Bialgebraic Methods and Modal Logic in Structural Operational Semantics

Bartek Klin [1]

University of Edinburgh, Warsaw University

Abstract

Bialgebraic semantics, invented a decade ago by Turi and Plotkin, is an approach to formal reasoning about well-behaved structural operational semantics (SOS). An extension of algebraic and coalgebraic methods, it abstracts from concrete notions of syntax and system behaviour, thus treating various kinds of operational descriptions in a uniform fashion.

In this paper, bialgebraic semantics is combined with a coalgebraic approach to modal logic in a novel, general approach to proving the compositionality of process equivalences for languages defined by structural operational semantics. To prove compositionality, one provides a notion of behaviour for logical formulas, and defines an SOS-like specification of modal operators which reflects the original SOS specification of the language. This approach can be used to define SOS congruence formats as well as to prove compositionality for specific languages and equivalences.

Key words: structural operational semantics, coalgebra, bialgebra, modal logic, congruence format

1 Introduction

Structural Operational Semantics (SOS) [51,1] is one of the most successful frameworks for the formal description of programming languages and process calculi. There, the behaviour of programs or processes is described by means of transition relations, also called *labeled transition systems* (LTSs), induced by inference rules following the syntactic structure of processes. For example,

Email address: `bklin@inf.ed.ac.uk` (Bartek Klin).
[1] Supported by EPSRC grant EP/D039045/1.

the rules:

$$\frac{\mathtt{x} \xrightarrow{a} \mathtt{x}'}{\mathtt{x}||\mathtt{y} \xrightarrow{a} \mathtt{x}'||\mathtt{y}} \qquad \frac{\mathtt{y} \xrightarrow{a} \mathtt{y}'}{\mathtt{x}||\mathtt{y} \xrightarrow{a} \mathtt{x}||\mathtt{y}'} \tag{1}$$

define the behaviour of a binary parallel composition operator $||$ without communication. In particular, the rule on the left says that if a process can do a transition labelled with a, then the same process put in parallel with any other process can do a similar transition. One could also enrich states and/or transitions in SOS specifications with environments, stores, probabilities, time durations etc., to induce other, more sophisticated kinds of transition systems. The intuitive appeal of SOS and, importantly, its inherent support for modeling nondeterministic behaviour, makes it a natural framework for the formal description of process algebras (see [8] for many examples).

For reasoning about processes a suitable notion of *process equivalence* is needed. Various equivalences on LTSs have been proposed (see [22] for a survey). Bisimilarity is the most widely studied, but other equivalences such as trace equivalence or testing equivalence have also been considered. Several equivalences have also been proposed for probabilistic, timed and other kinds of transition systems, including their respective notions of bisimilarity.

To support inductive reasoning, it is important for the chosen process equivalence to be *compositional*; indeed, it is useful to know that if a part of a process is replaced by an equivalent part then the resulting process will be equivalent to the original one. Compositionality proofs for specific languages can be quite lengthy, therefore in the literature many *congruence formats* have been proposed. Such a format is a syntactic restriction on SOS specifications that guarantees a specific equivalence to be compositional on the induced transition system. The most popular format is GSOS [10], which guarantees the compositionality of bisimilarity, but formats for other equivalences and/or kinds of transition systems have also been studied (see [1,23]).

The task of finding a reasonably permissive congruence format for a given equivalence is usually quite demanding, therefore it would be desirable to have a general framework for the derivation of formats as well as for proving compositionality for specific languages. To be sufficiently general, such a framework should be parametrized by the process equivalence and by the kind of transition system. It is the purpose of this paper to provide such a framework.

