Theoretical Computer Science 412 (2011) 5043–5069

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Bialgebras for structural operational semantics: An introduction

Bartek Klin *

University of Warsaw, Faculty of Mathematics, Informatics and Mechanics, Banacha 2, 02-097 Warsaw, Poland
University of Cambridge, Computer Laboratory, William Gates Bldg., 15 JJ Thomson Avenue, Cambridge CB3 0FD, UK

ARTICLE INFO

Keywords:
Structural operational semantics
Coalgebra
Bialgebra

ABSTRACT

Bialgebras and distributive laws are an abstract, categorical framework to study various flavors of structural operational semantics. This paper aims to introduce the reader to the basics of bialgebras for operational semantics, and to sketch the state of the art in this research area.

1. Introduction

Structural Operational Semantics (SOS) is one of the most popular frameworks for the formal description of programming languages and process calculi. It has become the formalism of choice for a clear and concise presentation of many ideas and formalisms (see [3] for examples), and it is a viable option for the description of fully grown programming languages [38].

In the simplest and most well-studied form of SOS [1], the semantics of processes is described by means of nondeterministic labeled transition systems (LTSs), induced from inference rules following their syntactic structure. For example, the rule

$$\frac{x \xrightarrow{a} x' \quad y \xrightarrow{\bar{a}} y'}{x||y \xrightarrow{\tau} x'||y'}$$

used in the definition of the well-known process calculus CCS [37] means that if a process x can make a transition labeled with an atomic action a, and if y can make a transition labeled with a corresponding action $\bar{a}$, then the composite process $x||y$ can combine the two transitions into one labeled with the label τ.

Already from the original paper on SOS ([40], reprinted as [41]) it was clear that simple LTSs are only one kind of dynamic systems worth considering, and that to model different computational paradigms one needs to study transition systems with state, environments, etc. Later, also probabilistic, stochastic, timed and other kinds of systems were defined in various flavors of SOS. Although each of these flavors is a little different, they all share a common underlying theme: *the interplay between the structure (syntax) and the dynamics (behavior) of systems*.

Although the latter expression might sound a little vague, in the late 1990s it has found an elegant and general formalization with the use of basic category theory. The main conceptual step was made with the development of universal coalgebra, a general categorical approach that described several different kinds of transition systems in a uniform way. Since syntax has traditionally been modeled in the dual framework of universal algebra, with the benefit of hindsight it seems natural that the two theories should somehow be combined to explain the various flavors of SOS. This indeed happened in the seminal paper [54] where, building upon earlier initial ideas of [44], SOS specifications were formalized as *distributive laws* of syntax over behavior, both modeled as endofunctors on the same category, and models of specifications were defined as *bialgebras* for these laws.

* Corresponding author at: University of Warsaw, Faculty of Mathematics, Informatics and Mechanics, Banacha 2, 02-097 Warsaw, Poland. Tel.: +48 225544484.
E-mail addresses: klin@mimuw.edu.pl, bklin@inf.ed.ac.uk.

0304-3975/$ – see front matter © 2011 Published by Elsevier B.V.
doi:10.1016/j.tcs.2011.03.023

Bloom, Istrail, Meyer

More specifically, it was shown in [54] that SOS specifications of LTSs that are in the so-called GSOS format [4], correspond to a certain type of distributive laws of functors that model syntax over a functor that models the behavior of LTSs. The main property of GSOS specifications, that bisimilarity on LTSs induced from them is always a congruence, was formulated and proved at the abstract level of distributive laws. Since the bialgebraic framework is parametrized by the notions of syntax and behavior, this opened a possibility to understand well-behaved SOS formalisms for other kinds of systems in a uniform manner. This has indeed happened since, and several novel, concrete specification formats such as probabilistic or stochastic GSOS have been derived by analysis of the corresponding abstract distributive laws. Although much remains to be done, the bialgebraic framework has a good claim to be the main abstract approach to SOS. Furthermore, bialgebras have been used for an abstract understanding of ideas seemingly unconnected to SOS, such as stream equations or regular languages.

The purpose of this paper is to provide a gentle introduction to the basic framework of distributive laws for SOS, and a survey of the current state of the art in the area. We shall define a few types of distributive laws, from simple distributive laws of endofunctors over endofunctors, to more complex GSOS and coGSOS laws, to the general case of distributing monads over comonads. For concrete kinds of transition systems, most of these types correspond to progressively more permissive formats of well-behaved SOS specifications; to simplify the presentation, we shall concentrate on the very simple stream systems, a kind of automata with deterministic output and no input at all.

After reading this expository paper, the reader should be prepared (and, hopefully, motivated) to study the field of bialgebra in more depth. For further reading, the author recommends to begin with [44,54] and perhaps [52] to see how the ideas originally developed, Chapter 3 of [2] for a thorough but gentle exposition, and [34] for a more abstract categorical perspective.

The structure of the paper is as follows: after Section 2 of preliminaries about algebras and coalgebras, further development is motivated in Section 3 by a few concrete examples related to stream systems. Distributive laws are initially motivated not by a study of inference rules but by well-behaved definitions of operations on infinite streams, but stream SOS rules soon naturally appear as an additional benefit. In Section 4, the examples of the preceding section are cast in a general setting of simple distributive laws of endofunctors over endofunctors, and a basic theory of such laws is developed, up to an abstract formulation of the congruence property of observational equivalence. Section 5 focuses on the world of stream systems again, and provides a concrete, rule-based presentation of simple distributive laws for such systems.

Since simple laws of Section 4 are not expressive enough to cover all examples of interest, in Section 6 more complex types of laws are introduced, motivated by further examples of operations on stream systems. Section 7 presents concrete rule formats obtained so far by analysis of distributive laws for various kinds of systems, and Section 8 lists other relevant work related to bialgebras and their applications to SOS.

2. Algebras and coalgebras

The reader is assumed to be familiar with basic notions of category theory such as categories, functors and natural transformations. A standard reference for these is [35].

2.1. Syntax via algebras

functional

An *algebraic signature* Σ is a collection of operation symbols $\{\mathtt{f}_i \mid i \in I\}$ where each $\mathtt{f}_i$ has an arity $n_i \in \mathbb{N}$. A *Σ-algebra* with carrier set X is a map $\coprod_{i\in I} X^{n_i} \to X$, and therefore a signature Σ shall be identified with the functor $\Sigma X = \coprod_{i\in I} X^{n_i}$ on the category **Set** of sets and functions. In general, given any endofunctor Σ on a category $\mathcal{C}$:

Definition 1. A Σ-algebra is an object X in $\mathcal{C}$ together with a map $g : \Sigma X \to X$. A Σ-algebra morphism from $g : \Sigma X \to X$ to $e : \Sigma Y \to Y$ is a map $f : X \to Y$ such that $e \circ \Sigma f = f \circ g$. Σ-algebras and their morphisms form a category Σ**-alg**.

There is an obvious forgetful functor $U^\Sigma : \Sigma$**-alg** $\to \mathcal{C}$.

If Σ has an initial algebra $a : \Sigma A \to A$, then for any algebra g, the unique algebra map f from a to g is called the *inductive extension* of g. The principle of defining maps from the carrier of an initial Σ-algebra by providing another Σ-algebra is called *induction*.

For Σ on **Set** arising from a signature, the set Σ^*0 of closed Σ-terms carries an initial Σ-algebra structure, and the inductive extension of an algebra $g : \Sigma X \to X$ is the unique interpretation of closed Σ-terms in g defined by structural induction.

More generally, the set of Σ-terms over a set of variables X is denoted by Σ^*X. It is easy to see that Σ^* extends to a functor on **Set**.

2.2. Behavior via coalgebras

A detailed description of the coalgebraic approach to system dynamics is beyond the scope of this paper; the interested reader can consult [45] for a thorough introduction. This section only briefly recalls basic notions and results that will be useful in the following.

Fix any endofunctor B on a category $\mathcal{C}$, called a *behavior functor* in this context.

Definition 2. A *B*-coalgebra is an object X in $\mathcal{C}$ together with a map $h : X \to BX$. A B-coalgebra morphism from $h : X \to BX$ to $k : Y \to BY$ is a map $f : X \to Y$ such that $Bf \circ h = k \circ f$. B-coalgebras and their morphisms form a category B-**coalg**.

There is an obvious forgetful functor $U_B : B\text{-}\mathbf{coalg} \to \mathcal{C}$.

If B has a final coalgebra $z : Z \to BZ$, then for any coalgebra h, the unique map f from h to z:

$$\begin{array}{ccc} X & \xrightarrow{h} & BX \\ {\scriptstyle f}\vdots & & \downarrow{\scriptstyle Bf} \\ Z & \xrightarrow[z]{} & BZ \end{array} \tag{1}$$

will be called the *coinductive extension* of h. This principle of defining maps into the carrier of a final B-coalgebra by providing another B-coalgebra is called *coinduction*.

For $\mathcal{C} = \mathbf{Set}$, given any coalgebra $h : X \to BX$, we say that $x, y \in X$ are *observationally equivalent* if they are equated by some coalgebra morphism from h. If B admits a final coalgebra, this is equivalent to saying that x and y are equated by the coinductive extension of h. In this case, observational equivalence on h is the kernel relation of the coinductive extension of h.

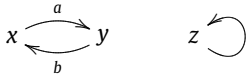
In much of the coalgebraic literature (see e.g. [45]), the notion of coalgebraic bisimilarity, based on spans of coalgebra morphisms, is used instead of observational equivalence. These two notions are equivalent if the behavior functor B preserves weak pullbacks; see [51] for a comparison of these and other coalgebraic notions of equivalence.

Example 3. A *stream system* for an alphabet L is a set X of states together with a transition function $h : X \to L \times X$, i.e., it is a coalgebra for the endofunctor $L \times -$ on **Set**. When h is understood from context, we shall use the graphical transition notation $x \xrightarrow{a} x'$ for $h(x) = (a, x')$.

The set L^ω of infinite streams of elements of L, together with the isomorphism $z = \langle \mathtt{hd}, \mathtt{tl} \rangle : L^\omega \to L \times L^\omega$, where:

$$\mathtt{hd}(a_1a_2a_3a_4\cdots) = a_1 \qquad \mathtt{tl}(a_1a_2a_3a_4\cdots) = a_2a_3a_4\cdots,$$

forms a final $(L \times -)$-coalgebra. The coinductive extension of a stream system maps each state to the infinite stream it produces over time; for example, if $L = \{a, b, c\}$, $X = \{x, y, z\}$ and h is described by the transition graph:



then the coinductive extension of h maps x to $ababab\cdots$, y to $bababa\cdots$ and z to $cccccc\cdots$. As a result, two states in a stream system are observationally equivalent if they produce the same stream.

See [46] for a detailed coalgebraic analysis of stream systems.

Example 4. A *Mealy machine* for an input alphabet K and output alphabet L is a set X of states together with a transition function $t : X \times K \to X \times L$. Similarly as for stream systems, we shall write $x \xrightarrow{a|b} y$ for $t(x, a) = (y, b)$ when t is understood from context. Mealy machines correspond to coalgebras for the endofunctor $(L \times -)^K$ on **Set**.

A function $f : K^\omega \to L^\omega$ is *causal* if for all $n \in \mathbb{N}$, the nth element of $f(\alpha)$ depends only on the first n elements of $\alpha \in K^\omega$. The set $\Gamma = \{f : A^\omega \to B^\omega \mid f \text{ is causal }\}$ carries a final Mealy coalgebra as shown in [47].

Two states in a Mealy machine are behaviorally equivalent if they have the same input–output behavior; see [47] for more details and a thorough presentation of the coalgebraic perspective on Mealy machines.

Example 5. A *labeled transition system* (LTS) is a triple $(X, L, \longrightarrow)$ where X is a set of states, L a set (alphabet) of labels, and $\longrightarrow \subseteq X \times L \times X$ a transition relation. One usually writes $x \xrightarrow{a} y$ for $(x, a, y) \in \longrightarrow$, and $x \not\xrightarrow{a}$ if there are no $y \in X$ such that $x \xrightarrow{a} y$. An LTS is *image-finite* if for each $x \in X$ and $a \in L$ there are only finitely many $y \in X$ such that $x \xrightarrow{a} y$.

LTSs labeled by L can be seen as coalgebra for the functor $(\mathcal{P}-)^L$, and image-finite LTSs for the functor $(\mathcal{P}_\omega-)^L$, where $\mathcal{P}$ is the (covariant) powerset, and $\mathcal{P}_\omega$ the finite powerset functor on **Set**. $\mathcal{P}$ does not admit final coalgebras for cardinality reasons, but a final $\mathcal{P}_\omega$-coalgebra exists. Two states in an LTS are observationally equivalent if and only if they are *bisimilar*; see e.g. [45] for more details on the coalgebraic understanding of LTSs.

Example 6. A finitely supported probability distribution on a set X is a function $\nu : X \to [0, 1]$ such that $\nu(x) = 0$ for all but finitely many x, and $\sum_{x \in X} \nu(x) = 1$. A (reactive) *probabilistic transition system* (PTS) is a triple (X, L, μ), where X is a set of states, L a set of labels, and the transition function $\mu : X \times L \times X \to [0, 1]$ is such that $\mu(x, a, -)$ is either the constantly zero function or a finitely supported probability distribution on X, for every $x \in X$ and $a \in L$.

PTSs are in one-to-one correspondence with coalgebras for the endofunctor $(1 + \mathcal{D}_\omega-)^L$, where $\mathcal{D}_\omega X$ is the set of all finitely supported probability distributions on X. This functor admits a final coalgebra. Two states in a PTS are observationally equivalent if and only if they are related by a *probabilistic bisimulation*; see [9] for a detailed coalgebraic treatment of PTSs.

Several other kinds of transition systems can be modeled as coalgebras for various behavior functors; see e.g. [45] for further examples.

3. Simple stream definitions and distributive laws: examples

In this section we shall see, on a few basic examples of operations on infinite streams, how a simple form of distributive laws appears in certain coinductive definitions. The additional structure present in these laws provides certain benefits, such as the presentation of definitions in terms of rules, or lifting stream systems to terms built of operations. The purpose of this section is to illustrate these benefits on simple examples and thus motivate the general development of Section 4. Later, in Section 5, the case of streams and their definitions will be revisited in more generality.

3.1. Some coinductive definitions

Coinduction, i.e., the use of coalgebra finality as in (1), is often used to define operations on carriers of final coalgebras. We shall now illustrate this on a few very simple examples of coinductively defined operations on infinite streams.

