vec{u})$, the second component of $\varrho_{X,Y,n}(\lambda.(t, f, \_))$ lies in $Y^X(n)$ and satisfies $\varrho_{X,Y,n}(\lambda.(t, f, \_))(e) = t'$. This matches precisely the corresponding clause of Definition 5.9.

Remark 5.11. Instantiating Definition 4.7, the operational model of the higher-order GSOS law ϱ^{cn} is the $B(\Lambda, -)$-coalgebra

$$
\iota^{\clubsuit} = \langle \gamma_1, \gamma_2 \rangle \colon \Lambda \to \langle\!\langle \Lambda, \Lambda \rangle\!\rangle \times (\Lambda + \Lambda^{\Lambda} + 1), \tag{5.11}
$$

that is uniquely determined by the following commutative diagram:

$$
\begin{tikzcd}
V + \delta\Lambda + \Lambda^2 \arrow[r, "\mathsf{id} + \delta\langle \mathsf{id}, \gamma_1, \gamma_2\rangle + \langle \mathsf{id}, \gamma_1, \gamma_2\rangle^2"] \arrow[ddd, "\iota = {[\mathsf{var}, \lambda.(-), \mathsf{app}]}"'] & V + \delta(\Lambda \times \langle\!\langle \Lambda, \Lambda \rangle\!\rangle \times (\Lambda + \Lambda^{\Lambda} + 1)) + (\Lambda \times \langle\!\langle \Lambda, \Lambda \rangle\!\rangle \times (\Lambda + \Lambda^{\Lambda} + 1))^2 \arrow[d, "\varrho^{\mathrm{cn}}_{\Lambda,\Lambda}"] \\
 & \langle\!\langle \Lambda, \Sigma^{\star}(\Lambda + \Lambda) \rangle\!\rangle \times (\Sigma^{\star}(\Lambda + \Lambda) + (\Sigma^{\star}(\Lambda + \Lambda))^{\Lambda} + 1) \arrow[d, "\langle\!\langle \mathsf{id}, \Sigma^{\star}\nabla \rangle\!\rangle \times (\Sigma^{\star}\nabla + (\Sigma^{\star}\nabla)^{\Lambda} + \mathsf{id})"] \\
 & \langle\!\langle \Lambda, \Sigma^{\star}\Lambda \rangle\!\rangle \times (\Sigma^{\star}\Lambda + (\Sigma^{\star}\Lambda)^{\Lambda} + 1) \arrow[d, "\langle\!\langle \mathsf{id}, \hat{\iota} \rangle\!\rangle \times (\hat{\iota} + \hat{\iota}^{\Lambda} + \mathsf{id})"] \\
\Lambda \arrow[r, "\langle \gamma_1, \gamma_2 \rangle"] & \langle\!\langle \Lambda, \Lambda \rangle\!\rangle \times (\Lambda + \Lambda^{\Lambda} + 1)
\end{tikzcd}
$$

The following two propositions assert that the coalgebra $\langle \gamma_1, \gamma_2 \rangle$ coincides with the intended operational model described in (5.7), that is, its first component exposes the substitution structure of λ-terms and its second component corresponds to the transition system $\to$ on λ-terms derived from the operational semantics in Figure 2.

Proposition 5.12. *For every $m, n \in \mathbb{F}$, $t \in \Lambda(n)$ and $\vec{u} \in \Lambda(m)^n$, we have*

$$\gamma_1(t)(\vec{u}) = t[\vec{u}].$$

Proof. We proceed by induction on the structure of t.

- For $t = v \in V(n)$,
$$\gamma_1(v)(\vec{u}) = \pi(v)(\vec{u}) = u_v = v[\vec{u}];$$
the first equality follows from the definition of γ_1 (Remark 5.11), the second one from the definition of π, and the third one from the definition of substitution.
- For $t = \lambda x.t'$ (where $x = n$ and $t' \in \Lambda(n+1)$),
$$\gamma_1(t)(\vec{u}) = \lambda m.(\gamma_1(t')(\mathsf{up}_{\Lambda,m}(\vec{u}), m)) = \lambda m.(t'[\mathsf{up}_{\Lambda,m}(\vec{u}), m]) = t[\vec{u}];$$
the first equality uses the definition of γ_1 in terms of ϱ_1 (5.9), the second one follows by induction and the third one by the definition of substitution for the case of λ-abstraction.
- For $t = t_1\, t_2$,
$$\gamma_1(t)(\vec{u}) = \gamma_1(t_1)(\vec{u})\, \gamma_1(t_2)(\vec{u}) = t_1[\vec{u}]\, t_2[\vec{u}] = t[\vec{u}];$$
the first equality follows from the definition of γ_1, the second one by induction, and the third one by the definition of substitution. ◻

Proposition 5.13. *For every $n \in \mathbb{F}$ and $t \in \Lambda(n)$ the following statements hold:*

(1) *If $\gamma_2(t) \in \Lambda(n)$ then $t \to \gamma_2(t)$;*

(2) *If $\gamma_2(t) \in \Lambda^{\Lambda}(n)$ then $t = \lambda x.t'$ for some t', and $\gamma_2(t)(e) = t'[e/x]$ for every $e \in \Lambda(n)$;*

(3) *If $\gamma_2(t) = *$ then t is stuck, i.e. $t = x\, s_1 \cdots s_k$ for $x \in V(n)$, $k \geq 0$ and $s_1, \ldots, s_k \in \Lambda(n)$.*

Remark 5.14. Note that partial converses of the above statements are implied:

(1) If t reduces, then $\gamma_2(t) \in \Lambda(n)$.

(2) If $t = \lambda x.t'$, then $\gamma_2(t) \in \Lambda^{\Lambda}(n)$.

(3) If t is stuck, then $\gamma_2(t) = *$

For instance, if t reduces, then it can neither hold that $\gamma_2(t) \in \Lambda^{\Lambda}(n)$ or $\gamma_2(t) = *$ by Proposition 5.13(2),(3), and therefore $\gamma_2(t) \in \Lambda(n)$. Similarly for the other cases.