Our approach is based on the categorical framework of *bialgebraic semantics* [57], where process syntax is modeled via algebras, and transition systems are viewed as coalgebras. For example, LTSs are coalgebras for the functor $(\mathcal{P}-)^A$ on the category **Set** of sets and functions, where $\mathcal{P}$ is the powerset functor and A a set of labels, and other kinds of transition systems are coalgebras for other functors, called *behaviour* functors in this context. Coalgebras also provide a general and abstract notion of bisimilarity (for more information on the coalgebraic theory of systems, see [53]). As it turns out, SOS specifications in the GSOS format are essentially *distributive laws* of syntax functors over $(\mathcal{P}-)^A$. Moreover, the process of inducing an LTS with a syntactic structure on processes from SOS rules is a special case of an abstract construction, where distributive laws of syntax over behaviour induce *bialgebras*, i.e., coalgebras with algebraic structures on their carriers. Also the fact that GSOS is a congruence format for bisimilarity can be proved at the level of distributive laws. This makes bialgebraic semantics a general framework for deriving congruence formats for bisimilarities, parametrized by the kind of transition systems; it was used to this purpose in [7,16,30] for probabilistic, timed and name-passing systems. In this paper, the framework is further parametrized by the notion of process equivalence.

Typically, process equivalences are characterized by *modal logics*. For example, two processes in an LTS are bisimilar if and only if they satisfy the same formulas in Hennessy-Milner logic [24], and fragments of that logic characterize other interesting equivalences on LTSs. Several attempts have been made to generalize such logics to coalgebras of arbitrary type [48,40,49,54]. Recently [38], based on earlier insights of [11,12,41,50], we have proposed a categorical generalization of modal logics for coalgebras in arbitrary categories. There, the syntax of a logic is modeled via algebras for an endofunctor, and its semantics via a suitable natural transformation connecting the logic syntax with the process behaviour.

The main contribution of this paper is a combination of the coalgebraic perspective on modal logic taken in [38] with the bialgebraic approach to SOS from [57]. Roughly speaking, to merge a logic and its semantics with a distributive law representing an SOS specification, one should provide a suitable notion of behaviour for the logic, and define a "dual", logical distributive law, where formulas play the role of processes, in a way that reflects the SOS specification. One might think of the logical behaviour as a way to decompose logical formulas over the syntax of processes. Our main result says that if such a logical distributive law exists, then the equivalence characterized by the logic is compositional on the transition system induced by the SOS specification.

For some kinds of logical behaviours, logical distributive laws can be presented as SOS-like inference rules where formulas act for processes, logical operators (modalities) for syntactic constructs, and logical inference operators for transitions. For example, rules:

$$\frac{\phi \dashv \psi || \sigma}{\langle a \rangle \phi \dashv \langle a \rangle \psi || \sigma} \qquad \frac{\phi \dashv \psi || \sigma}{\langle a \rangle \phi \dashv \psi || \langle a \rangle \sigma} \tag{2}$$

are used to define a logical distributive law reflecting (1). In particular, the rule on the left says that if a formula ϕ holds for every process of the form

$x||y$ such that ψ holds for x and σ holds for y, then the formula $\langle a\rangle\phi$ holds for every process of the form $z||w$ such that $\langle a\rangle\psi$ holds for z and σ holds for w. Since $\langle a\rangle\phi$ means that a process can do an a-transition to a process for which ϕ holds, this corresponds to the left rule in (1).

The framework proposed here can be seen as a very general "meta-congruence format", parametrized both by the notion of process equivalence and by the kind of transition system. It can be used directly to prove compositionality for specific languages and equivalences. Obviously it is hard to expect that such a general approach would be as easy to use as syntactic congruence formats designed for specific equivalences, and indeed finding the right logical distributive law and presenting it in a readable form is not always easy. However, our framework can also be used to derive specialized formats by proving that suitable distributive laws exist for a whole class of SOS specifications. The direct application to specific languages can be then left to problematic cases that do not fit in any known format.

The structure of the paper is as follows. The basics of classical SOS and congruence formats are presented in §2. In §3 the bialgebraic approach of [57] is explained on a series of very simple examples. A brief description of our approach to coalgebraic modal logic [38] follows in §4. In §5, the main technical result of the paper is obtained by merging the two approaches, and it is illustrated in §6 on some simple examples. Finally, §7 sketches some related and future work. Some familiarity with basic category theory is expected; [3,44] are good references.

The present paper is a full version of extended abstracts [36] and [37], with more detailed explanations provided and with more examples, including the substantial example of de Simone format in §6.3.

Acknowledgments. The author is grateful to Alexander Kurz and Gordon Plotkin for fruitful discussions, and to anonymous referees for several useful suggestions for improvement.