Example 7. Consider a simple alternating composition operation:

$$\mathtt{alt} : (L^\omega)^2 \to L^\omega$$

acting on infinite streams as follows:

$$\mathtt{alt}(a_1a_2a_3a_4\cdots, b_1b_2b_3b_4\cdots) = a_1b_2a_3b_4\cdots \tag{2}$$

To define alt formally by coinduction, one uses the finality of the coalgebra $z = \langle \mathtt{hd}, \mathtt{tl}\rangle : L^\omega \to L \times L^\omega$. To this end, pick an $(L \times -)$-coalgebra structure on the set $(L^\omega)^2$:

$$h^{\mathtt{alt}} : (L^\omega)^2 \to L \times (L^\omega)^2 \qquad h^{\mathtt{alt}}(\alpha, \beta) = \big(\mathtt{hd}(\alpha)\,,\ (\mathtt{tl}(\beta), \mathtt{tl}(\alpha))\big), \tag{3}$$

and define alt as the unique coalgebra morphism from $h^{\mathtt{alt}}$ to z:

$$\begin{array}{ccc} (L^\omega)^2 & \xrightarrow{h^{\mathtt{alt}}} & L \times (L^\omega)^2 \\ {\scriptstyle \mathtt{alt}}\big\downarrow & & \big\downarrow{\scriptstyle \mathrm{id}_L \times \mathtt{alt}} \\ L^\omega & \xrightarrow[z]{\cong} & L \times L^\omega. \end{array}$$

Example 8. Assume that the set L comes equipped with a binary operator $+$, which we shall call addition. Pointwise addition of infinite L-streams, $\oplus : (L^\omega)^2 \to L^\omega$, is then defined from an $(L \times -)$-coalgebra:

$$h^{\oplus} : (L^\omega)^2 \to L \times (L^\omega)^2 \qquad h^{\oplus}(\alpha, \beta) = \big(\mathtt{hd}(\alpha) + \mathtt{hd}(\beta)\,,\ (\mathtt{tl}(\alpha), \mathtt{tl}(\beta))\big),$$

again as the unique coalgebra morphism:

$$\begin{array}{ccc} (L^\omega)^2 & \xrightarrow{h^{\oplus}} & L \times (L^\omega)^2 \\ {\scriptstyle \oplus}\big\downarrow & & \big\downarrow{\scriptstyle \mathrm{id}_L \times \oplus} \\ L^\omega & \xrightarrow[z]{\cong} & L \times L^\omega. \end{array}$$

Example 9. Coinduction can also be used to define specific streams, seen as constant (i.e. nullary) operations. For example, for any $a \in L$, the constantly a stream:

$$\mathtt{a} = aaaaaaa\cdots \tag{4}$$

arises as the unique coalgebra morphism from $h^{\mathtt{a}}$:

$$h^{\mathtt{a}} : 1 \to L \times 1 \qquad h^{\mathtt{a}}(*) = (a, *)$$

to z:

$$\begin{array}{ccc} 1 & \xrightarrow{h^{\mathtt{a}}} & L \times 1 \\ {\scriptstyle \mathtt{a}}\big\downarrow & & \big\downarrow{\scriptstyle \mathrm{id}_L \times \mathtt{a}} \\ L^\omega & \xrightarrow[z]{\cong} & L \times L^\omega. \end{array}$$

Example 10. Examples as above can be combined in single definitions that provide whole families of operations. For instance, Examples 7 and 9 can be combined in a single coalgebra $h^{\mathtt{alt+L}} : \Sigma L^\omega \to L \times \Sigma L^\omega$, where $\Sigma X = X^2 + L$, that defines the operation alt and a family of constants $\{\mathtt{a} \mid a \in L\}$ at the same time. The function $h^{\mathtt{alt+L}}$ is obtained from $h^{\mathtt{alt}}$ and the $h^{\mathtt{a}}$ by cases (note that $L = \coprod_{a\in L} 1$):

$$h^{\mathtt{alt+L}}(\iota_{\mathtt{alt}}(\alpha,\beta)) = \big(\mathtt{hd}(\alpha), \iota_{\mathtt{alt}}(\mathtt{tl}(\beta), \mathtt{tl}(\alpha))\big)$$
$$h^{\mathtt{alt+L}}(\iota_{\mathtt{a}}(*)) = (a, \iota_{\mathtt{a}}(*)) \qquad \text{(for all } a \in A\text{)},$$

where $\iota_{\mathtt{alt}} : (L^\omega)^2 \to \Sigma(L^\omega)$ and the $\iota_{\mathtt{a}} : 1 \to \Sigma(L^\omega)$ are coproduct injections. It is easy to see that the Σ-algebra on L^ω obtained from $h^{\mathtt{alt+L}}$ by finality, can be in turn defined from the algebras `alt` and the `a` by cases.

This is a particularly simple example, since the operations under definition do not depend on one another. The same technique can also be used to provide mutually dependent definitions of operations. For example, one can define two constant streams

$$\mathtt{g} = \mathit{abababab}\cdots \qquad \mathtt{h} = \mathit{babababa}\cdots$$

by finality, with a coalgebra:

$$h^{\mathtt{gh}} : 2 \to L \times 2 \qquad \begin{array}{rcl} h^{\mathtt{gh}}(\iota_{\mathtt{g}}(*)) & = & (a, \iota_{\mathtt{h}}(*)) \\ h^{\mathtt{gh}}(\iota_{\mathtt{h}}(*)) & = & (b, \iota_{\mathtt{g}}(*)) \end{array}$$

where $\iota_{\mathtt{h}}, \iota_{\mathtt{g}} : 1 \to 2$ are the coproduct injections.

3.2. Distributive laws

All coalgebras used in the above examples are of a special, well-structured kind. The values of $h^{\mathtt{alt}}$ on $h^{\oplus}$ on given streams α and β are obtained by first applying the functions `hd` and `tl` to α and β, and then by suitable arrangement of the results of this application, without any other use of α and β themselves. Formally, both $h^{\mathtt{alt}}$ and $h^{\oplus}$ factor through the pointwise application of the final coalgebra $z : L^\omega \to L \times L^\omega$ as follows:

$$h^{\mathtt{alt}} = \lambda^{\mathtt{alt}}_{L^\omega} \circ (z \times z) \qquad h^{\oplus} = \lambda^{\oplus}_{L^\omega} \circ (z \times z) \tag{5}$$

where $\lambda^{\mathtt{alt}}_{L^\omega}$, $\lambda^{\oplus}_{L^\omega} : (L \times L^\omega)^2 \to L \times (L^\omega)^2$ are defined by:

$$\lambda^{\mathtt{alt}}_{L^\omega}(a,\alpha,b,\beta) = (a,\beta,\alpha)$$
$$\lambda^{\oplus}_{L^\omega}(a,\alpha,b,\beta) = (a+b,\alpha,\beta).$$

These maps $\lambda^{\mathtt{alt}}_{L^\omega}$ and $\lambda^{\oplus}_{L^\omega}$ are *natural* in arguments α and β, i.e., they merely rearrange them without any further dependence on their values. Formally, this means that they are components of natural transformations:

$$\lambda^{\mathtt{alt}}, \lambda^{\oplus} : (L \times -)^2 \Longrightarrow L \times (-)^2$$

with components on any set X defined by:

$$\lambda^{\mathtt{alt}}_X(a,x,b,y) = (a,y,x)$$
$$\lambda^{\oplus}_X(a,x,b,y) = (a+b,x,y).$$

It is easy to check that these do form natural transformations. Transformations of this shape are called distributive laws of the functor $(-)^2$ over the functor $L \times -$.

Somewhat trivially, the definition of $h^{\mathtt{a}}$ in Example 9 can be understood in a similar manner. Here, the relevant distributive law

$$\lambda^{\mathtt{a}} : K_1 = (L \times -)^0 \Longrightarrow L \times (-)^0 = L \times K_1$$

(where K_1 is the functor constant at the one-element set 1) has all components equal to $h^{\mathtt{a}}$, and the factorization through the "pointwise" application of z is trivial:

$$h^{\mathtt{a}} = \lambda^{\mathtt{a}}_{L^\omega} \circ z^0 = \lambda^{\mathtt{a}}_{L^\omega}$$

(note that z^0, the 0-th power of z, is the identity function on 1).

In Example 10, the combination of coalgebras to define multiple operations at the same time arises from a similar combination of the corresponding distributive laws. The joint definition of `alt` and the constants `a` arises from a distributive law defined by cases, as $\lambda^{\mathtt{alt}}$ or $\lambda^{\mathtt{a}}$:

$$\lambda^{\mathtt{alt+L}} : \Sigma(L \times -) \Longrightarrow L \times \Sigma - \qquad \begin{array}{rcl} \lambda^{\mathtt{alt+L}}_X(\iota_{\mathtt{alt}}(a,x,b,y)) & = & (a, \iota_{\mathtt{alt}}(y,x)) \\ \lambda^{\mathtt{alt+L}}_X(\iota_{\mathtt{a}}(*)) & = & (a, \iota_{\mathtt{a}}(*)). \end{array}$$

It is easy to check that, indeed, as in (5),

$$h^{\mathtt{alt+L}} = \lambda^{\mathtt{alt+L}}_{L^\omega} \circ \Sigma z.$$

3.3. Rules for stream definitions

Distributive laws that define simple operations on streams can often be conveniently presented using inference rules. Consider for example the alternating composition operation $\mathtt{alt} : (L^\omega)^2 \to L^\omega$ defined in Example 7. The definition (3) of the map $h^{\mathtt{alt}} : (L^\omega)^2 \to L \times (L^\omega)^2$, can be rephrased with the following family of statements, one for each $a, b \in L$:

For each $\alpha, \beta \in L^\omega$, *if*

$$\mathtt{hd}(\alpha) = a, \quad \mathtt{hd}(\beta) = b, \quad \mathtt{tl}(\alpha) = \alpha', \quad \mathtt{tl}(\beta) = \beta',$$

then

$$\mathtt{hd}(h^{\mathtt{alt}}(\alpha, \beta)) = a, \quad \mathtt{tl}(h^{\mathtt{alt}}(\alpha, \beta)) = (\beta', \alpha').$$

Using an arrow notation $\alpha \xrightarrow{a} \alpha'$ to say that $\mathtt{hd}(\alpha) = a$ and $\mathtt{tl}(\alpha) = \alpha'$, and an inference rule notation for implication, this can be rewritten as a family of rules:

$$\frac{\mathtt{x} \xrightarrow{a} \mathtt{x}' \quad \mathtt{y} \xrightarrow{b} \mathtt{y}'}{(\mathtt{x}, \mathtt{y}) \xrightarrow{a} (\mathtt{y}', \mathtt{x}')}$$

for each $a, b \in L$, where $\mathtt{x}, \mathtt{x}', \mathtt{y}, \mathtt{y}'$ are metavariables that denote streams.

When a whole collection of operations is defined at the same time, it is convenient to mention the operation under definition in the rules. Here, tupling of streams on both sides of the conclusion of the rule pertains to the operation $\mathtt{alt}$ under definition, and the rule might be written down as:

$$\frac{\mathtt{x} \xrightarrow{a} \mathtt{x}' \quad \mathtt{y} \xrightarrow{b} \mathtt{y}'}{\mathtt{alt}(\mathtt{x}, \mathtt{y}) \xrightarrow{a} \mathtt{alt}(\mathtt{y}', \mathtt{x}')}.$$

Notice a slight subtlety here: formally, we use a family of rules, one for each pair of labels $a, b \in L$, rather than a single rule where a, b are metavariables that represent labels. The reason for this choice might not be evident in the above example, but consider the definition of the pointwise stream addition operation $\oplus$ of Example 8, which can similarly be presented as:

$$\frac{\mathtt{x} \xrightarrow{a} \mathtt{x}' \quad \mathtt{y} \xrightarrow{b} \mathtt{y}'}{\mathtt{x} \oplus \mathtt{y} \xrightarrow{a+b} \mathtt{x}' \oplus \mathtt{y}'}.$$

Were this interpreted as a single rule with label metavariables a and b, we would need to give a formal meaning to the expression $a + b$ in the conclusion label. It is technically simpler to fix the value of a and b as labels in each rule, and interpret $a + b$ at the meta-level, as another element of L. Somewhat more elaborately, one might represent the above as a family of rules:

$$\frac{\mathtt{x} \xrightarrow{a} \mathtt{x}' \quad \mathtt{y} \xrightarrow{b} \mathtt{y}'}{\mathtt{x} \oplus \mathtt{y} \xrightarrow{c} \mathtt{x}' \oplus \mathtt{y}'}$$

where a, b, c range over L so that $c = a + b$. In the following, we shall avoid such pedantry and consistently adopt the convention that all labels in each rule are fixed elements of L.

A constant $\mathtt{a}$ of Example 9 can be defined by a single rule with no premises:

$$\frac{}{\mathtt{a} \xrightarrow{a} \mathtt{a}}.$$

Note that this is *not* a family of rules indexed by $a \in A$. This shows that one often needs to explicitly say whether one represents a single rule, or a rule schema parametrized by some labels. We shall, however, sometimes neglect to do this when no risk of confusion arises, in particular when repeating rules for operations that have been defined previously.

With operations under definition mentioned in the rules, it is quite easy to write down definitions for families of operations: usually it is enough to put the relevant rules together. For instance, $h^{\mathtt{alt}+\mathtt{L}}$ of Example 10 can be defined by rules:

$$\frac{\mathtt{x} \xrightarrow{a} \mathtt{x}' \quad \mathtt{y} \xrightarrow{b} \mathtt{y}'}{\mathtt{alt}(\mathtt{x}, \mathtt{y}) \xrightarrow{a} \mathtt{alt}(\mathtt{y}', \mathtt{x}')} (\forall a, b \in L) \qquad \frac{}{\mathtt{a} \xrightarrow{a} \mathtt{a}} \qquad \frac{}{\mathtt{b} \xrightarrow{b} \mathtt{b}} \qquad \cdots \tag{6}$$

We have seen just a few examples of rule-based presentations of coinductive definitions. A more general and formal treatment is postponed until Section 5, where we shall see that certain syntactic formats of inference rules correspond to various kinds of distributive laws.

3.4. Behavior of closed terms

A rule-based presentation immediately suggests a notion of inference, i.e., a tree where every node is an instance of a rule. For example,

$$\dfrac{\dfrac{}{\mathtt{a} \xrightarrow{a} \mathtt{a}} \qquad \dfrac{\mathtt{b} \xrightarrow{b} \mathtt{b} \quad \mathtt{c} \xrightarrow{c} \mathtt{c}}{\mathtt{alt(b,c)} \xrightarrow{b} \mathtt{alt(c,b)}}}{\mathtt{alt(a,\,alt(b,\,c))} \xrightarrow{a} \mathtt{alt(alt(c,\,b),\,a)}}$$

is an inference based on the set of rules given in (6). A general notion of inference will be formally defined in Section 5; for now, notice how rules induce a stream system on the set of closed terms built of operation symbols `alt` and `a` (for $a \in A$). Indeed, one may define $t \xrightarrow{a} t'$ if and only if it is derivable by rules from (6). The resulting stream system has e.g. the following transitions:

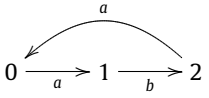
$$\mathtt{a} \circlearrowright a \qquad \mathtt{alt(a,\,b)} \underset{b}{\overset{a}{\rightleftarrows}} \mathtt{alt(b,\,a)} \qquad \mathtt{alt(a,\,alt(b,\,c))} \underset{c}{\overset{a}{\rightleftarrows}} \mathtt{alt(alt(c,\,b),\,a)} \tag{7}$$

Note how we have silently changed the meaning of operation names: `alt` is now a term-building construct, i.e., a binary operation symbol in some algebraic signature, rather than an operation on infinite streams. Similarly, `a` is now a constant symbol rather than an infinite stream. Formally, Eqs. (2) and (4) in Examples 7 and 9 should now be understood as interpretations of operation symbols in a particular algebra for that signature.

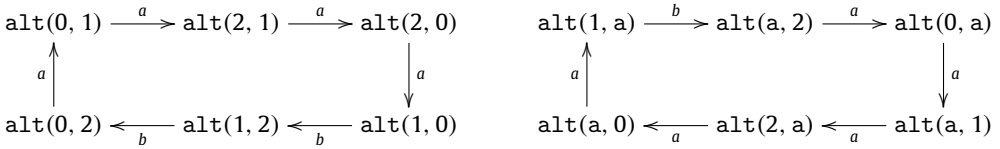
3.5. Lifting behavior to terms

The classical notion of inference easily extends to terms built over a nonempty set of variables, provided that the variables come equipped with some stream transitions. Formally, given a set of rules that define operations from a signature Σ, one might extend a stream system h with a carrier X to a stream system with carrier Σ^*X, the set of Σ-terms over X. In the extended system, for terms $t, t' \in \Sigma^*X$, there is a transition $t \xrightarrow{a} t'$ if and only if it is derivable from the set of premises $\{x \xrightarrow{b} x' \mid x \xrightarrow{b} x' \text{ in } h\}$.