Proof. We proceed by induction on the structure of t:

- For $t = v \in V(n)$, we have $\gamma_2(t) = *$ by the definition of γ_2 (Remark 5.11); hence case (3) applies, and t is stuck as claimed.
- For $t = \lambda x.t'$, we have $\gamma_2(t) \in \Lambda^{\Lambda}(n)$ using the definition of ϱ^{cn}. Then case (2) applies, and for every $e \in \Lambda(n)$ we have
$$\gamma_2(t)(e) = \varrho_2(\gamma_1(t'))(e) = \gamma_1(t')(0, \ldots, n-1, e) = t'[e/x]$$
as claimed, where the first two equalities use the definition of γ_2 and ϱ_2, respectively, and the third one follows from Proposition 5.12.
- For $t = t_1\, t_2$, we distinguish three cases:
  - If $t_1 = x \in V(n)$, then $\gamma_2(t) = *$ and $t = x\, t_2$. Hence case (3) applies, and t is stuck as claimed.

- If $t_1 = \lambda x.t_1'$, we have $t \to t' := t_1'[t_2/x]$. Then case (1) applies, and

$$\gamma_2(t) = \gamma_2(t_1)(t_2) = t_1'[t_2/x] = t'$$

as claimed, where the first equality uses the definition of γ_2 and the second one follows by induction.
- If $t_1 = t_{1,1}\, t_{1,2}$, then either case (1) or (3) applies to t_1. If case (1) applies to t_1, i.e. $\gamma_2(t_1) \in \Lambda(n)$, we know by induction that $t_1 \to \gamma_2(t_1)$. By definition of $\to$ and γ_2, this implies $\gamma_2(t) \in \Lambda(n)$ and

$$t \to \gamma_2(t_1)\, t_2 = \gamma_2(t),$$

proving (1) for t. If case (3) applies to t_1, i.e. $\gamma_2(t_1) = *$, we know by induction that $t_1 = x\, s_1 \cdots s_k$ for some $x \in V(n)$ and $s_1, \ldots, s_k \in \Lambda(n)$. Then also $\gamma_2(t) = *$ and $t = x\, s_1 \cdots s_k\, t_2$, proving (3) for t. □

Let $\sim^\Lambda \subseteq \Lambda \times \Lambda$ be the bisimilarity relation on the coalgebra (5.11). It turns out that $\sim^\Lambda$ matches the open extension of strong applicative bisimilarity, cf. [3].

Definition 5.15. *Strong applicative bisimilarity* is the greatest relation $\sim_0^{\mathrm{ap}} \subseteq \Lambda(0) \times \Lambda(0)$ such that for $t_1 \sim_0^{\mathrm{ap}} t_2$ the following conditions hold:

$$t_1 \to t_1' \implies \exists t_2'.\, t_2 \to t_2' \wedge t_1' \sim_0^{\mathrm{ap}} t_2'; \tag{A1}$$
$$t_1 = \lambda x.t_1' \implies \exists t_2'.\, t_2 = \lambda x.t_2' \wedge \forall e \in \Lambda(0).\, t_1'[e/x] \sim_0^{\mathrm{ap}} t_2'[e/x]; \tag{A2}$$
$$t_2 \to t_2' \implies \exists t_1'.\, t_1 \to t_1' \wedge t_1' \sim_0^{\mathrm{ap}} t_2'; \tag{A3}$$
$$t_2 = \lambda x.t_2' \implies \exists t_1'.\, t_1 = \lambda x.t_1' \wedge \forall e \in \Lambda(0).\, t_1'[e/x] \sim_0^{\mathrm{ap}} t_2'[e/x]. \tag{A4}$$

The *open extension* of strong applicative bisimilarity is the relation $\sim^{\mathrm{ap}} \subseteq \Lambda \times \Lambda$ where $\sim_n^{\mathrm{ap}} \subseteq \Lambda(n) \times \Lambda(n)$ for $n > 0$ is given by

$$t_1 \sim_n^{\mathrm{ap}} t_2 \qquad \text{iff} \qquad t_1[\vec{u}] \sim_0^{\mathrm{ap}} t_2[\vec{u}] \quad \text{for every } \vec{u} \in \Lambda(0)^n.$$

Proposition 5.16. *Coalgebraic bisimilarity coincides with the open extension of strong applicative bisimilarity:*

$$\sim^\Lambda \; = \; \sim^{\mathrm{ap}}.$$

Proof. By Propositions 5.6, 5.12 and 5.13, bisimilarity is the greatest relation $\sim^\Lambda \subseteq \Lambda \times \Lambda$ such that for every $n \in \mathbb{F}$ and $t_1 \sim_n^\Lambda t_2$ the following conditions hold:

(1) $t_1[r(0), \ldots, r(n-1)] \sim_m^\Lambda t_2[r(0), \ldots, r(n-1)]$ for all $r\colon n \to m$;

(2) $t_1[\vec{u}] \sim_m^\Lambda t_2[\vec{u}]$ for all $m \in \mathbb{F}$ and $\vec{u} \in \Lambda(m)^n$;

(3) $t_1 \to t_1' \implies \exists t_2'.\, t_2 \to t_2' \wedge t_1' \sim_n^\Lambda t_2'$;

(4) $t_1 = \lambda x.t_1' \implies \exists t_2'.t_2 = \lambda x.t_2' \wedge \forall e \in \Lambda(n).t_1'[e/x] \sim_n^\Lambda t_2'[e/x]$;

(5) $\exists x \in V(n), s_1, \ldots, s_k \in \Lambda(n).\, t_1 = x\, s_1 \cdots s_k \implies \exists y \in V(n), s_1', \ldots, s_m' \in \Lambda(n).\, t_2 = y\, s_1' \cdots s_m'$;

(6) $t_2 \to t_2' \implies \exists t_1'.\, t_1 \to t_1' \wedge t_1' \sim_n^\Lambda t_2'$;

(7) $t_2 = \lambda x.t_2' \implies \exists t_1'.t_1 = \lambda x.t_1' \wedge \forall e \in \Lambda(n).t_1'[e/x] \sim_n^\Lambda t_2'[e/x]$;

(8) $\exists y \in V(n), s_1', \ldots, s_m' \in \Lambda(n).\, t_2 = y\, s_1' \cdots s_m' \implies \exists x \in V(n), s_1, \ldots, s_k \in \Lambda(n).\, t_1 = x\, s_1 \cdots s_k$.