2 Structural Operational Semantics

We begin by recalling the classical framework of SOS as presented in [1]. A *labelled transition system* (LTS) $(X, A, \longrightarrow)$ is a set $X \ni x, y, \ldots$ of *processes*, a set $A \ni a, b, \ldots$ of *labels*, and a *transition relation* $\longrightarrow \subseteq X \times A \times X$, typically written $x \xrightarrow{a} y$ for $(x, a, y) \in \longrightarrow$; y is then an *a-successor*, or shortly a *successor* of x. An LTS is *image finite* if each process has only finitely many successors for each label, and it is *finitely branching* if each process has only finitely many successors altogether. One writes $x \not\xrightarrow{a}$ to say that x has no

a-successors, and $x \not\longrightarrow$ means that x has no successors at all.

Various equivalences are defined on processes in an LTS, and *bisimilarity* [45] is the most widely studied. In an LTS $(X, A, \longrightarrow)$, a relation $R \subseteq X \times X$ is a *bisimulation* if xRy implies:

- $\forall x \xrightarrow{a} x'.\ \exists y \xrightarrow{a} y'.\ x'Ry'$, and
- $\forall y \xrightarrow{a} y'.\ \exists x \xrightarrow{a} x'.\ x'Ry'$,

and processes $x, y \in X$ are *bisimilar* if there exists a bisimulation that relates them. On image finite LTSs, bisimilarity is characterized with finitary *Hennessy-Milner logic* (HML) [24], with syntax:

$$\phi ::= \top \mid \neg\phi \mid \phi \wedge \phi \mid \langle a\rangle\phi \tag{3}$$

where $a \in A$, and with semantics defined on a given LTS by:

$$\begin{aligned}
&x \models \top && \text{always}\\
&x \models \neg\phi &\iff& x \not\models \phi\\
&x \models \phi \wedge \psi &\iff& x \models \phi \text{ and } x \models \psi\\
&x \models \langle a\rangle\phi &\iff& \exists y \in X.\ x \xrightarrow{a} y,\ y \models \phi.
\end{aligned}$$

Indeed, processes are bisimilar if and only if they are logically equivalent, i.e., if they satisfy exactly the same formulas of the finitary HML. Various fragments of the logic have also been considered; see [22] for a survey. For example, the logic restricted to the syntax:

$$\phi ::= \top \mid \langle a\rangle\phi$$

characterizes *trace equivalence* on arbitrary LTSs, and the same fragment extended with a constant $\varnothing$:

$$\phi ::= \top \mid \varnothing \mid \langle a\rangle\phi \tag{4}$$

with semantics:

$$x \models \varnothing \iff x \not\longrightarrow$$

characterizes *completed trace equivalence* [22].

In the context of SOS, processes in LTSs usually are closed terms over some algebraic signature. A *signature* is a set $\Sigma \ni \mathtt{f}, \mathtt{g}, \ldots$ of *operation symbols* together with an *arity* function $ar : \Sigma \to \mathbb{N}$. A signature (Σ, ar) is usually denoted by Σ. Transition relations on Σ-terms are induced from sets of inference rules. Assuming a fixed set $\Xi \ni \mathtt{x}, \mathtt{y}, \ldots$ of variables, a *positive literal* over Σ is an expression of the form $\mathtt{t} \xrightarrow{a} \mathtt{s}$, where $\mathtt{t}$ and $\mathtt{s}$ are terms over Σ with variables from Ξ. Similarly, a *negative literal* is an expression of the form

$\mathtt{t} \overset{a}{\not\longrightarrow}$. An *inference rule* over Σ is an expression $\frac{H}{c}$, where H is a set of (positive or negative) literals, called *premises*, and c is a positive literal called the *conclusion*. A set of inference rules is called a *transition system specification* (TSS). An LTS *satisfies* a TSS R if it respects all inference rules in R in the obvious sense, and if the least LTS satisfying R exists then it is called the LTS *induced* by R.