For example consider rules (6) for $L = \{a, b\}$. The stream system on $X = \{0, 1, 2\}$:



induces a system with transitions, among others:



The example of Section 3.4 arises as a special case, where X is empty with the unique stream system structure.

In some sense, coinductive definitions presented in terms of inference rules provide a way to interpret the operations under definition not only on infinite streams, but on an arbitrary stream system. Although terms built over states of a system h are not, in general, interpreted as states of h again, their behavior is well defined by derivations, in an extended stream system.

3.6. Initial vs. final semantics

The simple development of the examples we have seen so far provides two ways of interpreting closed terms as infinite streams.

First, as shown in Sections 3.1 and 3.2, a distributive law such as $\lambda^{\mathtt{alt+L}}$ gives rise to a coalgebra $h^{\mathtt{alt+L}}$ on the set ΣL^ω, for $\Sigma X = X^2 + L$, the endofunctor corresponding to the relevant signature of operations. Further, this gives rise to a Σ-algebra on the set L^ω of streams. Now, since the set Σ^*0 of closed Σ-terms carries an initial Σ-algebra, this gives rise to a unique algebra morphism from Σ^*0 to L^ω. In our example, the Σ-algebra on L^ω is given by Eqs. (2) and (4), and this *initial semantics* maps, e.g., `alt(a, b)` to $ababab\cdots$.

On the other hand, the distributive law $\lambda^{\mathtt{alt+L}}$, via its rule-based presentation from Section 3.3, induces a stream system on Σ^*0 as described in Section 3.4. Since L^ω carries a final stream system, this gives rise to a unique coalgebra morphism,

which again is a function from Σ^*0 to L^ω. In our example, by looking at the appropriate fragment (7) of the term stream system, it is easy to see that e.g. `alt(a, b)` is again mapped to *abababab* $\cdots$ by this *final semantics*.

In fact, the initial semantics and the final semantics coincide: the inductive extension of the algebra of operations on L^ω equals the coinductive extension of the term stream system induced by $\lambda^{\mathtt{alt}+\mathtt{L}}$. As we shall see in Section 4, this result holds for any distributive law.

One consequence of this coincidence is that in the term stream system induced by a distributive law, the behavioral equivalence relation is a congruence, i.e, it is preserved by all operations in the language under definition. Indeed, recall that behavioral equivalence is the kernel relation of the coinductive extension of the system (the final semantics), and since that coinductive extension is also an algebra morphism (the initial semantics), its kernel relation is a congruence.

Compositionality of behavioral equivalence might seem like a trivial observation in our simple examples, but in more complex flavors of structural operational semantics, it is one of the most useful application of distributive laws, as we shall see later in this paper.

4. Simple distributive laws

We shall now cast the development of Section 3 in a more general setting, and present a basic theory of simple distributive laws for an arbitrary behavior functor.

Definition 11. A simple distributive law of an endofunctor Σ over an endofunctor B on the same category $\mathcal{C}$ is a natural transformation:

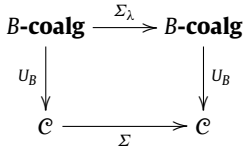
$$\lambda : \Sigma B \Longrightarrow B\Sigma.$$

4.1. Algebras on coalgebras

For any coalgebra $h : X \to BX$, a simple distributive law λ defines a B-coalgebra on ΣX, denoted and defined by:

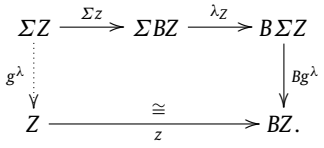
$$\Sigma_\lambda(h) : \Sigma X \to B\Sigma X \qquad \Sigma_\lambda(h) = \lambda_X \circ \Sigma h.$$

It is easy to see that this construction extends to an endofunctor Σ_λ on the category of coalgebras B-**coalg**, acting as Σ on coalgebra morphisms. This functor is a *lifting* of Σ to B-**coalg**, i.e., the diagram:



commutes.

If $z : Z \to BZ$ is a final B-coalgebra, a Σ-algebra $g^\lambda : \Sigma Z \to Z$ is defined as a unique Σ_λ-algebra structure on z, i.e., a unique coalgebra morphism from $\Sigma_\lambda(z)$ to z:



(8)

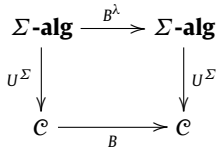
In examples of Section 3, this is the algebra of operations on the carrier of a final coalgebra, defined by a distributive law.

4.2. Coalgebras on algebras

Dually, for any algebra $g : \Sigma X \to X$, a distributive law λ defines a Σ-algebra on BX, denoted and defined by:

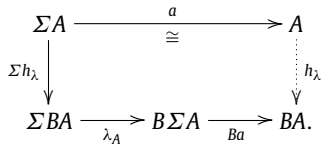
$$B^\lambda(g) : \Sigma BX \to BX \qquad B^\lambda(g) = Bg \circ \lambda_X.$$

This extends to an endofunctor B^λ on the category of algebras Σ-**alg**, acting as B on coalgebra morphisms. This functor is a lifting of B to Σ-**alg**, i.e., the diagram:



commutes.

If $a : \Sigma A \to A$ is an initial Σ-algebra, a B-coalgebra $h_\lambda : A \to BA$ is defined as a unique B^λ-coalgebra structure on a, i.e., a unique algebra morphism from a to $B^\lambda(a)$:

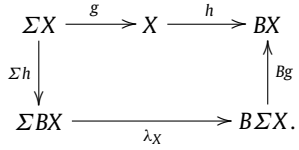


(9)

This shall be called the *coalgebra induced by* λ. In particular, if $\mathcal{C} = \mathbf{Set}$ and $a : \Sigma\Sigma^*0 \to \Sigma^*0$ is the initial algebra of closed Σ-terms, h_λ provides coalgebraic behavior of closed Σ-terms.

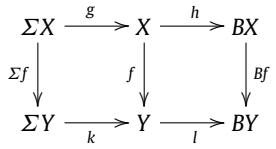
4.3. Bialgebras

For a simple distributive law $\lambda : \Sigma B \Longrightarrow B\Sigma$, a λ-*bialgebra* is a Σ-algebra $g : \Sigma X \to X$ together with a B-coalgebra $h : X \to BX$ with the same carrier, such that the following pentagon commutes:



(10)

A λ-bialgebra *morphism* from $\Sigma X \xrightarrow{g} X \xrightarrow{h} BX$ to $\Sigma Y \xrightarrow{k} Y \xrightarrow{l} BY$ is a morphism $f : X \to Y$ in $\mathcal{C}$ that is a Σ-algebra morphism and a B-coalgebra morphism at the same time, i.e., such that the diagram:



commutes. Bialgebras thus form a category λ-**bialg**.

Recalling the definition of Σ_λ from Section 4.1, note that a Σ_λ-algebra is:

- a B-coalgebra $h : X \to BX$, with
- a Σ_λ-algebra structure on h, i.e., a B-coalgebra morphism g from $\Sigma_\lambda(h)$ to h.

It is easy to check that the latter condition on g is exactly the diagram (10), therefore Σ_λ-algebras are exactly λ-bialgebras. This correspondence easily extends to Σ_λ-algebra and λ-bialgebra morphisms, and together with a dual argument for B^λ-coalgebras we obtain:

Proposition 12. *There is an isomorphism of categories:*

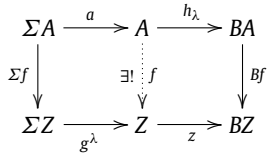
$$\Sigma_\lambda\textbf{-}\boldsymbol{alg} \cong \lambda\textbf{-}\boldsymbol{bialg} \cong B^\lambda\textbf{-}\boldsymbol{coalg}. \quad \square$$

Note that for any endofunctor on a category with a final object, the (necessarily unique) algebra structure on the final object is a final algebra. Dually, (necessarily unique) coalgebras on initial objects are initial coalgebras. Using the isomorphisms of Proposition 12, it immediately follows that, for a simple distributive law $\lambda : \Sigma B \Longrightarrow B\Sigma$:

- every final B-coalgebra extends (uniquely) to a final λ-bialgebra, and
- every initial Σ-algebra extends (uniquely) to an initial λ-bialgebra.

Note that the diagrams (8) and (9) define λ-bialgebras; the former is a final, the latter an initial one. As a result, the B-coalgebra induced by λ is the coalgebraic part of an initial λ-bialgebra.

Consider now the unique bialgebra morphism from the initial to the final one:



where the top row is taken from (9), and the bottom from (8). Note that since a is an initial Σ-algebra and z is a final B-coalgebra, f is at the same time initial semantics (i.e. the inductive extension of g^λ) and final semantics (i.e. the coinductive extension of h_λ). In the context of SOS, the most useful way to state this coincidence is the following theorem, stated under the assumption that initial Σ-algebras and final B-coalgebras exist:

Theorem 13. *For any simple distributive law λ, the coinductive extension of the B-coalgebra h_λ induced by λ is an algebra morphism from the initial Σ-algebra.* □

For $\mathcal{C} = \mathbf{Set}$, this implies that observational equivalence (i.e., the kernel relation of the coinductive extension) on h_λ is a congruence (i.e., the kernel relation of an algebra morphism) on the initial algebra of closed Σ-terms.

5. Simple stream SOS

We shall now again concentrate on the case of stream systems, where $BX = L \times X$ for a fixed set L of labels, and see how simple distributive laws can be presented in terms of inference rules, where the natural notion of a stream system inferred by a set of rules corresponds to the coalgebra induced by a distributive law.

5.1. Simple stream SOS specifications

Let Ξ be a fixed infinite set of metavariables, ranged over by $\mathtt{x}, \mathtt{x}', \mathtt{y}, \mathtt{y}', \ldots$. For an algebraic signature Σ, a *stream literal* is an expression $\mathtt{s} \xrightarrow{a} \mathtt{t}$, where $\mathtt{s}$ (called the *source* of the literal) and $\mathtt{t}$ (the *target*) are Σ-terms with variables from Ξ, and $a \in L$ is called the *label* of the literal. A literal is *closed* if no variables occur in $\mathtt{s}$ or $\mathtt{t}$. A *stream rule* is an expression of the form $\frac{H}{l}$, where H is a set of literals called *premises* and l is a literal called the *conclusion*. The *source* and *target* of a rule are the source and target of its conclusion, respectively.

Definition 14. A *simple stream SOS rule* is a stream rule of the form:

$$\frac{\mathtt{x}_1 \xrightarrow{a_1} \mathtt{x}'_1 \quad \cdots \quad \mathtt{x}_n \xrightarrow{a_n} \mathtt{x}'_n}{\mathtt{f}(\mathtt{x}_1, \ldots, \mathtt{x}_n) \xrightarrow{b} \mathtt{g}(\mathtt{y}_1, \ldots, \mathtt{y}_m)} \tag{11}$$

where:

- $\mathtt{f}$ and $\mathtt{g}$ are operations in Σ of arity n and m respectively,
- $\mathtt{x}_1, \ldots, \mathtt{x}_n, \mathtt{x}'_1, \ldots, \mathtt{x}'_n \in \Xi$ are pairwise distinct variables,
- $\mathtt{y}_j \in \{\mathtt{x}'_1, \ldots, \mathtt{x}'_n\}$ for each $j = 1..m$,
- $b, a_1, \ldots, a_n \in L$.

We shall say that a rule as above is a rule *for* the operator $\mathtt{f} \in \Sigma$, and is *triggered* by the tuple $(a_1, \ldots, a_n)$ of labels.

Note that up to bijective renaming of variables, a simple stream SOS rule R can be presented as a tuple:

$$R = (\mathtt{f}, \mathtt{g}, (a_1, \ldots, a_n), b, \theta), \tag{12}$$

comprising:

- operations $\mathtt{f}$ and $\mathtt{g}$ in Σ of arity n and m,
- the triggering tuple $(a_1, \ldots, a_n)$ of premise labels,
- the conclusion label $b \in L$,
- a function $\theta : \{1, \ldots, m\} \to \{1, \ldots, n\}$ that determines the choice and order of variables in the rule target.

Everything else in (11) is syntactic sugar.

Definition 15. A *simple stream SOS specification* for Σ is a set Λ of simple stream SOS rules such that for each $\mathtt{f} \in \Sigma$ (of arity, say, n), and each tuple $\vec{a} = (a_1, \ldots, a_n) \in L^n$, there is exactly one rule in Λ for $\mathtt{f}$ that is triggered by $\vec{a}$.

5.2. From simple stream SOS to distributive laws and back

Given a signature Σ and a set of labels L, a simple stream SOS specification Λ defines a distributive law $\lambda : \Sigma(L \times -) \Longrightarrow (L \times -)\Sigma$, as follows. For an arbitrary set X, we shall define the value of the component function $\lambda_X : \Sigma(L \times X) \to L \times \Sigma X$ on an argument

$$s = \mathtt{f}((a_1, x_1), \ldots, (a_n, x_n)).$$

To this end, it is useful to write down s as

$$s = \mathtt{f}[\sigma] \tag{13}$$

where $\sigma : \{1, \ldots, n\} \to L \times X$ is defined by $\sigma(i) = (a_i, x_i)$. To define $\lambda_X(s)$, let

$$R = (\mathtt{f}, \mathtt{g}, (a_1, \ldots, a_n), b, \theta)$$

be the unique rule for $\mathtt{f}$ in Λ that is triggered by $(a_1, \ldots, a_n)$, presented as in (12); then put

$$\lambda_X(s) = (b, \mathtt{g}[\pi_2 \circ \sigma \circ \theta])$$

where $\pi_2 : L \times X \to X$ is the projection function, and notation analogous to (13) is used on the right-hand side. Using the more syntactic presentation (11), this can be written as

$$\lambda_X(s) = (b, t[\mathtt{x}'_1 \mapsto x_1, \ldots, \mathtt{x}'_n \mapsto x_n]), \tag{14}$$

where b and $t = \mathtt{g}(\mathtt{y}_1, \ldots, \mathtt{y}_m)$ are taken from R as in (11). Note that since $\{\mathtt{y}_1, \ldots, \mathtt{y}_m\} \subseteq \{\mathtt{x}'_1, \ldots, \mathtt{x}'_n\}$, the result of the above substitution on t is a well-defined element of ΣX.