Note that condition (1) is redundant, as it follows from (2) by putting $\vec{u} = \mathsf{var}_m \cdot r$.

Proof of $\sim^\Lambda \subseteq \sim^{\mathrm{ap}}$. Note first that $\sim_0^\Lambda \subseteq \Lambda(0) \times \Lambda(0)$ is a strong applicative bisimulation: the above conditions (3), (4), (6), (7) for $n = 0$ correspond precisely to (A1)–(A4) with $\sim_0^{\mathrm{ap}}$ replaced by $\sim_0^\Lambda$. It follows that $\sim_0^\Lambda \subseteq \sim_0^{\mathrm{ap}}$ because $\sim_0^{\mathrm{ap}}$ is the greatest strong applicative bisimulation. Moreover, for $n > 0$ and $t_1 \sim_n^\Lambda t_2$, we have

$$t_1[\vec{u}] \sim_0^\Lambda t_2[\vec{u}] \quad \text{for every } \vec{u} \in \Lambda(0)^n$$

by condition (2), whence

$$t_1[\vec{u}] \sim_0^{\mathrm{ap}} t_2[\vec{u}] \quad \text{for every } \vec{u} \in \Lambda(0)^n$$

because $\sim_0^\Lambda \subseteq \sim_0^{\mathrm{ap}}$, and so $t_1 \sim_n^{\mathrm{ap}} t_2$. This proves $\sim_n^\Lambda \subseteq \sim_n^{\mathrm{ap}}$ for $n > 0$ and thus $\sim^\Lambda \subseteq \sim^{\mathrm{ap}}$ overall.

Proof of $\sim^{\mathrm{ap}} \subseteq \sim^\Lambda$. Since $\sim^\Lambda$ is the greatest bisimulation, it suffices to show that $\sim^{\mathrm{ap}}$ is a bisimulation. Thus suppose that $n \in \mathbb{F}$ and $t_1 \sim_n^{\mathrm{ap}} t_2$; we need to verify the above conditions (2)–(8) with $\sim_n^\Lambda$ replaced by $\sim_n^{\mathrm{ap}}$. Let us first consider the case $n = 0$:

(2) Since t_1 and t_2 are closed terms, this condition simply states that $t_1 \sim_m^{\mathrm{ap}} t_2$ for every $m > 0$. This holds by definition of $\sim_m^{\mathrm{ap}}$ because $t_1[\vec{u}] = t_1 \sim_0^{\mathrm{ap}} t_2 = t_2[\vec{u}]$ for every $\vec{u} \in \Lambda(0)^m$.

(3) holds by (A1).

(4) holds by (A2).

(5) holds vacuously because t_1 is a closed term.

(6) holds by (A3).

(7) holds by (A4).

(8) holds vacuously because t_2 is a closed term.

Now suppose that $n > 0$:

(2) Let $\vec{u} = (u_0, \ldots, u_{n-1}) \in \Lambda(m)^n$. If $m = 0$ we have $t_1[\vec{u}] \sim_0^{\mathrm{ap}} t_2[\vec{u}]$ by definition of $\sim_n^{\mathrm{ap}}$. If $m > 0$ and $\vec{v} \in \Lambda(0)^m$ we have

$$t_1[\vec{u}][\vec{v}] = t_1[u_0[\vec{v}], \ldots, u_{n-1}[\vec{v}]] \sim_0^{\mathrm{ap}} t_2[u_0[\vec{v}], \ldots, u_{n-1}[\vec{v}]] = t_2[\vec{u}][\vec{v}],$$

whence $t_1[\vec{u}] \sim_m^{\mathrm{ap}} t_2[\vec{u}]$.

(3) Suppose that $t_1 \to t_1'$. We only need to prove that t_2 reduces, that is, $t_2 \to t_2'$ for some $t_2' \in \Lambda(n)$. Then, for every $\vec{u} \in \Lambda(0)^n$ we have $t_1[\vec{u}] \sim_0^{\mathrm{ap}} t_2[\vec{u}]$ by definition of $\sim_n^{\mathrm{ap}}$, and $t_1[\vec{u}] \to t_1'[\vec{u}]$ and $t_2[\vec{u}] \to t_2'[\vec{u}]$ because reductions respect substitution. Therefore $t_1'[\vec{u}] \sim_0^{\mathrm{ap}} t_2'[\vec{u}]$ by (A1), which proves $t_1' \sim_n^{\mathrm{ap}} t_2'$ by definition of $\sim_n^{\mathrm{ap}}$.

To prove that t_2 reduces, suppose towards a contradiction that t_2 does not reduce. There are two possible cases:

<u>Case 1:</u> t_2 is a λ-abstraction.
Since the term t_1 reduces, it is neither a variable nor a λ-abstraction. Therefore, for arbitrary $\vec{u} \in \Lambda(0)^n$, the term $t_1[\vec{u}]$ is not a λ-abstraction, whereas the term $t_2[\vec{u}]$ is a λ-abstraction. Thus $t_1[\vec{u}] \nsim_0^{\mathrm{ap}} t_2[\vec{u}]$ and therefore $t_1 \nsim_n^{\mathrm{ap}} t_2$, a contradiction.

<u>Case 2:</u> $t_2 = x\, s_1 \cdots s_k$ for some $x \in V(n)$ and $s_1, \ldots, s_k \in \Lambda(n)$, $k \geq 0$.
Given λ-terms s, t and $m \geq 0$, we write $s \to^m t$ if s reduces to t in exactly m steps; in particular, $s \to^0 t$ if $s = t$. We shall prove that there exists $\vec{u} \in \Lambda(0)^n$ such that

$$t_1[\vec{u}] \to^m \tilde{t}_1 \qquad \text{and} \qquad t_2[\vec{u}] \to^m \tilde{t}_2 \qquad \text{for some } m \geq 0 \text{ and } \tilde{t}_1, \tilde{t}_2 \in \Lambda(0),$$

where exactly one of the terms $\tilde{t_1}$ and $\tilde{t_2}$ is a λ-abstraction. Then $\tilde{t_1} \not\sim_0^{\mathrm{ap}} \tilde{t_2}$ by (A1) and (A2), whence $t_1[\vec{u}] \not\sim_0^{\mathrm{ap}} t_2[\vec{u}]$ by m-fold application of (A1), and so $t_1 \not\sim_n^{\mathrm{ap}} t_2$, a contradiction.