Considered in this generality, inference rules do not guarantee the compositionality of any nontrivial process equivalence on the LTSs they induce. Indeed, it is not even clear that they induce any LTS. For these reasons, various restricted formats of SOS specifications have been proposed that guarantee these and other desirable properties. The most widely studied format is that of GSOS [10], where only rules of the following form are allowed:

$$\frac{\{\mathtt{x}_i \xrightarrow{a_{ij}} \mathtt{y}_{ij}\}^{1\le i\le n}_{1\le j\le m_i} \quad \{\mathtt{x}_i \overset{b_{ik}}{\not\longrightarrow}\}^{1\le i\le n}_{1\le k\le l_i}}{\mathtt{f}(\mathtt{x}_1,\dots,\mathtt{x}_n) \xrightarrow{c} \mathtt{t}} \tag{5}$$

where $\mathtt{f} \in \Sigma$, $n = ar(\mathtt{f})$, $m_i, l_i \in \mathbb{N}$, $a_{ij}, b_{ik}, c \in A$, $\mathtt{x}_i$ and $\mathtt{y}_{ij}$ are all distinct (i.e., $\mathtt{x}_i \ne \mathtt{x}_{i'}$ for $i \ne i'$, $\mathtt{y}_{ij} \ne \mathtt{y}_{i'j'}$ for $i \ne i'$ or $j = j'$, and $\mathtt{x}_i \ne \mathtt{y}_{ij}$ for all i, j), and no other variables occur in $\mathtt{t}$. A GSOS specification is *image finite* if it contains only finitely many rules for each $\mathtt{f} \in \Sigma$ and $c \in A$. GSOS specifications induce LTSs in an obvious way, as transitions for composite terms are fully determined by transitions for their subterms. Moreover, bisimilarity is guaranteed to be a congruence on the induced LTS. Also, LTSs induced by image finite GSOS specifications are image finite.

A well-known restriction of GSOS is de Simone format [55], where only rules of the following form are allowed:

$$\frac{\{\mathtt{x}_i \xrightarrow{a_i} \mathtt{y}_i\}_{i\in I}}{\mathtt{f}(\mathtt{x}_1,\dots,\mathtt{x}_n) \xrightarrow{c} \mathtt{t}} \tag{6}$$

where $I \subseteq \{1,\dots,n\}$, and $\mathtt{y}_i$ for $i \in I$ and $\mathtt{x}_j$ for $j \notin I$ are the only variables that occur in $\mathtt{t}$, with no variable occurring more than once in $\mathtt{t}$. This format guarantees also trace equivalence to be a congruence.

On the other hand, an extension of GSOS is the ntree format [19], where rules of the following form are allowed:

$$\frac{\{\mathtt{z}_i \xrightarrow{a_i} \mathtt{y}_i\}_{i\in I} \quad \{\mathtt{w}_j \overset{b_j}{\not\longrightarrow}\}_{j\in J}}{\mathtt{f}(\mathtt{x}_1,\dots,\mathtt{x}_n) \xrightarrow{c} \mathtt{t}} \tag{7}$$

where $\mathtt{x}_i$ and $\mathtt{y}_i$ are all distinct and are the only variables occurring in the rule (i.e., each $\mathtt{z}_i$ and $\mathtt{w}_j$ must be equal to some $\mathtt{x}_{i'}$ or $\mathtt{y}_{i'}$), I and J are countable sets, and the graph of positive premises is well-founded. Again, an ntree specification is *image finite* if it contains only finitely many rules for each $\mathtt{f} \in \Sigma$ and

$c \in A$. Ntree specifications do not necessarily induce an LTS in general, but they do if all rules are *safe*, meaning that $\mathtt{t}$ is either a variable or a term built of a single operation symbol and variables. Then bisimilarity is guaranteed to be a congruence on the induced LTS, which is image finite if the specification is image finite.

The safe ntree format is not an extension of GSOS, as it does not allow complex terms on the right sides of rule conclusions. Unlike GSOS however, it allows *lookahead* in premises, i.e., rules such as

$$\frac{\mathtt{x} \xrightarrow{a} \mathtt{y} \quad \mathtt{y} \xrightarrow{b} \mathtt{z}}{\mathtt{f}(\mathtt{x}) \xrightarrow{c} \mathtt{g}(z)}$$

are allowed.