Proposition 16. *As constructed above, $\lambda : \Sigma(A \times -) \Longrightarrow (A \times -)\Sigma$ is a natural transformation.*

Proof. For any function $g : X \to Y$, one needs to check that $\lambda_Y \circ \Sigma(L \times g) = (L \times \Sigma g) \circ \lambda_X$. To this end, pick any $s = \mathtt{f}((a_1, x_1), \ldots, (a_n, x_n)) \in \Sigma(L \times X)$. Then

$$(L \times \Sigma g)(\lambda_X(s)) = (b, t[\mathtt{x}'_1 \mapsto g(x_1), \ldots, \mathtt{x}'_n \mapsto g(x_n)]),$$

where b and t come from the unique rule R for $\mathtt{f}$ triggered by $(a_1, \ldots, a_n)$. Note that the same rule R is picked in the calculation of $\lambda_Y(\Sigma(L \times g)(s))$ according to (14), as

$$(\Sigma(L \times g)(s)) = \mathtt{f}((a_1, g(x_1)), \ldots, (a_n, g(x_n)))$$

has the same principal operator $\mathtt{f}$ and the sequence of argument labels $(a_1, \ldots, a_n)$ as s. As a result, by (14),

$$\lambda_Y((\Sigma(L \times g)(s))) = (b, t[\mathtt{x}'_1 \mapsto g(x_1), \ldots, \mathtt{x}'_n \mapsto g(x_n)])$$

hence $\lambda_Y((\Sigma(L \times g)(s))) = (L \times \Sigma g)(\lambda_X(s))$. □

Every distributive law $\lambda : \Sigma(L \times -) \Longrightarrow (L \times -)\Sigma$ arises in this way from a simple stream SOS specification. Indeed, for any $\mathtt{f} \in \Sigma$ of arity n and any $\vec{a} = (a_1, \ldots, a_n) \in L^n$, pick any set of $2n$ distinct variables $\mathtt{x}_1, \ldots, \mathtt{x}_n, \mathtt{x}'_1, \ldots, \mathtt{x}'_n \in \Xi$ and consider a rule

$$\frac{\mathtt{x}_1 \xrightarrow{a_1} \mathtt{x}'_1 \quad \cdots \quad \mathtt{x}_n \xrightarrow{a_n} \mathtt{x}'_n}{\mathtt{f}(\mathtt{x}_1, \ldots, \mathtt{x}_n) \xrightarrow{b} \mathtt{g}(\mathtt{y}_1, \ldots, \mathtt{y}_m)}$$

where

$$(b, \mathtt{g}(\mathtt{y}_1, \ldots, \mathtt{y}_m)) = \lambda_X(\mathtt{f}((a_1, \mathtt{x}'_1), \ldots, (a_n, \mathtt{x}'_n)))$$

for $X = \{\mathtt{x}'_1, \ldots, \mathtt{x}'_n\} \subseteq \Xi$. Since by definition $\mathtt{y}_1, \ldots, \mathtt{y}_m \in X$, this is a simple stream SOS rule, and the collection of such rules for each $\mathtt{f}$ and $\vec{a}$ forms a simple stream SOS specification. It is straightforward to check that λ arises from this specification as in (14).

5.3. Behavior of terms

Given a set Λ of stream rules, a *proof* is an upwardly branching tree of finite depth, with nodes labeled by closed literals, such that if H is the set of labels of nodes directly above a node with label l, then $\frac{H}{l}$ is an instance of some rule R in Λ (i.e., it arises from R by some substitution $v : \Xi \to \Sigma^*0$). A closed literal l is *provable* from Λ if there is a proof with the root labeled with l.

Proposition 17. *If Λ is a simple stream SOS specification then for each $s \in \Sigma^*0$ there exist unique $a \in L$ and $t \in \Sigma^*0$ such that $\mathtt{s} \xrightarrow{a} \mathtt{t}$ is provable from Λ.*

Proof. Straightforward induction on the structure of $\mathtt{s}$. □

For a simple stream SOS specification Λ, the stream system h_Λ on Σ^*0 defined by:

$$h_\Lambda(s) = (a, t) \quad \text{where } \mathtt{s} \xrightarrow{a} \mathtt{t} \text{ is provable from } \Lambda$$

is called the system *inferred* from Λ. This is a well-defined stream system by Proposition 17.

Proposition 18. *For a distributive law λ presented by a simple stream SOS specification Λ, the stream coalgebra h_λ induced from λ coincides with the stream system h_Λ inferred from Λ.*

Proof. Looking at the definition (9) of h_λ, it is enough to show that the diagram

$$\begin{array}{ccccc} \Sigma\Sigma^*0 & & \xrightarrow{\quad a \quad} & & \Sigma^*0 \\ & & \cong & & \\ {\scriptstyle \Sigma h_\Lambda}\downarrow & & & & \downarrow{\scriptstyle h_\Lambda} \\ \Sigma B\Sigma^*0 & \xrightarrow[\lambda_{\Sigma^*0}]{} & B\Sigma\Sigma^*0 & \xrightarrow[Ba]{} & B\Sigma^*0 \end{array} \tag{15}$$

commutes. To this end, take any $s = \mathtt{f}(s_1, \ldots, s_n) \in \Sigma\Sigma^*0$ and note that, by definition of provability, $s \xrightarrow{a} t$ is provable from Λ if and only if for the (necessarily unique, by Proposition 17) $s_i \xrightarrow{a_i} t_i$ provable from Λ,

$$\frac{s_1 \xrightarrow{a_1} t_1 \quad \cdots \quad s_n \xrightarrow{a_n} t_n}{s \xrightarrow{a} t}$$

is an instance of a rule in Λ. On the other hand, by the definition of λ from Λ, the latter is equivalent to saying that

$$\lambda_{\Sigma^*0}(\mathtt{f}((a_1, t_1), \ldots, (a_n, t_n))) = (a, t);$$

from this (15) follows. □

From Theorem 13 we thus obtain:

Corollary 19. *For any simple stream SOS specification Λ, observational equivalence on the stream system inferred from Λ is a congruence.* □

6. More distributive laws

The framework of simple distributive laws has rather limited expressive power. There are many operations on final coalgebras, and well-behaved coalgebra structures on initial algebras, that cannot be presented via simple distributive laws. In this section we shall see a few examples based on stream systems, and present a list of progressively complex types of distributive laws that are able to cover these examples: copointed, pointed and bipointed laws, GSOS and coGSOS laws. A version of Theorem 13 can be proved for each of these types, but there is little need to prove each of them separately, as all types of laws are subsumed by a single one: distributive laws of monads over comonads. Note that it is still useful to distinguish the more restrictive types of laws since, as it turns out, distributive laws of monads over comonads in general lack an easy presentation in terms of rules, even in the relatively simple case of stream systems. Also, it should be interesting to observe how different types of laws correspond to more or less permissive formats of stream SOS specifications.

6.1. Copointed laws

As an alternative to Example 7, consider a "zipping composition" operation on infinite streams:

$$\mathtt{zip} : (L^\omega)^2 \to L^\omega$$

acting as follows:

$$\mathtt{zip}(a_1a_2a_3a_4\cdots, b_1b_2b_3b_4\cdots) = a_1b_1a_2b_2a_3\cdots$$

As in Section 3.1, this operation arises as the coinductive extension of an $(L \times -)$-coalgebra structure on $(L^\omega)^2$, denoted and defined by:

$$h^{\mathtt{zip}} : (L^\omega)^2 \to L \times (L^\omega)^2 \qquad h^{\mathtt{zip}}(\alpha, \beta) = \big(\mathtt{hd}(\alpha)\,,\ (\beta, \mathtt{tl}(\alpha))\big)$$

(compare with (3)). It might be encouraging to see that, similarly to $h^{\mathtt{alt}}$ in Section 3.2, $h^{\mathtt{zip}}$ factors through the pointwise application of the final coalgebra $z : L^\omega \to L \times L^\omega$:

$$h^{\mathtt{zip}} = k \circ (z \times z) \qquad k(a, \alpha, b, \beta) = (a, b.\beta, \alpha)$$

where $\_.\_ : L \times L^\omega \to L^\omega$ is the obvious prefixing operation, i.e., the inverse of z. However, the similarity ends here: $\_.\_$ is not a natural transformation, and so k does not extend to a distributive law of $(-)^2$ over $(L \times -)$.

Somewhat informally at this stage, this problem can also be explained in terms of inference rules. Intuitively, it is reasonably clear that the operation zip can be presented with a family of rules:

$$\frac{\mathtt{x} \xrightarrow{a} \mathtt{x}'}{\mathtt{zip(x, y)} \xrightarrow{a} \mathtt{zip(y, x')}} \tag{16}$$

where a ranges over L. This, however, is not a simple stream SOS rule: the third condition of Definition 14 is violated by the use of the variable y in the rule conclusion.

In spite of these difficulties, the definition of zip can be covered with a slight extension to the notion of simple distributive law. Note that the function $h^{\mathtt{zip}}$ factors through the pointwise application of the function $\langle \mathrm{id}, z\rangle : L^\omega \to L^\omega \times L \times L^\omega$:

$$h^{\mathtt{zip}} = \rho^{\mathtt{zip}}_{L^\omega} \circ \langle \mathrm{id}, z\rangle^2$$

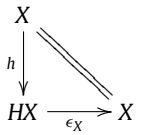
where $\rho^{\mathtt{zip}}_{L^\omega} : (L^\omega \times L \times L^\omega)^2 \to L \times (L^\omega)^2$ is defined by:

$$\rho^{\mathtt{zip}}_{L^\omega}(\alpha, a, \alpha', \beta, b, \beta') = (a, \beta, \alpha')$$

This easily extends to a natural transformation $\rho : (\mathrm{Id} \times B)^2 \Longrightarrow B(-)^2$, for $BX = L \times X$. To extend the framework of Section 4 to transformations of this type, it is convenient to speak in terms of copointed functors.

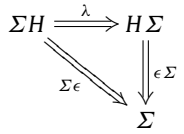
Definition 20. A copointed endofunctor (H, ϵ) on a category $\mathcal{C}$ is an endofunctor H on $\mathcal{C}$ together with a natural transformation $\epsilon : H \Longrightarrow \mathrm{Id}$, called the *counit*.

Definition 21. A copointed coalgebra for a copointed functor (H, ϵ) is a H-coalgebra $h : X \to HX$ such that the diagram



commutes.

Definition 22. A distributive law of an endofunctor Σ over a copointed functor (H, ϵ) is a natural transformation $\lambda : \Sigma H \Longrightarrow H\Sigma$ such that the diagram

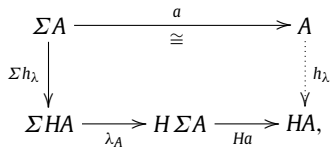


commutes.

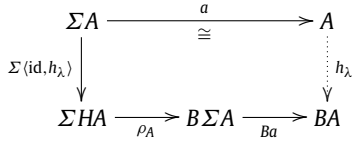
If $\mathcal{C}$ has binary products then for any endofunctor B, the functor $\mathrm{Id} \times B$ is copointed, with ϵ the first projection. It is then called the cofree copointed endofunctor over B. It is easy to prove that copointed $(\mathrm{Id} \times B)$-coalgebras bijectively correspond to B-coalgebras.

Proposition 23. *Distributive laws of an endofunctor Σ over the copointed endofunctor $(Id \times B)$ are in one-to-one correspondence with natural transformations $\rho : \Sigma(Id \times B) \Longrightarrow B\Sigma$.* □

One can now repeat the development of Section 4, replacing the endofunctor B with the copointed endofunctor $H = \mathrm{Id} \times B$, and B-coalgebras with copointed H-coalgebras throughout. In particular, the definition (9) of the coalgebra induced by a distributive law translates to



and h_λ is copointed thanks to the axiom of Definition 22, by initiality of a. This can be equivalently rewritten, along the correspondence between copointed H-coalgebras and B-coalgebras, in terms of the corresponding natural transformation ρ as in Proposition 23, with h_λ becoming the unique morphism that makes the diagram:



commute. A counterpart of Theorem 13 then says that for every ρ, the coinductive extension of h_λ is an algebra morphism from a.

For stream systems, where $BX = L \times X$, natural transformations $\rho : \Sigma(\mathrm{Id} \times B) \Longrightarrow B\Sigma$ can be presented in terms of inference rules just as simple distributive laws in Section 5. Indeed, one might simply define:

Definition 24. A *copointed stream SOS rule* is defined as a simple stream SOS rule in Definition 14, with the third condition relaxed to:

- $\mathrm{y}_j \in \{\mathrm{x}_1, \ldots, \mathrm{x}_n, \mathrm{x}'_1, \ldots, \mathrm{x}'_n\}$ for each $j = 1..m$.

A more compact presentation of copointed stream SOS rules is also possible:

$$R = (\mathrm{f}, \mathrm{g}, (a_1, \ldots, a_n), b, \theta),$$

defined as in (12), with the only difference in the type of θ:

$$\theta : \{1, \ldots, m\} \to 2 \times \{1, \ldots, n\};$$

the additional two-element component 2 determines whether variables in the target conclusion come from the sources or from the targets of their corresponding premises.

Definition 25. A *copointed stream SOS specification* is a set of copointed stream SOS rules subject to the condition of Definition 15.

A correspondence of copointed stream SOS specifications with copointed distributive laws is then proved by analogy to the argument of Section 5.2.

Every simple stream SOS specification is immediately also a copointed stream SOS specification. This can be understood at the level of distributive laws: every simple distributive law $\lambda : \Sigma B \Longrightarrow B\Sigma$ easily gives rise to a natural transformation $\rho : \Sigma(\mathrm{Id} \times B) \Longrightarrow B\Sigma$, by composition with a projection natural transformation.

Note that, formally speaking, rule (16) is not a copointed stream SOS rule, as it lacks a premise for the metavariable $\mathtt{y}$. To fit Definition 24, it should be understood as a family of rules

$$\frac{\mathtt{x} \xrightarrow{a} \mathtt{x}' \quad \mathtt{y} \xrightarrow{b} \mathtt{y}'}{\mathtt{zip}(\mathtt{x}, \mathtt{y}) \xrightarrow{a} \mathtt{zip}(\mathtt{y}, \mathtt{x}')}$$

for all $b \in L$. This kind of syntactic sugar is usual in rule-based presentations of more complex types of distributive laws.

6.2. Pointed and bipointed laws

For a fixed $a \in L$, consider a unary "head replacement" operation on infinite streams:

$$a/- : L^\omega \to L^\omega$$

acting as:

$$a/(b_1 b_2 b_3 b_4 \cdots) = a b_2 b_3 b_3 \cdots$$

Any similarity to examples from Section 3.1 ends very quickly here, as there is no coalgebra structure on L^ω for which $a/-$ would be a coinductive extension. For suppose there is a map $h : L^\omega \to L \times L^\omega$ such that the diagram:

$$\begin{array}{ccc} L^\omega & \xrightarrow{h} & L \times L^\omega \\ {\scriptstyle a/-}\downarrow & & \downarrow{\scriptstyle \mathrm{id}\times(a/-)} \\ L^\omega & \xrightarrow[z]{\cong} & L \times L^\omega \end{array}$$

commutes. For every $\alpha \in L^\omega$ one has $\mathtt{hd}(a/\alpha) = a$, hence there must be $\pi_1(h(\alpha)) = a$. But for each h with this property, the constant function mapping every stream to $aaaaaa\cdots$ is the coinductive extension of h, hence $a/-$ cannot be the coinductive extension.

All is not lost, however: the closely related function

$$[\mathrm{id}, a/-] : L^\omega + L^\omega \to L^\omega$$

is a coinductive extension of the coalgebra:

$$h^{a/-} : L^\omega + L^\omega \to L \times (L^\omega + L^\omega) \qquad \begin{array}{rcl} h^{a/-}(\iota_1(\alpha)) & = & \big(\mathtt{hd}(\alpha), \iota_1(\mathtt{tl}(\alpha))\big) \\ h^{a/-}(\iota_2(\alpha)) & = & \big(a, \iota_1(\mathtt{tl}(\alpha))\big) \end{array}$$

which factors through the componentwise application of the final $(L \times -)$-coalgebra z:

$$h^{a/-} = \lambda_{L^\omega}^{a/-} \circ (z + z)$$

where the natural transformation $\lambda^{a/-} : (\mathrm{Id} + \mathrm{Id})(L \times -) \Longrightarrow (L \times -)(\mathrm{Id} + \mathrm{Id})$ is defined by:

$$\lambda_X^{a/-}(\iota_1(b, x)) = (b, \iota_1(x)) \qquad \lambda_X^{a/-}(\iota_2(b, x)) = (a, \iota_1(x)).$$

This situation can be explained in the framework of distributive laws, once the notion of pointed endofunctor is used. The following development is dual to the one of copointed functors and laws in Section 6.1.