In order to construct $\vec{u} \in \Lambda(0)^n$ with the desired property, we consider several subcases:

Case 2.1: $t_1 \to^k \overline{t_1}$ for some $\overline{t_1}$.

Case 2.1.1: $\overline{t_1}$ is a λ-abstraction.
Choose $\vec{u}$ such that $u_x \to u_x$ (e.g. $u_x = (\lambda y.y\,y)\,(\lambda y.y\,y)$). Then $t_2[\vec{u}] \to^k t_2[\vec{u}]$ and $t_2[\vec{u}]$ is not a λ-abstraction, while $t_1[\vec{u}] \to^k \overline{t_1}[\vec{u}]$ and $\overline{t_1}[\vec{u}]$ is a λ-abstraction.

Case 2.1.2: $\overline{t_1}$ is an application $\overline{t_{1,1}}\,\overline{t_{1,2}}$.
Choose $\vec{u}$ such that $u_x = \lambda x_1.\lambda x_2.\ldots\lambda x_k.\lambda y.y$. Then $t_2[\vec{u}] \to^k \lambda y.y$, while $t_1[\vec{u}] \to^k \overline{t_1}[\vec{u}]$ and $\overline{t_1}[\vec{u}]$ is not a λ-abstraction.

Case 2.1.3: $\overline{t_1} = x$.
Choose $\vec{u}$ such that $u_x = \lambda x_1.\lambda x_2.\ldots\lambda x_k.t$ where t is an arbitrary closed term that is not a λ-abstraction. Note that u_x is a λ-abstraction: Since t_1 reduces, we have $t_1 \neq x = \overline{t_1}$ and thus necessarily $k > 0$. Thus $t_1[\vec{u}] \to^k \overline{t_1}[\vec{u}] = u_x$ and u_x is a λ-abstraction, while $t_2[\vec{u}] \to^k t$ and t is not a λ-abstraction.

Case 2.1.4: $\overline{t_1} = y$ for some variable $y \neq x$.
Choose $\vec{u}$ such that $u_x = \lambda x_1.\lambda x_2.\ldots\lambda x_k.\lambda y.y$ and u_y is not a λ-abstraction. Then $t_2[\vec{u}] \to^k \lambda y.y$, while $t_1[\vec{u}] \to^k \overline{t_1}[\vec{u}] = u_y$ and u_y is not a λ-abstraction.

Case 2.2: $t_1 \to^m \overline{t_1}$ for some $m < k$ such that $\overline{t_1}$ does not reduce. (Note that $m \neq 0$ because $t_1 \to t_1'$.)

Case 2.2.1: $\overline{t_1}$ is a λ-abstraction.
Choose $\vec{u}$ such that $u_x \to u_x$. Then $t_1[\vec{u}] \to^m \overline{t_1}[\vec{u}]$ and $\overline{t_1}[\vec{u}]$ is a λ-abstraction, while $t_2[\vec{u}] \xrightarrow{m} t_2[\vec{u}]$ and $t_2[\vec{u}]$ is not a λ-abstraction.

Case 2.2.2: $\overline{t_1} = y\,s_1' \ldots s_l'$ for some variable $y \neq x$ and terms $s_1', \ldots, s_l'$, $l \geq 0$.
Choose $\vec{u}$ such that $u_x \to u_x$ and $u_y = \lambda x_1.\lambda x_2.\ldots\lambda x_l.\lambda y.y$. Then $t_1[\vec{u}] \to^m \overline{t_1}[\vec{u}] \to^l \lambda y.y$ while $t_2[\vec{u}] \to^{m+1} t_2[\vec{u}]$ and $t_2[\vec{u}]$ is not a λ-abstraction.

Case 2.2.3: $\overline{t_1} = x\,s_1' \cdots s_l'$ for some $l > k - m$ and terms $s_1', \ldots, s_l'$.
Choose $\vec{u}$ such that $u_x = \lambda x_1.\lambda x_2.\cdots\lambda x_k.\lambda y.y$. Then $t_2[\vec{u}] \to^k \lambda y.y$, while

$$t_1[\vec{u}] \to^m \overline{t_1}[\vec{u}] \to^{k-m} (\lambda x_{k-m+1}.\cdots\lambda x_k.\lambda y.y)\, s_{k-m+1}'[\vec{u}] \cdots s_l'[\vec{u}]$$

and $(\lambda x_{k-m+1}.\cdots\lambda x_k.\lambda y.y)\, s_{k-m+1}'[\vec{u}] \cdots s_l'[\vec{u}]$ is not a λ-abstraction.

Case 2.2.4: $\overline{t_1} = x\,s_1' \cdots s_l'$ for some $l \leq k - m$ and terms $s_1', \ldots, s_l'$.
Choose $\vec{u}$ such that $u_x = \lambda x_1.\lambda x_2.\ldots\lambda x_k.t$ where t is an arbitrary closed term that is not a λ-abstraction. Then

$$t_2[\vec{u}] \to^{m+l} (\lambda x_{m+l+1}.\cdots\lambda x_k.t)\, s_{m+l+1}[\vec{u}] \cdots s_k[\vec{u}]$$

and $(\lambda x_{m+l+1}\cdots\lambda x_k.t)\, s_{m+l+1}[\vec{u}] \cdots s_k[\vec{u}]$ is not a λ-abstraction (for $l = k - m$, this is just the term t), while

$$t_1[\vec{u}] \to^m \overline{t_1}[\vec{u}] xto^l \lambda x_{l+1}.\cdots\lambda x_k.t$$

and $\lambda x_{l+1}.\cdots\lambda x_k.t$ is a λ-abstraction since $l < k$. (Recall that $m \neq 0$.)