Many other congruence formats have been studied in the literature. For example, interesting formats for various “decorated trace” equivalences were proposed in [9]. A considerably more complex format for completed trace equivalence was defined in [39,35]. For a detailed study of various congruence formats and their properties, see [1,23].

In some applications, it is useful to impose additional structure on labels in transition systems. For example, in probabilistic transition systems they are interpreted as probabilities [28]; in timed transition systems, as action durations [5]; in systems with name passing, they carry information about free and bound names [46]. These different kinds of systems are induced by various kinds of transition system specifications, similar to these described above. However, SOS congruence formats for equivalences on LTSs cannot be immediately reused for other kinds of systems. In the remainder of this paper, a general framework is described that avoids the need to rework the entire approach to SOS specifications and to congruence formats from scratch for each new kind of transition systems.

3 Bialgebraic semantics

In this section, the basic framework of bialgebraic semantics [57] is recalled and explained on a few simple examples, followed by a brief description of some results from the literature that have been obtained through the use of bialgebras.

The study of transition systems as coalgebras is motivated by the simple observation that LTSs are equivalent to functions

$$h : X \to (\mathcal{P}X)^A$$

where $\mathcal{P}$ is the powerset construction. Indeed, an LTS maps a process $x \in X$ and a label $a \in A$ to the set of all processes $y \in X$ such that $x \xrightarrow{a} y$. In the language of category theory, a function as above is called a *coalgebra* for the *functor* $(\mathcal{P}-)^A$ on the category **Set** of sets and functions.

In general, for any functor B on a category $\mathcal{C}$, a B-coalgebra is an object X (the *carrier*) together with a map $h : X \to BX$ (*the structure*). A coalgebra (X, h) is usually denoted simply by h. A *coalgebra morphism* from $h : X \to BX$ to $g : Y \to BY$ is a map $f : X \to Y$ such that the diagram

$$\begin{tikzcd}
BX \arrow[r, "Bf"] & BY \\
X \arrow[u, "h"] \arrow[r, "f"'] & Y \arrow[u, "g"']
\end{tikzcd}$$

commutes. If $B = (\mathcal{P}-)^A$ on **Set**, B-coalgebra morphisms are functional bisimulations on LTSs. In this context, B is called a *behaviour functor.*

As it happens, coalgebras for some other functors **Set** correspond to other well-known types of transition systems. For example:

- Coalgebras for $\mathcal{P}_\omega(A \times -)$, where $\mathcal{P}_\omega$ is the finite powerset functor, are finitely branching LTSs. Coalgebras for $(\mathcal{P}_\omega-)^A$ are image finite LTSs.
- Coalgebras for $\mathcal{D}(A\times -)+1$, where $\mathcal{D}$ is the probability distribution functor, are generative probabilistic transition systems.
- Coalgebras for $(S \times (1 + -))^S$, where S is a fixed set of memory states, are deterministic transition systems with state and termination.

Many other examples of systems modeled as coalgebras for functors on **Set** can be found in [53]. Coalgebras for functors on other categories have also been considered; for example, in [16], coalgebras for a certain functor on the category **Nom** of nominal sets and equivariant functions [21] are shown to correspond to a kind of labelled transition systems with name binding. The coalgebraic abstraction allows one to treat many different kinds of systems in a uniform manner. At the same time, many important notions used in reasoning about transition systems can be explained at the abstract, coalgebraic level. For example, a *coalgebraic bisimulation* [2,53] on a coalgebra $h : X \to BX$ is a relation R on X such that there exists a coalgebra structure $r : R \to BR$

for which the projections $\pi_1, \pi_2 : R \to X$ extend to a span of coalgebra morphisms:

$$\begin{tikzcd}
BX & BR \arrow[l, "B\pi_1"'] \arrow[r, "B\pi_2"] & BX \\
X \arrow[u, "h"'] & R \arrow[l, "\pi_1"] \arrow[r, "\pi_2"'] \arrow[u, "r"] & X \arrow[u, "h"']
\end{tikzcd}$$

For $B = (\mathcal{P}-)^A$, this notion specializes to ordinary bisimulation on an LTS; also for other choices of B it corresponds to canonical notions of bisimulations on the respective kinds of transition systems.