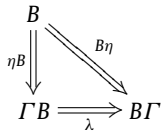
Definition 26. A pointed endofunctor (Γ, η) on a category $\mathcal{C}$ is an endofunctor Γ on $\mathcal{C}$ together with a natural transformation $\eta : \mathrm{Id} \Longrightarrow \Gamma$, called the *unit*.

Definition 27. A (pointed) algebra for a pointed functor (Γ, η) is a Γ-algebra $g : \Gamma X \to X$ such that the diagram

$$\begin{array}{ccc} X & \xrightarrow{\eta X} & \Gamma X \\ & \searrow & \downarrow{\scriptstyle g} \\ & & X \end{array}$$

commutes.

Definition 28. A distributive law of a pointed functor (Γ, η) over an endofunctor B is a natural transformation $\lambda : \Gamma B \Longrightarrow B\Gamma$ such that the diagram



commutes.

If $\mathcal{C}$ has binary coproducts then for any endofunctor Σ, the functor $\mathrm{Id}+\Sigma$ is copointed, with η the first coproduct injection. It is then called the free pointed endofunctor over Σ. Dually to the situation of copointed functors, pointed $\mathrm{Id}+\Sigma$-algebras bijectively correspond to Σ-algebras.

Proposition 29. *Distributive laws of the pointed endofunctor $Id + \Sigma$ over an endofunctor B are in one-to-one correspondence with natural transformations $\rho : \Sigma B \Longrightarrow B(Id + \Sigma)$.* □

As before, for $BX = L \times X$ such natural transformations, and the process of inducing B-coalgebras from them, can be presented in terms of rules and inferences.

Definition 30. A *pointed stream SOS rule* is a stream rule of the form:

$$\frac{\mathtt{x}_1 \xrightarrow{a_1} \mathtt{x}'_1 \quad \cdots \quad \mathtt{x}_n \xrightarrow{a_n} \mathtt{x}'_n}{\mathtt{f}(\mathtt{x}_1, \ldots, \mathtt{x}_n) \xrightarrow{b} \mathtt{t}} \tag{17}$$

where:

- $\mathtt{f}$ is an operation in Σ of arity n,
- $\mathtt{x}_1, \ldots, \mathtt{x}_n, \mathtt{x}'_1, \ldots, \mathtt{x}'_n \in \Xi$ are pairwise distinct variables,
- $\mathtt{t}$ is either a variable in $\{\mathtt{x}'_1, \ldots, \mathtt{x}'_n\}$, or a term of the form $\mathtt{g}(\mathtt{y}_1, \ldots, \mathtt{y}_m)$, where $\mathtt{g}$ is an operation in Σ of arity m and $\mathtt{y}_j \in \{\mathtt{x}'_1, \ldots, \mathtt{x}'_n\}$ for each $j = 1..m$,
- $b, a_1, \ldots, a_n \in L$.

As before, a more compact presentation of copointed stream SOS rules is also possible, where every rule is in one of two forms

$$R_1 = (\mathtt{f}, \mathtt{g}, (a_1, \ldots, a_n), b, \theta) \qquad R_2 = (\mathtt{f}, (a_1, \ldots, a_n), b, k)$$

with R_1 defined as in (12) and R_2 defined similarly, with $k \in \{1, \ldots, n\}$ determining the variable in the conclusion target.

Definition 31. A *pointed stream SOS specification* is a set of pointed stream SOS rules subject to the condition of Definition 15.

For example, the operation $a/-$ explained above can be defined with a pointed stream SOS specification:

$$\frac{\mathtt{x} \xrightarrow{b} \mathtt{x}'}{a/\mathtt{x} \xrightarrow{a} \mathtt{x}'}$$

where b ranges over L.

Every simple stream SOS specification is also a pointed stream SOS specification. At the level of distributive laws, every simple distributive law $\lambda : \Sigma B \Longrightarrow B\Sigma$ easily gives rise to a natural transformation $\rho : \Sigma B \Longrightarrow B(\mathrm{Id} + \Sigma)$, by composition with a coproduct injection.

A common generalization of copointed and pointed distributive laws are *bipointed* ones, i.e., distributive laws of pointed functors over copointed functors subject to obvious axioms. Such a law of a free pointed functor $\mathrm{Id} + \Sigma$ over a cofree copointed functor $\mathrm{Id} \times B$ is equivalent to a natural transformation $\rho : \Sigma(\mathrm{Id} \times B) \Longrightarrow B(\mathrm{Id} + \Sigma)$. For $BX = L \times X$, these can be presented in terms of inference rules; the definition of *bipointed stream SOS* should be evident from Definitions 24 and 30.

One very simple example of a bipointed stream SOS specification is the following definition of a unary prefixing operation $a.-$:

$$\frac{}{a.\mathtt{x} \xrightarrow{a} \mathtt{x}}$$

understood as shorthand for a family of rules

$$\frac{\mathtt{x} \xrightarrow{b} \mathtt{x}'}{a.\mathtt{x} \xrightarrow{a} \mathtt{x}}$$

where b ranges over L. As before, a version of Theorem 13 implies that observational equivalence on systems induced from such specifications is always a congruence.

6.3. GSOS laws

Distributive laws can be generalized further, while retaining both the congruence property of observational equivalence (Theorem 13) and convenient rule-based presentations for specific behavior functors B. We shall now study the class of laws with the most practical importance: GSOS distributive laws, whose name will be justified in Section 7.2.

As an example, consider a unary stream operation

$$\mathtt{hdrep} : L^\omega \to L^\omega$$

that repeats the head of its argument at every odd position of the result:

$$\mathtt{hdrep}(a_1a_2a_3a_4\cdots) = a_1a_2a_1a_3a_1a_4a_1\cdots$$

With our interpretation of stream rules and proofs, it is natural to specify this operation by rules:

$$\frac{\mathtt{x} \xrightarrow{a} \mathtt{x}'}{\mathtt{hdrep(x)} \xrightarrow{a} \mathtt{zip(x', a)}}$$

where a (with the corresponding $\mathtt{a}$) ranges over L and operations $\mathtt{zip}$, $\mathtt{a}$ are specified as before.

Note, however, that the rule above is not in either of the stream rule formats described so far, as the target of its conclusion is a complex term built of two operation symbols. Although it is possible to define $\mathtt{hdrep}$ with a copointed stream SOS specification:

$$\frac{\mathtt{x} \xrightarrow{a} \mathtt{x}'}{\mathtt{hdrep(x)} \xrightarrow{a} \mathtt{zipl}_a\mathtt{(x')}} \qquad \frac{\mathtt{x} \xrightarrow{b} \mathtt{x}'}{\mathtt{zipl}_a\mathtt{(x)} \xrightarrow{b} \mathtt{zipr}_a\mathtt{(x')}} \qquad \frac{}{\mathtt{zipr}_a\mathtt{(x)} \xrightarrow{a} \mathtt{zipl}_a\mathtt{(x)}}$$

(where a, b range over L and a redundant premise is elided in the rightmost rule), this comes at a price: the language signature needs to be extended with a potentially infinite family of auxiliary operations $\mathtt{zipl}_a$ and $\mathtt{zipr}_a$.

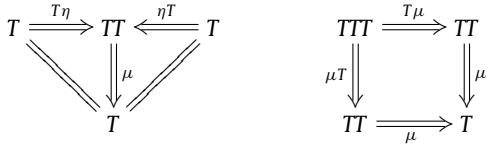
Instead, one may understand the rules for $\mathtt{hdrep}$ (together with ones for $\mathtt{zip}$ and $\mathtt{a}$) directly as a natural transformation

$$\rho : \Sigma(\mathrm{Id} \times B) \Longrightarrow B\Sigma^* \tag{18}$$

where $BX = L \times X$, $\Sigma X = X + X^2 + L$ arises from the basic signature of operations, and Σ^* is the term construction functor mentioned in Section 2.1. The use of Σ^* in the codomain of ρ corresponds the use of a complex term as a rule conclusion target, just as the use of $\mathrm{Id} + \Sigma$ in pointed laws corresponded to the use of variables as conclusion targets.

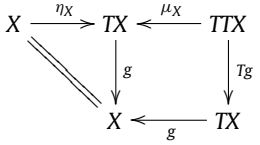
To understand such transformations in the framework of distributive laws, it is convenient to use copointed functors (see Section 6.1) and the standard categorical notion of a monad, which extends that of a pointed functor.

Definition 32. A monad (T, η, μ) on a category $\mathcal{C}$ is an endofunctor T on $\mathcal{C}$ together with natural transformation $\eta : \mathrm{Id} \Longrightarrow T$ (called the *unit*) and $\mu : TT \Longrightarrow T$ (called the *multiplication*), such that the diagrams:



commute.

Definition 33. An Eilenberg–Moore algebra for a monad (T, η, μ) is a T-algebra $g : TX \to X$ such that the diagram



commutes. With ordinary T-algebra morphisms, these form a category T**-Alg**.

For example, given an algebraic signature Σ, the term functor Σ^* together with obvious natural transformations $\eta : \mathrm{Id} \Longrightarrow \Sigma^*$ (interpretation of variables as terms) and $\mu : \Sigma^*\Sigma^* \Longrightarrow \Sigma^*$ (glueing terms built of terms) is a monad. It is called the free monad over Σ. By structural induction on terms, any algebra $g : \Sigma X \to X$ induces a function $g^\sharp : \Sigma^* X \to X$ (i.e. term interpretation in g). The Σ^*-algebra $g^\sharp$ is an Eilenberg–Moore algebra for the monad Σ^*. The construction of $g^\sharp$ from g provides a one-to-one correspondence between Σ-algebras and Eilenberg–Moore Σ^*-algebras, and an isomorphism of categories

$$\Sigma\textbf{-alg} \cong \Sigma^*\textbf{-Alg}.$$

In general, for any endofunctor Σ on a category $\mathcal{C}$, if the forgetful functor $U^\Sigma : \Sigma\textbf{-alg} \to \mathcal{C}$ has a left adjoint F^Σ, then the monad $\Sigma^* = U^\Sigma F^\Sigma$ arising from the adjunction is called the free monad over Σ, and the correspondence between Σ-algebras and Eilenberg–Moore Σ^*-algebras still holds. If $\mathcal{C}$ has coproducts then $\Sigma^* X$ is the carrier of an initial algebra for the functor $\Sigma_X Y = X + \Sigma Y$.

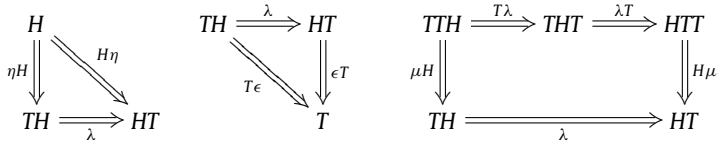
From now on, assume endofunctors Σ, B on a category $\mathcal{C}$ with products, such that a free monad Σ^* over Σ exists. In such a situation (18) makes sense:

Definition 34. A GSOS law (of Σ over B) is a natural transformation

$$\rho : \Sigma(\mathrm{Id} \times B) \Longrightarrow B\Sigma^*.$$

Using copointed functors, GSOS laws can be seen as distributive laws.

Definition 35. A distributive law of a monad (T, η, μ) over a copointed functor (H, ϵ) is a natural transformation $\lambda : TH \Longrightarrow HT$ such that the diagrams:



commute.

Proposition 36. *GSOS laws of Σ over B are in one-to-one correspondence with distributive laws of the free monad Σ^* over the cofree copointed endofunctor $\mathrm{Id} \times B$.*

Proof. This is not quite as obvious as Propositions 23 and 29. A proof can be found in [33]. □

For $BX = L \times X$ on **Set** and Σ arising from a signature, GSOS laws can be presented in terms of inference rules much as bipointed stream SOS rules (compare Definition 30):

Definition 37. A *stream GSOS rule* is a stream rule of the form:

$$\frac{\mathtt{x}_1 \xrightarrow{a_1} \mathtt{x}'_1 \quad \cdots \quad \mathtt{x}_n \xrightarrow{a_n} \mathtt{x}'_n}{\mathtt{f}(\mathtt{x}_1, \ldots, \mathtt{x}_n) \xrightarrow{b} \mathtt{t}} \tag{19}$$

where:

- $\mathtt{f}$ is an operation in Σ of arity n,
- $\mathtt{x}_1, \ldots, \mathtt{x}_n, \mathtt{x}'_1, \ldots, \mathtt{x}'_n \in \Xi$ are pairwise distinct variables,
- $\mathtt{t}$ is a Σ-term built over variables $\{\mathtt{x}_1, \ldots, \mathtt{x}_n, \mathtt{x}'_1, \ldots, \mathtt{x}'_n\}$,
- $b, a_1, \ldots, a_n \in L$.

As before, some syntactic sugar can be removed from stream GSOS rules. A more compact representation of a rule is a tuple:

$$R = (\mathtt{f}, (a_1, \ldots, a_n), b, \mathtt{t}),$$

comprising:

- an operations $\mathtt{f}$ in Σ of arity n,
- the triggering tuple $(a_1, \ldots, a_n)$ of premise labels,
- the conclusion label $b \in L$,
- a Σ-term $\mathtt{t}$ over a fixed set of $2n$ variables.

Definition 38. A *stream GSOS specification* is a set of stream GSOS rules subject to the condition of Definition 15.

Proposition 39. *Every stream GSOS specification gives rise to a GSOS law for $BX = L \times X$, and every such GSOS law arises from a stream GSOS specification.*

Proof. By analogy to Proposition 5.2. □

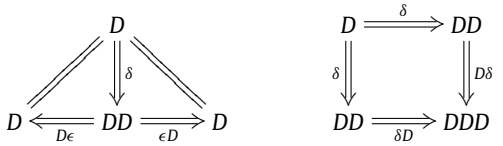
From general results in Section 6.5 it will follow that behavioral equivalence on coalgebras induced by GSOS laws is a congruence.

Note that every bipointed stream SOS specification is a stream GSOS specification. At the level of distributive laws, any natural transformation $\rho : \Sigma(\mathrm{Id} \times B) \Longrightarrow B(\mathrm{Id} + \Sigma)$ immediately yields a GSOS law by composition with the obvious natural inclusion from $\mathrm{Id} + \Sigma$ to Σ^*.

6.4. coGSOS laws

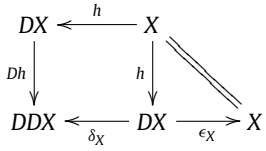
Dually to GSOS laws, one can generalize bipointed laws to natural transformations that involve cofree comonads over behavior functors.

Definition 40. A comonad (D, ϵ, δ) on a category $\mathcal{C}$ is an endofunctor D on $\mathcal{C}$ together with natural transformation $\epsilon : D \Longrightarrow \mathrm{Id}$ (called the *counit*) and $\delta : D \Longrightarrow DD$ (called the *comultiplication*), such that the diagrams:



commute.

Definition 41. An Eilenberg–Moore coalgebra for a comonad (D, ϵ, δ) is a D-coalgebra $h : X \to DX$ such that the diagram



commutes. With ordinary D-coalgebra morphisms, these form a category D-**Coalg**.