(6) holds by symmetry to (3).

(4) Suppose that $t_1 = \lambda x.t_1'$. Then t_2 does not reduce (otherwise t_1 reduces by (6), a contradiction). Moreover, t_2 cannot be of the form $y\, s_1' \cdots s_l'$ where y is variable and $s_1', \dots, s_l'$ are terms. In fact, suppose the contrary, and choose $\vec{u} \in \Lambda(0)^n$ such that u_y is not a λ-abstraction. Then $t_1[\vec{u}] \nsim_0^{\mathrm{ap}} t_2[\vec{u}]$ since $t_1[\vec{u}]$ is a λ-abstraction and $t_2[\vec{u}]$ is not, contradicting $t_1 \sim^{\mathrm{ap}} t_2$.

Thus $t_2 = \lambda x.t_2'$ for $x = n$ and $t_2' \in \Lambda(n+1)$. Moreover, for every $e \in \Lambda(n)$ and $\vec{u} \in \Lambda(0)^n$ we have

$$\begin{aligned} t_1'[e/x][\vec{u}] &= t_1'[\vec{u}, e[\vec{u}]] = t_1'[\vec{u}, x][e[\vec{u}]/x] \\ \sim_0^{\mathrm{ap}} t_2'[\vec{u}, x][e[\vec{u}]/x] &= t_2'[\vec{u}, e[\vec{u}]] = t_2'[e/x][\vec{u}] \end{aligned}$$

using (A2) and that $t_1[\vec{u}] \sim_0^{\mathrm{ap}} t_2[\vec{u}]$ by definition of $\sim_n^{\mathrm{ap}}$. This proves $t_1'[e/x] \sim_n^{\mathrm{ap}} t_2'[e/x]$.

(7) holds by symmetry to (4).

(5) Suppose that $t_1 = x\, s_1 \cdots s_k$. Then t_2 does not reduce by (6) and is not a λ-abstraction by (7), so it must be of the form $t_2 = y\, s_1' \cdots s_m'$.

(8) holds by symmetry to (5). ◻

The above proposition and our general compositionality result (Theorem 4.14) imply:

Corollary 5.17. *The open extension $\sim^{\mathrm{ap}}$ of strong applicative bisimilarity is a congruence.*

Proof. We only need to verify that our present setting satisfies the conditions of Theorem 4.14:

(1) The presheaf category $\mathbf{Set}^{\mathbb{F}}$ is regular, being a topos.

(2) The functor $\Sigma X = V + \delta X + X \times X$ preserves reflexive coequalizers. In fact, δ is a left adjoint (with right adjoint $\langle\!\langle V+1, - \rangle\!\rangle$, see [17]) and thus preserves all colimits. Moreover, reflexive coequalizers commute with finite products and coproducts in $\mathbf{Set}$, hence also in $\mathbf{Set}^{\mathbb{F}}$ since limits and colimits are formed pointwise.

(3) The functor $B(X,Y) = \langle\!\langle X, Y \rangle\!\rangle \times (Y + Y^X + 1)$ preserves monos. To see this, note first that monos in $\mathbf{Set}^{\mathbb{F}}$ are the componentwise injective natural transformations and thus stable under products and coproducts; hence it suffices to show that the functors $(X,Y) \mapsto Y^X$ and $(X,Y) \mapsto \langle\!\langle X, Y \rangle\!\rangle$ preserve monos. The first functor preserves monos in the covariant component because $(-)^X \colon \mathbf{Set}^{\mathbb{F}} \to \mathbf{Set}^{\mathbb{F}}$ is a right adjoint, and in the contravariant component because $Y^{(-)} \colon (\mathbf{Set}^{\mathbb{F}})^{\mathsf{op}} \to \mathbf{Set}^{\mathbb{F}}$ is a right adjoint (with left adjoint $(Y^{(-)})^{\mathsf{op}} \colon \mathbf{Set}^{\mathbb{F}} \to (\mathbf{Set}^{\mathbb{F}})^{\mathsf{op}}$). The second functor preserves monos in the covariant component because $\langle\!\langle X, - \rangle\!\rangle \colon \mathbf{Set}^{\mathbb{F}} \to \mathbf{Set}^{\mathbb{F}}$ is a right adjoint. To see that it preserves monos in the contravariant component, suppose that $f \colon X' \to X$ is an epimorphism in $\mathbf{Set}^{\mathbb{F}}$. Then $\langle\!\langle f, Y \rangle\!\rangle \colon \langle\!\langle X, Y \rangle\!\rangle \to \langle\!\langle X', Y \rangle\!\rangle$ is the natural transformation with components

$$\langle\!\langle f, Y \rangle\!\rangle_n \colon \mathsf{Nat}(X^n, Y) \to \mathsf{Nat}((X')^n, Y), \qquad g \mapsto g \cdot f^n.$$

This map is clearly monic because f is epic. Thus $\langle\!\langle f, Y \rangle\!\rangle$ is monic in $\mathbf{Set}^{\mathbb{F}}$. ◻

5.5 Call-by-value evaluation

Much analogously to the call-by-name λ-calculus, we can implement the call-by-value λ-calculus (Figure 4) in higher-order abstract GSOS. The corresponding higher-order GSOS law differs from the one in Definition 5.9 only in the case of application on closed terms.

$$\mathsf{app1}\,\frac{q = \lambda x.\_}{(\lambda x.p)\, q \to p[q/x]} \qquad \mathsf{app2}\,\frac{p \to p'}{p\, q \to p'\, q} \qquad \mathsf{app3}\,\frac{q \to q'}{(\lambda x.p)\, q \to (\lambda x.p)\, q'}$$

Fig. 4. Small-step operational semantics of the call-by-value λ-calculus.