Another coalgebraic approach to bisimilarity for functors B on **Set** is via *behavioural equivalence*: two processes $x, y \in X$ in a coalgebra $h : X \to BX$ are equivalent if they are equated by some coalgebra morphism from h to some other B-coalgebra. For $B = (\mathcal{P}-)^A$ behavioural equivalence coincides with bisimilarity; similarly for other typical behaviour functors. In general, under the mild condition that B preserves weak pullbacks, behavioural equivalence coincides with the notion of bisimilarity based on coalgebraic bisimulations. However, some examples where that condition does not hold [40] suggest behavioural equivalence as the more basic notion of canonical process equivalence on coalgebras.

For more information on the coalgebraic approach to the theory of processes, see [53,27,25].

3.2 Terms as algebras

In SOS, processes are closed terms over some algebraic signature. It is standard to consider sets of such terms as algebras for certain functors on **Set**. For example, a signature described by the grammar

$$t ::= \texttt{nil} \mid a.t \mid t + t \mid t||t\,,$$

where a ranges over a fixed set A, corresponds to the functor

$$\Sigma X = 1 + A \times X + X \times X + X \times X$$

where 1 is a singleton set, $\times$ is cartesian product and $+$ is disjoint union. Note that an element of the set ΣX can be seen as a simple term over the above grammar, built of exactly one syntactic construct with variables from X (such terms will be called *flat terms* in the following). It turns out that algebras for the signature (in the usual sense of universal algebra) are maps

$$g : \Sigma X \to X$$

i.e., *algebras* for the functor Σ. This way, a simple syntax corresponds to a functor on **Set**. To model more advanced syntactic features such as variable binding, one needs to move to more complex categories, such as **Nom**.

Algebras provide a general, abstract perspective on the notion of congruence, in analogy to the coalgebraic treatment of behavioural equivalence. More specifically, an *algebra morphism* from $g : \Sigma X \to X$ to $h : \Sigma Y \to Y$ is a map $f : X \to Y$ such that the diagram

$$\begin{tikzcd}
\Sigma X \arrow[r, "\Sigma f"] \arrow[d, "g"'] & \Sigma Y \arrow[d, "h"] \\
X \arrow[r, "f"'] & Y
\end{tikzcd}$$

commutes. (The kernel of) an algebra morphism from $g : \Sigma X \to X$ is called a *congruence* on the algebra g. It is easy to see that for Σ on **Set** corresponding to an algebraic signature, this notion coincides with the notion of congruence known from universal algebra.

If a functor Σ corresponds to an algebraic signature then the set of terms over the signature and over a set X of variables is denoted $T_\Sigma X$, or TX if Σ is clear from context. In particular, $T0$ is the set of closed terms over Σ. This set admits an obvious and canonical algebra structure, denoted $a_\Sigma : \Sigma T0 \to T0$. This Σ-algebra is *initial*: for any other algebra $g : \Sigma X \to X$ there is a unique algebra morphism $g^\sharp : T0 \to X$ from a_Σ to g. Intuitively, $g^\sharp$ is defined by structural induction, where g defines the inductive step. The construction T extends to a functor, and it is called the monad freely generated by Σ. The notions of initial algebra and freely generated monad do not depend on Σ corresponding to an algebraic signature, and can be defined for many other functors: in general, $T_\Sigma X$ is the carrier of an initial $(X + \Sigma-)$-algebra, if the latter exists.

For more intuition about this categorical approach to induction, see e.g. [27].

3.3 SOS and distributive laws

SOS specifications induce LTSs with closed terms as processes. In other words, the set of processes is equipped with both a coalgebraic structure, which maps a process to a structure of its successors, and an algebraic structure, which describes how to obtain a process by combining other processes. Formally, induced LTSs are coalgebras $h : T0 \to BT0$ for a suitable behaviour B, and for T the monad freely generated by syntax Σ.