If the forgetful functor $U_B : B\text{-}\mathbf{coalg} \to \mathcal{C}$ has a right adjoint G_B, then the resulting comonad $B^\infty = U_B G_B$ on $\mathcal{C}$ is called the cofree comonad over B. Dually to the situation with monads, Eilenberg–Moore B^∞-coalgebras are in bijective correspondence with B-coalgebras:

$$B^\infty\text{-}\mathbf{Coalg} \cong B\text{-}\mathbf{coalg}.$$

If $\mathcal{C}$ has products then $B^\infty X$ is the carrier of a final coalgebra for the functor $B_X Y = X \times BY$.

For example, the cofree comonad over $BX = L \times X$ on **Set** is $B^\infty X = (L \times X)^\omega$, with counit and comultiplication defined by:

$$\epsilon_X\big((a_1, x_1)(a_2, x_2)(a_3, x_3)\cdots\big) = x_1$$

$$\delta_X\big((a_1, x_1)(a_2, x_2)(a_3, x_3)\cdots\big) = \Big(\big(a_1, (a_1, x_1)(a_2, x_2)\cdots\big)\ \big(a_2, (a_2, x_2)(a_3, x_3)\cdots\big)\ \cdots\Big).$$

The Eilenberg–Moore $(L \times -)^\omega$-coalgebra corresponding to a stream system $h : X \to L \times X$ maps every state $x \in X$ to the stream of labels and states produced by h starting from x.

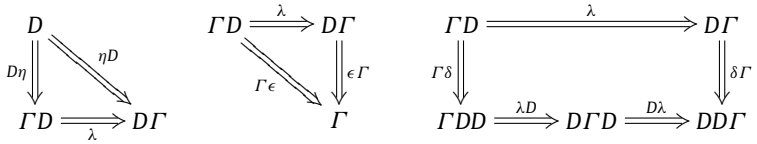
Assume endofunctors Σ, B on a category $\mathcal{C}$ with coproducts, such that a cofree comonad B^∞ over B exists.

Definition 42. A coGSOS law (of Σ over B) is a natural transformation

$$\rho : \Sigma B^\infty \Longrightarrow B(\mathrm{Id} + \Sigma).$$

This can be cast in the framework of distributive laws dually to Definition 34:

Definition 43. A distributive law of a pointed functor (Γ, η) over a comonad (D, ϵ, δ) is a natural transformation $\lambda : \Gamma D \Longrightarrow D\Gamma$ such that the diagrams:



commute.

Proposition 44. *CoGSOS laws of Σ over B are in one-to-one correspondence with distributive laws of the free pointed endofunctor $\mathrm{Id} + \Sigma$ over the cofree comonad B^∞.*

Proof. Dual to Proposition 36. □

Again, for $BX = L \times X$ and polynomial Σ, coGSOS laws can be presented in terms of inference rules:

Definition 45. A *stream coGSOS rule* is a stream rule of the form:

$$\frac{\begin{array}{cccccc} \mathtt{x}_1 \xrightarrow{a_1^1} \mathtt{x}_1' & \mathtt{x}_1' \xrightarrow{a_1^2} \mathtt{x}_1'' & \cdots & \mathtt{x}_1^{(i-1)} \xrightarrow{a_1^i} \mathtt{x}_1^{(i)} & \cdots \\ \vdots & \vdots & & \vdots & \\ \mathtt{x}_n \xrightarrow{a_n^1} \mathtt{x}_n' & \mathtt{x}_n' \xrightarrow{a_n^2} \mathtt{x}_n'' & \cdots & \mathtt{x}_n^{(i-1)} \xrightarrow{a_n^i} \mathtt{x}_n^{(i)} & \cdots \end{array}}{\mathtt{f}(\mathtt{x}_1, \ldots, \mathtt{x}_n) \xrightarrow{b} \mathtt{t}}$$

where:

- $\mathtt{f}$ is an operation in Σ of arity n,
- $\mathtt{x}_1, \ldots, \mathtt{x}_n, \mathtt{x}_1', \ldots, \mathtt{x}_n', \ldots, \mathtt{x}_1^{(i)}, \ldots, \mathtt{x}_n^{(i)}, \ldots \in \Xi$ are pairwise distinct variables,
- $\mathtt{t}$ is either a variable that occurs in one of the premises, or a term of the form $\mathtt{g}(\mathtt{y}_1, \ldots, \mathtt{y}_m)$, where g is an operation in Σ of arity m and $\mathtt{y}_1, \ldots, \mathtt{y}_m \in \Xi$ are (not necessarily pairwise distinct) variables that all occur in the premises of the rule,
- $b, a_1^1, \ldots, a_n^1, \ldots, a_1^i, \ldots, a_n^i, \ldots \in L$.

We shall say that a rule as above is a rule *for* the operator $\mathtt{f} \in \Sigma$, and is *triggered* by the tuple $\langle (a_1^1 a_1^2 a_1^3 \cdots), \ldots, (a_n^1 a_n^2 a_n^3 \cdots) \rangle$ of streams of labels.

Removing syntactic sugar one obtains a more compact representation, where each stream coGSOS rule is in one of the forms:

$$R_1 = (\mathtt{f}, \mathtt{g}, (\alpha_1, \ldots, \alpha_n), b, \theta) \qquad R_2 = (\mathtt{f}, (\alpha_1, \ldots, \alpha_n), b, i, j)$$

where

- $\mathtt{f}$ and g are operations in Σ of arity n and m,
- $(\alpha_1, \ldots, \alpha_n)$ is the triggering tuple of premise label sequences, where each $\alpha_i \in A^\omega$,
- $b \in L$ is the conclusion label,
- $\theta : \{1, \ldots, m\} \to \mathbb{N} \times \{1, \ldots, n\}$ or $i \in \{1, \ldots, n\}, j \in \mathbb{N}$ determine the choice and order of variables in the conclusion target.

Note that unlike all previous rule formats considered so far, stream coGSOS rules allow *lookahead* in their premises, where variables from targets of premises can appear as sources of other premises. Intuitively, this means that to decide the initial transition of a term $\mathtt{f}(t_1, \ldots, t_n)$ one is allowed to test more than one step of behavior of the subterms $t_1, \ldots, t_n$.

Definition 46. A *stream coGSOS specification* for Σ is a set Λ of stream coGSOS rules such that for each $\mathtt{f} \in \Sigma$ (of arity, say, n), and each tuple $\vec{\alpha} = (\alpha_1, \ldots, \alpha_n) \in (L^\omega)^n$, there is exactly one rule in Λ for $\mathtt{f}$ that is triggered by $\vec{\alpha}$.

Notice that in a stream coGSOS specification, inferred transitions originating in a term $\mathtt{f}(t_1, \ldots, t_n)$ may depend on transitions of terms other than the subterms $t_1, \ldots, t_n$. This means that it takes a little more care to prove that a stream coGSOS specification meaningfully infers a stream system: formally, a counterpart of Proposition 17 is not as obvious as for all classes of laws considered in previous sections. Nevertheless, the proposition is still true thanks to the third condition of Definition 45. Indeed, the condition ensures that no provable transition increases the depth of the term under transition, therefore transitions originating from a given term depend only on transitions of terms of smaller depth and Proposition 17 can be proved by induction on the depth of terms.

Proposition 47. *Every stream coGSOS specification gives rise to a coGSOS law for $BX = L \times X$, and every such coGSOS law arises from a stream coGSOS specification.*

Proof. Recall that $B^\infty X = (L \times X)^\omega$ and proceed by analogy to Section 5.2. □

Typically one should strive for a finite representation of SOS rules and specifications, so a practical coGSOS specification would normally involve some shorthand and syntactic sugar. For example, the unary tail operation $\mathtt{tl}$ on streams can be specified by a family of rules:

$$\frac{\mathtt{x} \xrightarrow{a} \mathtt{x}' \quad \mathtt{x}' \xrightarrow{a'} \mathtt{x}''}{\mathtt{tl}(\mathtt{x}) \xrightarrow{a'} \mathtt{x}''} \tag{20}$$

where a, a' range over L, which should be understood as shorthand for

$$\frac{\mathtt{x} \xrightarrow{a} \mathtt{x}' \quad \mathtt{x}' \xrightarrow{a'} \mathtt{x}'' \quad \cdots \quad \mathtt{x}^{(i-1)} \xrightarrow{a^{(i-1)}} \mathtt{x}^{(i)} \quad \cdots}{\mathtt{tl}(\mathtt{x}) \xrightarrow{a'} \mathtt{x}''}$$

for all $a, a', a'', \ldots \in L$.

Note, however, that the stream coGSOS format does not force one to use any such shorthand, and it allows infinitely many premises in a single rule. As a result, it is straightforward to specify stream operations like $\mathtt{skip}_a$ that removes all a's from its argument except an infinite tail of a's if it is present:

$$\frac{\mathtt{x} \xrightarrow{a} \mathtt{x}' \quad \mathtt{x}' \xrightarrow{a} \mathtt{x}'' \quad \cdots \quad \mathtt{x}^{(n-1)} \xrightarrow{a} \mathtt{x}^{(n)} \quad \mathtt{x}^{(n)} \xrightarrow{b} \mathtt{y}}{\mathtt{skip}_a(\mathtt{x}) \xrightarrow{b} \mathtt{skip}_a(\mathtt{y})}$$

$$\frac{\mathtt{x} \xrightarrow{a} \mathtt{x}' \quad \mathtt{x}' \xrightarrow{a} \mathtt{x}'' \quad \cdots \quad \mathtt{x}^{(i-1)} \xrightarrow{a} \mathtt{x}^{(i)} \quad \cdots}{\mathtt{skip}_a(\mathtt{x}) \xrightarrow{a} \mathtt{x}'}$$

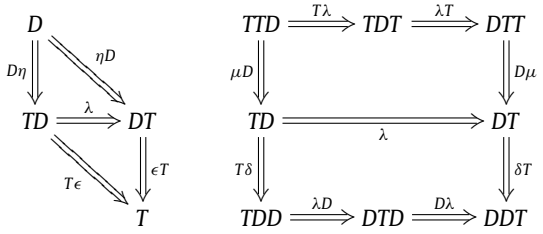
where n ranges over $\mathbb{N}$ and b over $L \setminus \{a\}$. Such operations have little operational sense: note that $\mathtt{skip}_a$ positively detects, in a single step of computation, that its argument will never produce a label different from a. Nevertheless, from general results in Section 6.5 it follows that behavioral equivalence on coalgebras induced by coGSOS laws is a congruence.

As in Section 6.3, every bipointed stream SOS specification is a stream coGSOS specification. At the level of distributive laws, any natural transformation $\rho : \Sigma(\mathrm{Id} \times B) \Longrightarrow B(\mathrm{Id} + \Sigma)$ yields a coGSOS law by composition with the obvious natural projection from B^∞ to $\mathrm{Id} \times B$.

6.5. The general case: distributing monads over comonads

We now move to the most expressive type of distributive laws considered in the bialgebraic study of SOS.

Definition 48. A distributive law of a monad (T, η, μ) over a comonad (D, ϵ, δ) is a natural transformation $\lambda : TD \Longrightarrow DT$ such that the diagrams:



commute.

One can repeat the entire development of Section 4 for distributive laws of monads over comonads, replacing Σ with a monad T, B with a comonad D, Σ-algebras with Eilenberg–Moore T-algebras and B-coalgebras with Eilenberg–Moore D-coalgebras throughout.

It is straightforward to check that, as in Section 4, a distributive law λ of a monad T over a comonad D lifts the monad T to D-**Coalg** and the comonad D to T-**Alg**. In this setting, unlike for simple laws in Section 4, the converse holds as well:

Proposition 49. *For any monad T and comonad D, the following are equivalent:*

- *Distributive laws of T over D,*
- *Liftings of T to D-**Coalg**,*
- *Liftings of D to T-**Alg**.*

Proof. See [54] or [22]. □

Bialgebras for λ are defined as in Section 4.3, and the counterpart of Proposition 12 is proved by routine calculations. As a result, a version of Theorem 13 also holds:

Theorem 50. *For any distributive law λ of T over D, the coinductive extension of the D-coalgebra induced by λ is an algebra morphism from the initial T-algebra.* □

Here the D-coalgebra h induced by λ is defined by analogy to (9):

$$\begin{array}{ccccc} TT0 & & \xrightarrow{\quad\mu_0\quad} & & T0 \\ {\scriptstyle Th}\downarrow & & & & \downarrow{\scriptstyle h} \\ TDT0 & \xrightarrow[\lambda_{T0}]{} & DTT0 & \xrightarrow[D\mu_0]{} & DT0, \end{array} \tag{21}$$

since $\mu_0 : TT0 \to T0$ is an initial Eilenberg–Moore T-algebra.

Distributive laws of monads over comonads generalize all other laws considered so far in this section. For GSOS laws, assuming B admits a cofree comonad B^∞:

Proposition 51. *Every GSOS law $\rho : \Sigma(Id \times B) \Longrightarrow B\Sigma^*$ induces a distributive law $\lambda : \Sigma^* B^\infty \Longrightarrow B^\infty \Sigma^*$ of the free monad Σ^* over the cofree comonad B^∞.*

Proof. This is proved both in [54] and [2], and [33] gives a particularly simple proof in terms of functor and monad liftings, with Proposition 49 as a crucial proof step. □

Moreover, the B-coalgebra induced by a GSOS coincides with the B^∞-coalgebra induced as in (21) by the corresponding distributive law λ, along the correspondence of B-coalgebras and Eilenberg–Moore B^∞-coalgebras. Thus from Theorem 50 it follows that coinductive extensions of B-coalgebras induced by GSOS laws are algebra morphisms.

Dually, it can be proved that coGSOS laws are a special case of monadic distributive laws. The same follows for other, simpler types of laws we considered.

Although distributive laws of monads over comonads offer the most general abstract perspective on well-behaved SOS, so far they have not been applied in concrete studies of SOS rule formats. Indeed, no convenient rule-based characterizations of such laws are known, even for the simplest behavior functors and even if only free monads and cofree comonads are considered.

To illustrate the problem, consider the behavior functor $BX = L \times X$ of stream systems, and any functor Σ corresponding to an algebraic signature. Since distributive laws of Σ^* over B^∞ generalize both GSOS and coGSOS laws of Σ over B, the corresponding format of stream SOS rules should generalize both stream GSOS and stream coGSOS. It is therefore tempting to allow rules where both complex target terms (as in stream GSOS) and lookahead premises (as in stream coGSOS) are allowed. However, this immediately leads into trouble: consider $L = \{a, b\}$ and a language with one constant $\mathtt{c}$ and one unary operation $\mathtt{f}$, "specified" by rules:

$$\frac{}{\mathtt{c} \xrightarrow{a} \mathtt{f(c)}} \qquad \frac{\mathtt{x} \xrightarrow{a} \mathtt{x}' \quad \mathtt{x}' \xrightarrow{a} \mathtt{x}''}{\mathtt{f(x)} \xrightarrow{b} \mathtt{c}} \qquad \frac{\mathtt{x} \xrightarrow{a} \mathtt{x}' \quad \mathtt{x}' \xrightarrow{b} \mathtt{x}''}{\mathtt{f(x)} \xrightarrow{a} \mathtt{c}} \qquad \frac{\mathtt{x} \xrightarrow{b} \mathtt{x}'}{\mathtt{f(x)} \xrightarrow{\cdots} \cdots}$$

where the shorthand convention of (20) is used. The conclusion of the rightmost rule is irrelevant and does not influence the essence of the example. It is easy to see that no unique outgoing transition from the term $\mathtt{f(c)}$ can be inferred from these rules: even if infinite or circular inferences are allowed, $\mathtt{f(c)} \xrightarrow{\mathtt{a}} \mathtt{c}$ is derivable if and only if $\mathtt{f(c)} \xrightarrow{\mathtt{b}} \mathtt{c}$ is. As a result, the above rules do not meaningfully define a stream system. Note how the rule for $\mathtt{c}$ is in the stream GSOS format, and the rules for $\mathtt{f}$ are in the stream coGSOS format, but when put together they do not define a distributive law at all. Similar examples of ill-behaved stream specifications are given e.g. in [6].