Definition 5.18 (V-pointed higher-order GSOS law of the call-by-value λ-calculus)**.**

$$\varrho^{\mathrm{cv}}_{X,Y} \colon \Sigma(jX \times B(jX, Y)) \to B(jX, \Sigma^{\star}(jX + Y))$$

is given by

$$\begin{array}{lll}
\varrho^{\mathrm{cv}}_{X,Y,n}(tr) = \texttt{case}\ tr\ \texttt{of} & & \\
v \in V(n) & \mapsto & \pi(v), * \\
\lambda.(t, f, _) & \mapsto & \langle\!\langle X, \lambda.(-) \cdot \eta \cdot \mathsf{inr} \rangle\!\rangle(\varrho_1(f)), (\eta \cdot \mathsf{inr})^X(\varrho_2(f)) \\
(t_1, g, t_1')\,(t_2, h, _) & \mapsto & \lambda m, \vec{u} \in X(m)^n.\,(g_m(\vec{u})\; h_m(\vec{u})), t_1'\, t_2 \\
(t_1, g, k)\,(t_2, h, t_2') & \mapsto & \lambda m, \vec{u} \in X(m)^n.\,(g_m(\vec{u})\; h_m(\vec{u})), t_1\, t_2' \\
(t_1, g, k)\,(t_2, h, k_2) & \mapsto & \lambda m, \vec{u} \in X(m)^n.\,(g_m(\vec{u})\; h_m(\vec{u})), \eta \cdot \mathsf{inr} \cdot k(t_2) \\
(t_1, g, k)\,(t_2, h, _) & \mapsto & \lambda m, \vec{u} \in X(m)^n.\,(g_m(\vec{u})\; h_m(\vec{u})), \eta \cdot \mathsf{inr} \cdot k(t_2) \\
(t_1, g, *)\,(t_2, h, _) & \mapsto & \lambda m, \vec{u} \in X(m)^n.\,(g_m(\vec{u})\; h_m(\vec{u})), *
\end{array}$$

where $t \in \delta X(n)$, $f \in \delta\langle\!\langle X, Y \rangle\!\rangle(n)$, $g, h \in \langle\!\langle X, Y \rangle\!\rangle(n)$, $k, k_2 \in Y^X(n)$, $t_1, t_2 \in X(n)$ and $t_1', t_2' \in Y(n)$. Again brackets around the pairs on the right are omitted, and the last five clauses refer to the application operator $p\,q = \mathsf{app}(p, q)$.

Applying Theorem 4.14 to the call-by-value λ-calculus shows that coalgebraic bisimilarity, as expressed in Proposition 5.6, is a congruence. Note however that unlike the case for the call-by-name λ-calculus, coalgebraic bisimilarity does not correspond to a strong version of call-by-value applicative bisimilarity (see e.g. [44]): The former relates terms if they exhibit the same behaviour when applied to arbitrary closed terms, while the latter considers only application to *values*. Capturing call-by-value applicative bisimilarity in the coalgebraic framework is left as an open problem; see also Section 6.

5.6 *Typed λ-calculi*

We have shown in the present section how to implement untyped λ-calculi in the higher-order abstract GSOS framework, using presheaf models. Higher-order languages with types can be treated in a very similar manner, as demonstrated in [25] for the case of simple type systems, and in [23] for recursive types. In a nutshell, one moves from the base category $\mathbf{Set}^{\mathbb{F}}$ to the category $(\mathbf{Set}^{\mathbb{F}/\mathsf{Ty}})^{\mathsf{Ty}}$, where Ty is the set of types (regarded as a discrete category) and $\mathbb{F}/\mathsf{Ty}$ is the comma category of *typed variable contexts*, i.e. pairs (n, Γ) of a finite cardinal n and a function $\Gamma \colon n \to \mathsf{Ty}$. Informally, a presheaf $X \in (\mathbf{Set}^{\mathbb{F}/\mathsf{Ty}})^{\mathsf{Ty}}$ associates to every $\tau \in \mathsf{Ty}$ and $\Gamma \in \mathbb{F}/\mathsf{Ty}$ a set $X_\tau(\Gamma)$ of terms of type τ in context Γ. Using this categorical setup, it is not difficult to devise higher-order GSOS laws for typed λ-calculi, which are essentially type-indexed versions of Definition 5.9 and Definition 5.18 above. We refer the reader to *op. cit.* for more details.

6 Conclusions, further developments, and future work

We have introduced the notion of *higher-order GSOS law*, effectively transferring the principles behind the bialgebraic framework by [56] to higher-order languages. We have demonstrated that, under mild assumptions, strong coalgebraic bisimilarity in systems specified by higher-order GSOS laws is a congruence, a result guaranteeing the compositionality of semantics within our abstract framework. In addition, we have implemented combinatory logics as well as the call-by-name λ-calculus as higher-order GSOS laws in suitable categories.

Our compositionality result for strong coalgebraic bisimilarity illustrates that the higher-order abstract GSOS framework can not only *model* the operational semantics of higher-order languages, but more importantly also provides the means to *reason* about such languages at a high level of abstraction. In recent work, we have further substantiated this point by lifting several key operational techniques to the categorical generality of our framework. In [58] we introduce a generalization of *Howe's method*, see [32, 33], and apply it to derive a general congruence result for (weak) applicative similarity. In [25] we study unary logical predicates along with induction up-to techniques to reason about them efficiently, with proofs of strong normalization as a key application. Finally, in [23] we develop a theory of step-indexed logical relations as a sound proof method for contextual equivalence. We note that all these operational techniques are usually introduced in an ad hoc manner and need to be carefully adapted for each individual language. The more principled approach based on higher-order abstract GSOS provides a clean conceptual separation between their non-trivial, language-dependent core and their generic, language-independent aspects.

Substantial progress has also been achieved towards supporting different paradigms of operational semantics. Recall that Section 5.5 hints that the naive treatment of call-by-value languages in the present work is not entirely satisfactory, as the ensuing coalgebraic notion of bisimilarity does not match standard applicative bisimilarity. In [26], we resolve this issue by moving from **Set**-valued presheaves to $\mathbf{Set}^2$-valued presheaves, with one sort for values and one for non-value terms. In recent work [22], a similar method is used to apply our framework to stateful languages. By doing so, we also reconcile first-order abstract GSOS with stateful languages, (a well-known problem, see e.g. [2] and [20]). Moreover, Urbat [57] extends the theory of higher-order abstract GSOS to reason about behavioural conformances in languages with quantitative features.