To model the process of inducing LTSs with syntax abstractly, a sufficiently abstract notion of structural operational description is needed. For a simple example, consider a standard set of operational inference rules for a toy language with synchronous product:

$$t ::= \texttt{nil} \mid a \mid t \otimes t$$

$$\frac{\mathtt{x} \xrightarrow{a} \mathtt{x}' \quad \mathtt{y} \xrightarrow{a} \mathtt{y}'}{\mathtt{x} \otimes \mathtt{y} \xrightarrow{a} \mathtt{x}' \otimes \mathtt{y}'} \qquad \frac{}{a \xrightarrow{a} \texttt{nil}} \tag{8}$$

where a ranges over a fixed set A of labels. The syntax of this language corresponds, as mentioned in §3.2, to the functor

$$\Sigma X = 1 + A + X \times X\,.$$

Rules (8) induce an image finite LTS labelled with A, i.e., a coalgebra for the functor

$$BX = (\mathcal{P}_\omega X)^A\,.$$

But how to model the rules on the abstract level? Informally, they define the behaviour (i.e., the set of successors) of a term built of a single syntactic construct and variables, based on some information about the behaviour of subterms represented by the variables. For example, given processes x, y from any set X, and sets of successors for x and for y, the first rule defines the set of successors for the process $x \otimes y$. Note that while successors of x and y are variables and therefore can be thought of as arbitrary elements of X, the derived successors of $x \otimes y$ are simple terms from ΣX. Formally, the first rule in (8) represents a function

$$\lambda_\otimes : BX \times BX \to B\Sigma X$$

defined by

$$\lambda_\otimes(\beta, \gamma)(b) = \{\, x \otimes y \in \Sigma X \mid\; x \in \beta(b) \wedge y \in \gamma(b)\,\}\,.$$

Similarly, the right rule represents a function $\lambda_A : A \to B\Sigma X$ defined by

$$\lambda_A(a)(b) = \begin{cases} \{\texttt{nil}\} & \text{if } a = b \\ \emptyset & \text{otherwise} \end{cases}$$

and even the lack of any rules for the construct $\texttt{nil}$ defines its behaviour: the process $\texttt{nil}$ has no successors. This can be viewed as a function $\lambda_{\texttt{nil}} : 1 \to B\Sigma X$:

$$\lambda_{\texttt{nil}}(\star)(b) = \emptyset\,.$$

The three functions can be combined into a function

$$\lambda : \Sigma BX \to B\Sigma X$$

defined by cases and corresponding to (8). Note that the structure of X and the nature of its elements are completely ignored in the definition of λ. Formally, λ is natural over X:

$$\lambda : \Sigma B \Longrightarrow B\Sigma\,. \tag{9}$$

A natural transformation like this is called a *distributive law* of Σ over B, and a first attempt to model structural operational rules would be to consider distributive laws of the syntax functor over the behaviour functor. We have just seen a reasonable example covered by this notion. It turns out that the process of inferring LTSs from SOS rules can be explained abstractly at the level of distributive laws. Indeed, the unique algebra morphism h_λ from the initial Σ-algebra as below:

$$\begin{tikzcd} T0 \arrow[d, dotted, "h_\lambda"'] & & \Sigma T0 \arrow[ll, "a_\Sigma"'] \arrow[d, "\Sigma h_\lambda"] \\ BT0 & B\Sigma T0 \arrow[l, "Ba_\Sigma"] & \Sigma BT0 \arrow[l, "\lambda_{T0}"] \end{tikzcd} \tag{10}$$

is an LTS of the required type. The pair (a_Σ, h_λ) is an (initial) λ-*bialgebra* [57], the central notion of bialgebraic semantics. This inductive definition of the coalgebraic part of the initial algebra corresponds to the inductive definition of an LTS from SOS rules such as in (8). For the transformation λ defined as above, the inductively defined h_λ is exactly the expected LTS induced by (8).