Due to these difficulties, and due to the problematic operational meaning of coGSOS-definable operations as explained in Section 6.4, concrete rule-based presentations of distributive laws are almost always restricted to GSOS laws (exceptions include the characterization of general distributive laws for a simple kind of timed systems, see Section 7.4, and a treatment of the tyft/tyxt format for LTSs, see Section 8.5).

7. GSOS formats from distributive laws

We shall sketch the current state of the art in the area of concrete presentations of GSOS laws for various behavior functors. The simple example of stream GSOS was explained in Section 6.3. In all examples that follow (except the last one in Section 7.4), the underlying category is **Set** and syntactic functors Σ arise from algebraic signatures. All the following rule formats use metavariables from a fixed infinite set Ξ.

7.1. Mealy machines

Recall from Example 4 that Mealy machines with input alphabet K and output alphabet L are coalgebras for the functor $BX = (L \times X)^K$. We now give a rule-based characterization of GSOS laws for this behavior functor.

Definition 52. A *Mealy GSOS rule* is an expression of the form:

$$\frac{\mathtt{x}_1 \xrightarrow{a_1|b_1} \mathtt{x}_1' \quad \cdots \quad \mathtt{x}_n \xrightarrow{a_n|b_n} \mathtt{x}_n'}{\mathtt{f}(\mathtt{x}_1, \ldots, \mathtt{x}_n) \xrightarrow{a|b} \mathtt{t}} \tag{22}$$

where:

- $\mathtt{f}$ is an operation in Σ of arity n,
- $\mathtt{x}_1, \ldots, \mathtt{x}_n, \mathtt{x}_1', \ldots, \mathtt{x}_n' \in \Xi$ are pairwise distinct variables,
- $\mathtt{t}$ is a Σ-term built over variables $\{\mathtt{x}_1, \ldots, \mathtt{x}_n, \mathtt{x}_1', \ldots, \mathtt{x}_n'\}$,
- $a, a_1, \ldots, a_n \in K$ and $b, b_1, b_2, \ldots, b_n \in L$.

A rule as above is triggered by a tuple $(f_1, \ldots, f_n)$, where $f_i : K \to L$, if $b_i = f_i(a_i)$ for $i = 1..n$.

Definition 53. A *Mealy GSOS specification* for Σ is a set Λ of Mealy GSOS rules such that for each $\mathtt{f} \in \Sigma$ (of arity, say, n), each label $a \in K$ and tuple $\vec{f} = (f_1, \ldots, f_n) \in (L^K)^n$, there is exactly one rule in Λ for $\mathtt{f}$ that is triggered by $\vec{f}$.

A Mealy machine inferred from a Mealy GSOS specification is defined in a straightforward way, via a standard notion of inference. Using methods similar to those used for stream systems, one then proves:

Proposition 54. *Every Mealy GSOS specification gives rise to a GSOS law for $BX = (L \times X)^K$, and every such law arises from a Mealy GSOS specification in this way.* □

Corollary 55. *Observational equivalence on the Mealy machine inferred from a Mealy GSOS specification is a congruence.* □

As in the case of stream systems, shorthand notation and syntactic sugar might be used to present Mealy GSOS specification in practice. Some small examples of Mealy GSOS specifications can be found in [15].

7.2. Labeled transition systems

Specifications of nondeterministic labeled transition systems are by far the most studied flavor of SOS (see [1] for a survey), and were the original motivating example in [54] for the abstract study of SOS in terms of distributive laws. Recall from Example 5 that (image-finite) labeled transition systems are coalgebras for the functor $BX = (\mathcal{P}_\omega X)^L$.

The following well-known definition was formulated first in [4].

Definition 56. A GSOS rule is an expression of the form

$$\frac{\left\{\mathtt{x}_{i_j} \xrightarrow{a_j} \mathtt{y}_j\right\}_{j=1..m} \quad \left\{\mathtt{x}_{i_k} \overset{b_k}{\not\longrightarrow}\right\}_{k=1..l}}{\mathtt{f}(\mathtt{x}_1, \ldots, \mathtt{x}_n) \xrightarrow{c} \mathtt{t}}$$

where:

- $\mathtt{f}$ is an operation in Σ of arity n,
- $m \in \mathbb{N}$ is the number of positive, and $l \in \mathbb{N}$ of negative premises in the rule,
- all $i_j, i_k \in \{1, \ldots, n\}$,
- $\mathtt{t}$ is a Σ-term over Ξ,
- all the $\mathtt{x}_i$ and $\mathtt{y}_j$ are distinct variables, and no other variables appear in $\mathtt{t}$,
- $a_j, b_k, c \in L$.

A rule as above is triggered by a tuple $(E_1, \ldots, E_n)$ of sets of *enabled labels*, where each $E_i \subseteq L$, if:

- $a_j \in E_{i_j}$ for all $j = 1..m$, and
- $b_k \notin E_{i_k}$ for all $k = 1..l$.

Definition 57. An image-finite GSOS specification is a set Λ of GSOS rules such that for each operation name $\mathtt{f}$ in Σ, each $c \in L$, and each tuple $\vec{E} = (E_1, \ldots, E_n)$ of subsets of L, there are only finitely many rules for $\mathtt{f}$ in Λ with c as the conclusion label, that are triggered by $\vec{E}$.

The LTS inferred from a GSOS specification is defined as expected, with negative premises $\mathtt{x}_i \overset{b_{ik}}{\not\longrightarrow}$ satisfied by the lack of a corresponding transition. The finiteness condition in Definition 57 ensures that the inferred LTS is image-finite.

Proposition 58. *Every GSOS specification gives rise to a GSOS law for $BX = (\mathcal{P}_\omega X)^L$, and every such law arises from a GSOS specification in this way.*

Proof. This result, especially the second part of it, is much more delicate to prove than the related Propositions 39 and 54. Although the result was first stated in [54,52], the first complete proof was given in [2]. □

Corollary 59. *Observational equivalence (i.e., bisimilarity) on the LTS inferred from a GSOS specification is a congruence.* □

CoGSOS laws for LTSs have also been given a characterization in [54], in terms of the so-called safe ntree specifications.

Definition 60. A safe ntree rule is an expression of the form

$$\frac{\left\{\mathtt{z}_i \xrightarrow{a_i} \mathtt{y}_i\right\}_{i\in I} \quad \left\{\mathtt{w}_j \overset{b_j}{\not\longrightarrow}\right\}_{j\in J}}{\mathtt{f}(\mathtt{x}_1, \ldots, \mathtt{x}_n) \xrightarrow{c} \mathtt{t}}$$

where:

- $\mathtt{f}$ is an operation in Σ of arity n,
- I and J are countable, possibly infinite index sets,
- the $\mathtt{x}_k$, $\mathtt{y}_i$, $\mathtt{z}_i$ and $\mathtt{w}_j$ are variables, the $\mathtt{x}_k$ and $\mathtt{y}_i$ are all distinct and they are the only variables that occur in the rule (in particular, each $\mathtt{z}_i$ and $\mathtt{w}_j$ is an occurrence of some $\mathtt{x}_k$ or $\mathtt{y}_i$),
- the *dependency graph* of premise variables (where positive premises are seen as directed edges) is well founded, i.e., it does not contain cycles or infinite backward chains,

- $\mathtt{t}$ is either a variable or a term built of a single operator from Σ and variables,
- $a_i, b_j, c \in L$.

See [54] for further details.

Proposition 61. *Every safe ntree specification gives rise to a coGSOS law for $BX = (\mathcal{P}_\omega X)^L$.*

It is not known whether this characterization is complete, i.e., whether every coGSOS law arises from a safe ntree specification.

7.3. Probabilistic systems

Recall from Example 6 that (reactive) probabilistic transition systems (PTSs) are coalgebras for the functor $BX = (\mathcal{D}_\omega X + 1)^L$. GSOS laws for this functor were characterized as *probabilistic GSOS* (PGSOS) specifications in [2].

Definition 62. A PGSOS rule is an expression of the form

$$\frac{\left\{\mathtt{x}_i \xrightarrow{\ a\ }\right\}_{i=1..n,\ a\in R_i} \quad \left\{\mathtt{x}_i \not\xrightarrow{\ a\ }\right\}_{i=1..n,\ a\in P_i} \quad \left\{\mathtt{x}_{i_j} \xrightarrow{\ b_j[u_j]\ \triangleright} \mathtt{y}_j\right\}_{j=1..m}}{\mathtt{f}(\mathtt{x}_1,\ldots,\mathtt{x}_n) \xrightarrow{\ c[w\cdot u_1\cdot\ldots\cdot u_m]\ \triangleright} \mathtt{t}}$$

where:

- $\mathtt{f}$ is an operation in Σ of arity n,
- $m \in \mathbb{N}$, and all $i_j \in \{1,\ldots,n\}$,
- $P_i, R_i \subseteq L$ with $P_i \cap R_i = \emptyset$, are sets of prohibited and requested labels for each $i = 1..n$ respectively,
- $\mathtt{t}$ is a Σ-term over Ξ,
- all the $\mathtt{x}_i$ and $\mathtt{y}_j$ are distinct variables in Ξ, and no other variables appear in $\mathtt{t}$,
- $c \in L$,
- $u_1,\ldots,u_m$ are distinct *probability variables* taken from some fixed set,
- $w \in (0, 1]$ is the *weight* of the rule.

A rule as above is triggered by a tuple of sets of enabled labels $(E_1,\ldots,E_n)$, where each $E_i \subseteq L$, if $R_i \subseteq E_i$ and $P_i \cap E_i = \emptyset$ for all $i = 1..n$.

Definition 63. A PGSOS specification is a set Λ of PGSOS rules such that for each operation name $\mathtt{f}$ in Σ, each $c \in L$, and each tuple $\vec{E} = (E_1,\ldots,E_n)$ of subsets of L, there are only finitely many rules for $\mathtt{f}$ in Λ with c as the conclusion label, that are triggered by $\vec{E}$; moreover, if there are any such rules, their weights must add up to 1.

As hinted by a different shape of some arrows in the definition of a PGSOS rule, a PTS is inferred from a PGSOS specification in a way different from simple proof-based definitions we have seen so far. The PTS structure on closed Σ-terms is defined by structural induction on sources of transitions as follows: for a term $s = \mathtt{f}(s_1,\ldots,s_n)$, calculate the sets E_i of enabled labels, for $i = 1..n$. For each rule R for $\mathtt{f}$ triggered by $(E_1,\ldots,E_n)$, calculate its contributions to the inferred probabilistic transition system as follows: for each tuple of processes $(t_j)_{j=1..m}$, let $v_j = \mu(s_{i_j} \xrightarrow{\ b_j\ } t_j)$ and define the contribution of R to the transition

$$s \xrightarrow{\ c\ } t[x_i \mapsto s_i,\ y_j \mapsto t_j]$$

to be the product $w \cdot v_1 \cdot \ldots \cdot v_j$. The probability of a transition from s is then defined as the sum of contributions of all rules for $\mathtt{f}$ and c triggered by $(E_1,\ldots,E_n)$. The conditions on a PGSOS specification ensure that the outgoing probabilities from any given process and any label add up to 0 or 1.

Proposition 64. *Every PGSOS specification gives rise to a GSOS law for $BX = (\mathcal{D}_\omega X + 1)^L$, and every such law arises from a PGSOS specification in this way.*

Proof. See [2]. □

Corollary 65. *Observational equivalence (i.e., probabilistic bisimilarity) on the PTS inferred from a GSOS specification is a congruence.* □

In [2], GSOS laws for the more complex *Segala systems* are also studied, where probabilities are combined with nondeterminism in a two-layered behavior. A rather complex rule format is defined there, called Segala-GSOS, and it is proved that every Segala-GSOS specification induces a GSOS law for the Segala behavior functor. It is not known whether every such GSOS law arises in this way.

7.4. Other systems

In [31], a rule-based presentation of GSOS laws for *stochastic systems* was given; these are similar to reactive probabilistic transition systems, but without the condition that rates of outgoing transitions add up to 1. Consequently, stochastic GSOS is rather similar to PGSOS of Section 7.3.

In [29], a more general class of *weighted transition systems* was studied. There, transitions are labeled with weights taken from some commutative monoid $\mathbb{W}$, and observational equivalence adds together weights of transitions as needed. Labeled transition systems appear as a special case $\mathbb{W}$ is the two-element monoid of truth values and logical disjunction. Stochastic systems arise for $\mathbb{W} = \mathbb{R}_{\geq 0}$ the monoid of nonnegative real numbers with addition. A general rule-based presentation of GSOS laws for a "weighted" behavior functor was given in [29], parametrized by $\mathbb{W}$ and called $\mathbb{W}$-GSOS. Ordinary GSOS and stochastic GSOS are rediscovered from the general format. However, for arbitrary $\mathbb{W}$, it was only proved that $\mathbb{W}$-GSOS specifications induce GSOS laws for weighted behavior; it is not known whether every such GSOS law arises in this way.

In [23,24], Kick studied a certain basic type of timed systems as coalgebras for the so-called evolution comonad $E_{\mathcal{T}}$ on **Set**. He provided a complete rule-based characterization of distributive laws of free term monads over $E_{\mathcal{T}}$; this is the only such characterization known for a nontrivial type of behavior. However, the evolution comonad is rather basic, and to deal with practical examples one must combine it with other types of behavior, as argued in [25,26].

Some attention has been put to the study of SOS specifications for name-passing calculi such as the π-calculus [48]. Name-passing systems can be understood as coalgebras for endofunctors on categories of presheaves, or on the category **Nom** of nominal sets [11]. In this framework, additional complications arise from the fact that syntactic signature functors typically act on a different category than the behavior functors. To deal with this, a generalized treatment of distributive laws was developed in [14], and modified later in [12]. A rule-based presentation of generalized GSOS laws for a particular choice of behavior functor on **Nom** was given in [12,13].

8. Related work

8.1. Distributive laws and regular expressions

A neat example application of distributive laws is Jacobs's study [19] of deterministic automata and regular expressions. It is well known that deterministic automata are coalgebras (quite similar to Mealy machines from Example 4) and that the set of all languages carries a final coalgebra structure. In [19] a GSOS law of the syntax of regular expressions over the deterministic automata behavior is defined. Regular expressions carry an initial, and all languages a final bialgebra structure. Further, a completeness proof for Kozen's axiomatization of regular expressions is formulated in the language of bialgebras for the distributive law.

8.2. Structured coalgebras

Closely related to bialgebras are *structured coalgebras*, i.e., coalgebras for functors on categories $\mathbf{Alg}(\Gamma)$ for an equational specification Γ. By Proposition 12, bialgebras are coalgebras for a functor on a category of algebras. On the other hand, if the functor on $\mathbf{Alg}(\Gamma)$ is a lifting of some endofunctor on the underlying category $\mathcal{C}$, then its coalgebras are bialgebras for a distributive law of a monad over a comonad, by Proposition 49.