Last but not least, in [24] we provide a higher-order bialgebraic account of denotational semantics in the style of Turi and Plotkin, completing the bialgebraic picture of Section 4.3.

Let us conclude with outlining some directions for future research. A powerful technique for compositionality results for effectful languages is given by *environmental bisimulations*, see [51], which we aim to understand from the perspective of our categorical approach. Another goal of interest is to extend the notion of a *morphism of distributive laws*, see [59] and [34], to higher-order GSOS laws. As a potential application, this would enable modeling compilers of higher-order languages that preserve semantic properties across compilation. In first-order abstract GSOS, this idea has been previously explored by [55] and [1]. Finally, we aim to develop a fibrational theory of logical relations in higher-order abstract GSOS, with potential applications to parametricity [45] and dependent types.

Acknowledgement. Stelios Tsampas wishes to thank Andreas Nuyts and Christian Williams for the numerous and fruitful discussions.

Funding. The authors acknowledge the following support:

- Sergey Goncharov is supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 527481841.
- Stefan Milius and Lutz Schröder are supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 517924115.
- Henning Urbat is supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 470467389.
- Stelios Tsampas is supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 419850228 and 527481841.

References

[1] Abate, C., Busi, M. & Tsampas, S. (2021) Fully abstract and robust compilation: And how to reconcile the two, abstractly. 19th Asian Symposium on Programming Languages and Systems, APLAS'21. Springer. pp. 83–101.

[2] Abou-Saleh, F. & Pattinson, D. (2011) Towards effects in mathematical operational semantics. Mathematical Foundations of Programming Semantics, MFPS 2011. Elsevier. pp. 81–104.

[3] Abramsky, S. (1990) The lazy λ-calculus. In *Research topics in Functional Programming*. Addison Wesley. pp. 65–117.

[4] Abramsky, S. & Ong, C. L. (1993) Full abstraction in the lazy lambda calculus. *Inf. Comput.* **105**(2).

[5] Adámek, J. (1974) Free algebras and automata realizations in the language of categories. *Comment. Math. Univ. Carol.* **15**(4), 589–602.

[6] Adámek, J., Rosický, J. & Vitale, E. (2011) *Algebraic Theories*. Cambridge University Press.

[7] Barr, M. (1970) Coequalizers and free triples. *Math. Z.* **116**, 307–322.

[8] Bartels, F. (2004) *On generalised coinduction and probabilistic specification formats: Distributive laws in coalgebraic modelling*. Ph.D. thesis. Vrije Universiteit Amsterdam.

[9] Bernstein, K. L. (1998) A congruence theorem for structured operational semantics of higher-order languages. 13th Annual IEEE Symposium on Logic in Computer Science, LICS'98. IEEE Computer Society. pp. 153–164.

[10] Bloom, B., Istrail, S. & Meyer, A. R. (1995) Bisimulation can't be traced. *J. ACM*. **42**(1), 232–268.

[11] Borthelle, P., Hirschowitz, T. & Lafont, A. (2020) A cellular Howe theorem. 35th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS'20. ACM. pp. 273–286.

[12] Curry, H. B. (1930) Grundlagen der kombinatorischen Logik. *Am. J. Math.* **52**(3), 509–536.

[13] Dal Lago, U., Gavazzo, F. & Levy, P. B. (2017) Effectful applicative bisimilarity: Monads, relators, and Howe's method. 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2017). IEEE Computer Society. pp. 1–12.

[14] Di Gianantonio, P., Honsell, F. & Lenisa, M. (2008) Rpo, second-order contexts, and λ-calculus. Foundations of Software Science and Computational Structures. Berlin, Heidelberg. Springer Berlin Heidelberg. pp. 334–349.

[15] Dreyer, D., Ahmed, A. & Birkedal, L. (2011) Logical step-indexed logical relations. *Log. Methods Comput. Sci.* **7**(2).

[16] Fiore, M. P. (2008) Second-order and dependently-sorted abstract syntax. 23d Annual IEEE Symposium on Logic in Computer Science (LICS 2008). IEEE Computer Society. pp. 57–68.

[17] Fiore, M. P., Plotkin, G. D. & Turi, D. (1999) Abstract syntax and variable binding. 14th Annual IEEE Symposium on Logic in Computer Science (LICS 1999). IEEE Computer Society. pp. 193–202.

[18] Fiore, M. P. & Staton, S. (2006) A congruence rule format for name-passing process calculi from mathematical structural operational semantics. 21st Annual IEEE Symposium on Logic in Computer Science, LICS'06. IEEE Computer Society. pp. 49–58.

[19] Fiore, M. P. & Turi, D. (2001) Semantics of name and value passing. 16th Annual IEEE Symposium on Logic in Computer Science (LICS 2001). IEEE Computer Society. pp. 93–104.

[20] Goncharov, S., Milius, S., Schröder, L., Tsampas, S. & Urbat, H. (2022) Stateful structural operational semantics. 7th International Conference on Formal Structures for Computation and Deduction, FSCD'22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik. pp. 30:1–30:19.

[21] Goncharov, S., Milius, S., Schröder, L., Tsampas, S. & Urbat, H. (2023) Towards a higher-order mathematical operational semantics. 50th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL

2023). ACM.

[22] Goncharov, S., Milius, S., Schröder, L., Tsampas, S. & Urbat, H. (2025b) Bialgebraic reasoning on stateful languages.

[23] Goncharov, S., Milius, S., Tsampas, S. & Urbat, H. (2024b) Bialgebraic reasoning on higher-order program equivalence. 39th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2024). IEEE Computer Society Press. Preprint: https://arxiv.org/abs/2402.00625.

[24] Goncharov, S., Peressotti, M., Tsampas, S., Urbat, H. & Volpe, S. (2026) Towards a higher-order bialgebraic denotational semantics.

[25] Goncharov, S., Santamaria, A., Schröder, L., Tsampas, S. & Urbat, H. (2024a) Logical predicates in higher-order mathematical operational semantics. 27th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2024). Springer. pp. 47–69.

[26] Goncharov, S., Tsampas, S. & Urbat, H. (2025a) Abstract operational methods for call-by-push-value. *Proc. ACM Program. Lang.* **9**(POPL).