In [7], structured coalgebras were used to study labeled transition systems where both states and transition labels are models of some specifications. Also, lax coalgebras were studied there to relax the standard coalgebraic notion of transition system morphism. In the context of structured coalgebras, it is natural to study structural equations on terms of the language under definition. In [8], a bisimulation-like condition on such equations was defined that guarantees the congruence property of bisimilarity. Moreover, the notion of dynamic bisimilarity, where system states can be tested by putting them in syntactic contexts, was formalized in terms of structured coalgebras.

In some contexts (e.g. [5]), it is useful to consider structured coalgebras for functors on $\mathbf{Alg}(\Gamma)$ that do not lift any endofunctors on **Set**, and hence do not arise from distributive laws. This situation is more general, but considerably less structured than the bialgebraic framework.

8.3. Variety of system equivalences

The basic framework of distributive laws aims at proving congruence results for the canonical notion of observational equivalence, or coalgebraic bisimilarity, for each categorical notion of system behavior. This contrasts with the multitude of equivalences defined even for single kinds of systems (see e.g. [55]).

One idea to circumvent this problem is to study systems as coalgebras where the chosen equivalence becomes the canonical observational equivalence. For example, when labeled transition systems (LTSs) are understood as coalgebras in the category of semi-lattices, trace equivalence becomes the canonical notion. This general idea has been used on the coalgebraic level several times (see e.g. [16,18,21,42]). In [53], it was used in an example of an SOS specification for which trace equivalence is a congruence, with semantics defined by a GSOS law in the category of semi-lattices.

Another idea is to extend the framework of distributive laws to cope with multiple equivalences instead of the canonical one. In the approach of *logical distributive laws* [27,28,30], process equivalences are modeled via modal logics that characterize them, such as Hennessy–Milner logic and its fragments. An abstract treatment of logics is developed in the framework of *coalgebraic modal logic*, which is combined with the basic approach to distributive laws to provide an abstract understanding of SOS that behaves well with respect to various process equivalences.

A recent alternative approach is [39], where system equivalences are modeled via special "objects of observations" in the underlying category.

8.4. Categories of distributive laws

In this paper, distributive laws and the corresponding SOS specifications were studied as isolated objects. However, from the point of view of category theory, it is natural to speak of morphisms of distributive laws, and categories of them. Distributive law morphism should hopefully provide an abstract perspective on well-behaved translations of SOS specifications. Also, via standard categorical notions of limits and colimits, distributive laws can be combined to form more complex ones; this opens a possibility of an abstract framework for modular SOS development.

Basic notions of distributive law morphisms were studied at the abstract level in [43]. However, little has been done to understand them in terms of concrete rule formats, their translations and combinations, apart from a few examples provided in [56] and [26].

8.5. Beyond GSOS

In the world of classical SOS specifications of labeled transition systems [1], GSOS is far from the most general format that guarantees bisimilarity to be a congruence. One of the more general ones is the ntyft/ntyxt format, where arbitrary terms are allowed as sources of premises. The presence of negative premises at this level of generality makes it far from obvious whether an SOS specification meaningfully defines an LTS at all; however, if an LTS is defined, then bisimilarity on it is a congruence (see [1] for details).

In [50], a categorical approach was developed to the tyft/tyxt format, where negative premises are forbidden, but arbitrary terms are allowed as sources of premises. Using basic techniques of topos theory, a general definition of a tyft/tyxt specification is provided which is more concrete, but also considerably more complicated, than various definitions of distributive laws considered in this paper. It is then proved that any such specification defines a lifting of the syntactic monad of the language under definition, to a category of transition systems; this provides a link to the distributive law framework. The general approach is also instantiated in a topos of nominal sets, and a few examples of specifications for name-passing systems are seen as special cases.

8.6. Stream equations and productivity

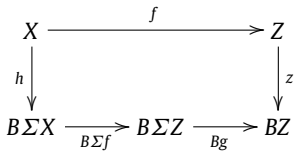
Specifications of infinite streams and operations on them, used in Section 3 as a simple framework to demonstrate the workings of distributive laws, is an active research area with many interesting recent developments. A coalgebraic perspective on the subject was explained in [46].

Traditionally, operations on streams are defined by systems of mutually recursive equations, but it is not difficult to translate such systems into sets of stream rules and back. The main concern in papers such as [10] is whether systems of equations uniquely define a collection of operations. General conditions on equation systems have been developed that guarantee that, based on notions such a productivity [49]. Although stream GSOS and stream coGSOS formats can easily be translated to conditions on stream equations that guarantee uniquely defined operations on streams, these do not seem as permissive as other conditions known in the community.

It not as yet clear whether the general framework of distributive laws of monads over comonads (Section 6.5) can bring anything new to the understanding of stream equation systems.

8.7. Generalized coinduction

It is well known that the basic coinduction principle (1) does not capture definitions of many useful operations on final coalgebras. Several authors enhanced the expressivity of coinduction with generalized "coiteration schemata" dually to various iteration schemata such as primitive recursion or course-of-value iteration. In [2,6,20,32], distributive laws were used to bring some order to this area and provide a uniform view on various extended coiteration schemata. For example, in [2,20], based on earlier ideas of [32], the principle of λ-*coiteration*, for a distributive law $\lambda : \Sigma B \Longrightarrow B\Sigma$ is defined as follows: for a $B\Sigma$-coalgebra h, its λ-coiterative extension is a map that makes the diagram:



commute, where z is the final B-coalgebra and g is the algebraic part of the final λ-bialgebra, defined as in (8). Under mild conditions, a unique such f exists. Similar definition principles are provided also for more complex types of distributive laws λ. The same idea is used to define the notion of λ-bisimulation, a coalgebraic generalization of the bisimulation up to principle.

Similar ideas were pursued in [6], where various coiteration principles are derived from a notion of generalized distributive law that connects three different endofunctors.

For recent developments in this area, see [36].

8.8. Microcosm principle

As explained in this paper, distributive laws can be used to define operations on the carriers of coalgebras. Sometimes, however, it might be interesting to study operations on coalgebras themselves. For example, the parallel composition operation of the process algebra CCS is usually seen as an operation on states of a labeled transition system, but it also makes sense to speak of whole transition systems running in parallel. As argued in [17], this two-layered view of parallel composition is an instance of a general phenomenon called the microcosm principle. It is argued that GSOS specifications define operations on labeled transition systems, which then yield the standard interpretation as operations on elements of the final coalgebra. This might be seen as a refinement of the distributive law formalization of classical GSOS. It is unclear whether it can be applied to GSOS specification formats for other behavior functors.

Acknowledgment

This work was supported by the EPSRC grant EP/F042337/1.

References

[1] L. Aceto, W.J. Fokkink, C. Verhoef, Structural operational semantics, in: J.A. Bergstra, A. Ponse, S. Smolka (Eds.), Handbook of Process Algebra, Elsevier, 2002, pp. 197–292.
[2] F. Bartels, On Generalised Coinduction and Probabilistic Specification Formats. Ph.D. Dissertation, CWI, Amsterdam, 2004.
[3] J.A. Bergstra, A. Ponse, S. Smolka, Handbook of Process Algebra, Elsevier, 2002.
[4] B. Bloom, S. Istrail, A. Meyer, Bisimulation can't be traced, Journal of the ACM 42 (1995) 232–268.
[5] F. Bonchi, U. Montanari, Coalgebraic symbolic semantics, in: Procs. CALCO'09, in: LNCS, vol. 5728, 2009, pp. 173–190.
[6] Daniela Cancila, Furio Honsell, Marina Lenisa, Generalized coiteration schemata, Electronic Notes in Theoretical Computer Science 82 (1) (2003).
[7] A. Corradini, M. Große-Rhode, R. Heckel, A coalgebraic presentation of structured transition systems, Theoretical Computer Science 260 (1–2) (2001) 27–55.
[8] A. Corradini, R. Heckel, U. Montanari, Compositional SOS and beyond: a coalgebraic view of open systems, Theoretical Computer Science 280 (2002) 163–192.
[9] E.P. de Vink, J.J.M.M. Rutten, Bisimulation for probabilistic transition systems: A coalgebraic approach, Theoretical Computer Science 221 (1–2) (1999) 271–293.
[10] J. Endrullis, C. Grabmayer, D. Hendriks, A. Isihara, J.W. Klop, Productivity of stream definitions, Theoretical Computer Science 411 (4–5) (2010) 765–782.
[11] M. Fiore, S. Staton, Comparing operational models of name-passing process calculi, Information and Computation 204 (2006) 524–560.
[12] M. Fiore, S. Staton, A congruence rule format for name-passing process calculi from mathematical structural operational semantics, in: Proc. LICS'06, IEEE Computer Society Press, 2006, pp. 49–58.
[13] M. Fiore, S. Staton, A congruence rule format for name-passing process calculi, Information and Computation 207 (2) (2009) 209–236.
[14] M.P. Fiore, D. Turi, Semantics of name and value passing, in: Proc. LICS'01, IEEE Computer Society Press, 2001, pp. 93–104.
[15] H. Hansen, B. Klin, Pointwise extensions of GSOS-defined operations, Mathematical Structures in Computer Science 21 (2) (2011) 321–361.
[16] I. Hasuo, B. Jacobs, A. Sokolova, Generic trace semantics via coinduction, Logical Methods in Computer Science 3 (4) (2007).
[17] I. Hasuo, B. Jacobs, A. Sokolova, The microcosm principle and concurrency in coalgebra, in: Procs. FoSSaCS'08, in: LNCS, vol. 4962, 2008, pp. 246–260.
[18] C. Hermida, B. Jacobs, Structural induction and coinduction in a fibrational setting, Information and Computation 145 (2) (1998) 107–152.
[19] B. Jacobs, A bialgebraic review of deterministic automata, regular expressions and languages, in: Essays dedicated to Joseph A. Goguen, in: LNCS, vol. 4060, 2006, pp. 375–404.
[20] B. Jacobs, Distributive laws for the coinductive solution of recursive equations, Information and Computation 204 (2006) 561–587.
[21] B. Jacobs, J. Hughes, Simulations in coalgebra, Electronic Notes in Theoretical Computer Science 82 (2003).
[22] P.T. Johnstone, Adjoint lifting theorems for categories of algebras, Bulletin of the London Mathematical Society 7 (1975) 294–297.
[23] M. Kick, Bialgebraic modelling of timed processes, in: Proc. ICALP'02, in: LNCS, vol. 2380, Springer, 2002, pp. 525–536.
[24] M. Kick, Rule formats for timed processes, in: Proc. CMCIM'02, in: ENTCS, vol. 68, Elsevier, 2002, pp. 12–31.
[25] M. Kick, J. Power, Modularity of behaviours for mathematical operational semantics, in: Procs. CMCS'04, in: ENTCS, vol. 106, Elsevier, 2004, pp. 185–200.
[26] M. Kick, J. Power, A. Simpson, Coalgebraic semantics for timed processes, Information and Computation 204 (2006) 588–609.
[27] B. Klin, Bialgebraic semantics and modal logic, in: Proc. LiCS'07, IEEE Computer Society Press, 2007, pp. 336–345.
[28] B. Klin, Bialgebraic methods and modal logic in structural operational semantics, Information and Computation 207 (2009) 237–257.
[29] B. Klin, Structural operational semantics for weighted transition systems, in: Semantics and Algebraic Specification, in: LNCS, vol. 5700, 2009, pp. 121–139.
[30] B. Klin, Structural operational semantics and modal logic, revisited, in: Proc. CMCS'10, in: ENTCS, vol. 264, Elsevier, 2010, pp. 155–175.

[31] B. Klin, V. Sassone, Structural operational semantic for stochastic systems, Proc. FOSSACS'08 vol. 4962 (2008) 428–442.
[32] M. Lenisa, From set-theoretic coinduction to coalgebraic coinduction: some results, some problems, in: ENTCS, vol. 19, 1999.
[33] M. Lenisa, J. Power, H. Watanabe, Distributivity for endofunctors, pointed and co-pointed endofunctors, monads and comonads, in: Proc. CMCS'00, in: ENTCS, vol. 33, Elsevier, 2000, pp. 230–260.
[34] M. Lenisa, J. Power, H. Watanabe, Category theory for operational semantics, Theoretical Computer Science 327 (1–2) (2004) 135–154.
[35] S. Mac Lane, Categories for the Working Mathematician, second edition, Springer, 1998.
[36] S. Milius, L.S. Moss, D. Schwencke, CIA structures and the semantics of recursion, in: Procs. FOSSACS'10, in: LNCS, vol. 6014, 2010, pp. 312–327.
[37] R. Milner, Communication and Concurrency, Prentice Hall, 1988.
[38] R. Milner, M. Tofte, The Definition of Standard ML, revised edition, MIT Press, 1997.
[39] Luís Monteiro, A coalgebraic characterization of behaviours in the linear time — branching time spectrum, in: Procs. WADT'08, in: Lecture Notes in Computer Science, vol. 5486, 2009, pp. 251–265.
[40] G.D. Plotkin, A structural approach to operational semantics. DAIMI Report FN-19, Computer Science Department, Aarhus University, 1981.
[41] G.D. Plotkin, A structural approach to operational semantics, Journal of Logic and Algebraic Programming 60–61 (2004) 17–139.
[42] J. Power, D. Turi, A coalgebraic foundation for linear time semantics, in: ENTCS, vol. 29, 1999.
[43] J. Power, H. Watanabe, Distributivity for a monad and a comonad, in: Procs. CMCS'99, in: ENTCS, vol. 19, Elsevier, 1999, p. 102.
[44] J.J.M.M. Rutten, D. Turi, Initial algebra and final coalgebra semantics for concurrency, in: J. de Bakker, et al. (Eds.), Proc. of the REX Workshop a Decade of Concurrency — Reflections and Perspectives, in: LNCS, vol. 803, Springer, 1994, pp. 530–582.
[45] J.J.M.M. Rutten, Universal coalgebra: a theory of systems, Theoretical Computer Science 249 (2000) 3–80.
[46] J.J.M.M. Rutten, Behavioural differential equations: a coinductive calculus of streams, automata and power series, Theoretical Computer Science 308 (1) (2003) 1–53.
[47] J.J.M.M. Rutten, Algebraic specification and coalgebraic synthesis of Mealy machines, in: Procs. FACS 2005, in: ENTCS, vol. 160, Elsevier Science Publishers, 2006, pp. 305–319.
[48] D. Sangiorgi, D. Walker, The π-Calculus: a Theory of Mobile Processes, Cambridge University Press, 2003.
[49] B.A. Sijtsma, On the productivity of recursive list definitions, ACM Transactions on Programming Languages and Systems 11 (4) (1989) 633–649.
[50] S. Staton, General structural operational semantics through categorical logic, in: Procs. LICS'08, 2008, pp. 166–177.
[51] Sam Staton, Relating coalgebraic notions of bisimulation, in: Procs. CALCO'09, in: LNCS, vol. 5728, 2009, pp. 191–205.
[52] D. Turi, Functorial operational semantics and its denotational dual, Ph.D. Thesis, Vrije Universiteit, Amsterdam, 1996.
[53] D. Turi, Categorical modeling of structural operational rules: case studies, in: Proc. CTCS'97, in: LNCS, vol. 1290, Springer, 1997, pp. 127–146.
[54] D. Turi, G.D. Plotkin, Towards a mathematical operational semantics, in: Proc. LICS'97, IEEE Computer Society Press, 1997, pp. 280–291.
[55] R.J. van Glabbeek, The linear time — branching time spectrum I, in: J.A. Bergstra, A. Ponse, S. Smolka (Eds.), Handbook of Process Algebra, Elsevier, 1999, pp. 3–99.
[56] H. Watanabe, Well-behaved translations between structural operational semantics, Electronic Notes in Theoretical Computer Science 65 (2002).