[27] Gordon, A. D. & Pitts, A. M., editors. (1999) *Higher Order Operational Techniques in Semantics*. Cambridge University Press. USA.

[28] Hermida, C., Reddy, U. S. & Robinson, E. P. (2014) Logical relations and parametricity - A Reynolds programme for category theory and programming languages. *Electron. Notes Theor. Comput. Sci.* **303**, 149–180.

[29] Hindley, J. R. & Seldin, J. P. (2008) *Lambda-Calculus and Combinators: An Introduction*. Cambridge University Press. second edition.

[30] Hirschowitz, T. (2019) Familial monads and structural operational semantics. 46th ACM-SIGACT Symposium on Principles of Programming Languages, POPL'19. ACM. pp. 21:1–21:28.

[31] Hirschowitz, T. & Lafont, A. (2022) A categorical framework for congruence of applicative bisimilarity in higher-order languages. *Log. Methods Comput. Sci.* **18**(3).

[32] Howe, D. J. (1989) Equality in lazy computation systems. 4th Annual Symposium on Logic in Computer Science (LICS 1989). IEEE Computer Society. pp. 198–203.

[33] Howe, D. J. (1996) Proving congruence of bisimulation in functional programming languages. *Inf. Comput.* **124**(2), 103–112.

[34] Klin, B. & Nachyla, B. (2015) Presenting morphisms of distributive laws. 6th Conference on Algebra and Coalgebra in Computer Science, CALCO'15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik. pp. 190–204.

[35] Klin, B. & Sassone, V. (2008) Structural operational semantics for stochastic process calculi. 11th International Conference Foundations of Software Science and Computational Structures, FOSSACS'08. Springer. pp. 428–442.

[36] Lassen, S. B. (2005) Eager normal form bisimulation. 20th IEEE Symposium on Logic in Computer Science, LICS'05. IEEE Computer Society. pp. 345–354.

[37] Mac Lane, S. (1978) *Categories for the Working Mathematician*. vol. 5 of *Graduate Texts in Mathematics*. Springer. second edition.

[38] Mac Lane, S. & Moerdijk, I. (1994) *Sheaves in Geometry and Logic: A First Introduction to Topos Theory*. Universitext. Springer.

[39] Miculan, M. & Peressotti, M. (2016) Structural operational semantics for non-deterministic processes with quantitative aspects. *Theor. Comput. Sci.* **655**, 135–154.

[40] Milner, R. (1989) *Communication and concurrency*. PHI Series in computer science. Prentice Hall.

[41] Milner, R., Parrow, J. & Walker, D. (1992) A calculus of mobile processes, I. *Inf. Comput.* **100**(1).

[42] Mousavi, M. R. & Reniers, M. A. (2007) On well-foundedness and expressiveness of promoted tyft: Being promoted makes a difference. *Electron. Notes Theor. Comput. Sci.* **175**(1), 45–56.

[43] O'Hearn, P. W. & Riecke, J. G. (1995) Kripke logical relations and PCF. *Inf. Comput.* **120**(1), 107–116.

[44] Pitts, A. (2011) Howe's method for higher-order languages. In *Advanced Topics in Bisimulation and Coinduction*, Sangiorgi, D. & Rutten, J. (eds). Cambridge Tracts in Theoretical Computer Science. Cambridge University Press. pp. 197–232.

[45] Reynolds, J. C. (1983) Types, abstraction and parametric polymorphism. Information Processing 83, Proceedings of the IFIP 9th World Computer Congress, Paris, France, September 19-23, 1983. North-Holland/IFIP. pp. 513–523.

[46] Rot, J., Bonsangue, M. M. & Rutten, J. J. M. M. (2013) Coalgebraic bisimulation-up-to. SOFSEM 2013:

Theory and Practice of Computer Science, 39th International Conference on Current Trends in Theory and Practice of Computer Science, Špindlerův Mlýn, Czech Republic, January 26-31, 2013. Proceedings. Springer. pp. 369–381.

[47] Rutten, J. J. M. M. (2000) Universal coalgebra: a theory of systems. *Theor. Comput. Sci.* **249**(1).

[48] Sangiorgi, D. (1994) The lazy lambda calculus in a concurrency scenario. *Inf. Comput.* **111**(1).

[49] Sangiorgi, D. (1996) Bisimulation for higher-order process calculi. *Inf. Comput.* **131**(2), 141–178.

[50] Sangiorgi, D. (1998) On the bisimulation proof method. *Mathematical Structures in Computer Science*. **8**(5), 447–479.

[51] Sangiorgi, D., Kobayashi, N. & Sumii, E. (2007) Environmental bisimulations for higher-order languages. 22nd Annual IEEE Symposium on Logic in Computer Science, LICS'07. IEEE Computer Society. pp. 293–302.

[52] Scott, D. (1970) Outline of a mathematical theory of computation. Technical Report PRG02. Oxford University Computing Laboratory.

[53] Statman, R. (1985) Logical relations and the typed lambda-calculus. *Inf. Control.* **65**(2/3), 85–97.

[54] Tait, W. W. (1967) Intensional interpretations of functionals of finite type I. *J. Symbolic Log.* **32**(2), 198–212.

[55] Tsampas, S., Nuyts, A., Devriese, D. & Piessens, F. (2020) A categorical approach to secure compilation. 15th IFIP WG 1.3 International Workshop on Coalgebraic Methods in Computer Science, CMCS'20. Springer. pp. 155–179.

[56] Turi, D. & Plotkin, G. D. (1997) Towards a mathematical operational semantics. 12th Annual IEEE Symposium on Logic in Computer Science (LICS 1997). pp. 280–291.

[57] Urbat, H. (2026) Higher-order behavioural conformances via fibrations. *Proc. ACM Program. Lang.* **10**(POPL).

[58] Urbat, H., Tsampas, S., Goncharov, S., Milius, S. & Schröder, L. (2023) Weak similarity in higher-order mathematical operational semantics. 38th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2023). IEEE Computer Society Press.

[59] Watanabe, H. (2002) Well-behaved translations between structural operational semantics. *Electr. Notes Theor. Comput. Sci.* **65**(1), 337–357.