3.4 More distributive laws: towards abstract GSOS

The example of the previous section encourages one to model SOS specifications as distributive laws. However, there are many examples which do not fit into the simple framework described so far. Consider, for example, a simple language featuring parallel composition without communication:

$$t ::= \mathtt{nil} \mid a \mid t||t$$

$$\frac{\mathtt{x} \xrightarrow{a} \mathtt{x}'}{\mathtt{x}||\mathtt{y} \xrightarrow{a} \mathtt{x}'||\mathtt{y}} \qquad \frac{\mathtt{y} \xrightarrow{a} \mathtt{y}'}{\mathtt{x}||\mathtt{y} \xrightarrow{a} \mathtt{x}||\mathtt{y}'} \qquad \frac{}{a \xrightarrow{a} \mathtt{nil}} \tag{11}$$

where a ranges over a fixed set A of labels. As before, the syntax and behaviour of the language is modeled by functors

$$\Sigma X = 1 + A + X \times X, \qquad BX = (\mathcal{P}_\omega -)^A.$$

However, it turns out that the two rules defining the parallel composition operator $||$, do not represent a function of the type $\lambda_{||} : BX \times BX \Longrightarrow B\Sigma X$. The reason for this is that in both rules, variables from the left sides of the conclusions appear on the right sides. This means that successors of composite terms cannot be built solely from the successors of their subterms; indeed, information about the subterms themselves is needed as well. Accordingly, the two problematic rules do represent, for any set X, a function of the form:

$$\lambda_{||} : (X \times BX) \times (X \times BX) \to B\Sigma X$$

defined by

$$\lambda_{||}\left((x, \beta), (y, \gamma)\right)(b) = \{\, x'||y \mid x' \in \beta(b) \,\} \cup \{\, x||y' \mid y' \in \gamma(b) \,\}\,.$$

Combined with λ_A and $\lambda_{\mathtt{nil}}$ defined as above, this gives a natural transformation

$$\lambda : \Sigma(\mathrm{Id} \times B) \Longrightarrow B\Sigma. \tag{12}$$

This is a distributive law slightly more general than considered before, and it induces an initial bialgebra with a diagram little more complicated than (10).

On the other hand, consider a language with nondeterministic choice, defined by:

$$t ::= \mathtt{nil} \mid a \mid t{+}t$$

$$\frac{\mathtt{x} \xrightarrow{a} \mathtt{x}'}{\mathtt{x} + \mathtt{y} \xrightarrow{a} \mathtt{x}'} \qquad \frac{\mathtt{y} \xrightarrow{a} \mathtt{y}'}{\mathtt{x} + \mathtt{y} \xrightarrow{a} \mathtt{y}'} \qquad \frac{}{a \xrightarrow{a} \mathtt{nil}} \tag{13}$$

where a ranges over a fixed set A of labels. Again, the syntax and behaviour of the language corresponds to functors

$$\Sigma X = 1 + A + X \times X, \qquad BX = (\mathcal{P}_\omega -)^A.$$

Here, successors of composite terms do not use their subterms. However, in the rules for $+$ the successors are variables rather than flat terms. This means that these rules do not represent a function of the type $\lambda_+ : BX \times BX \to B\Sigma X$ either. However, for any X they represent a function

$$\lambda_+ : BX \times BX \to BX$$

defined by

$$\lambda_+(\beta, \gamma)(b) = \beta(b) \cup \gamma(b)$$

and this, combined with λ_A and $\lambda_{\mathtt{nil}}$ as before, yields a natural transformation

$$\lambda : \Sigma B \Longrightarrow B(\mathrm{Id} + \Sigma). \tag{14}$$

Again, this type of distributive law induces initial bialgebras similarly as in (10).

A common generalization of (12) and (14) is

$$\lambda : \Sigma(\mathrm{Id} \times B) \Longrightarrow B(\mathrm{Id} + \Sigma). \tag{15}$$

Many interesting SOS specifications represent distributive laws of this type for $B = (\mathcal{P}_\omega -)^A$ and for Σ corresponding to algebraic signatures. For example, consider the following subset of CCS:

$$t ::= \texttt{nil} \mid a.t \mid t + t \mid t||t$$

$$\frac{}{a.\mathtt{x} \xrightarrow{a} \mathtt{x}} \qquad \frac{\mathtt{x} \xrightarrow{a} \mathtt{x}'}{\mathtt{x}+\mathtt{y} \xrightarrow{a} \mathtt{x}'} \qquad \frac{\mathtt{y} \xrightarrow{a} \mathtt{y}'}{\mathtt{x}+\mathtt{y} \xrightarrow{a} \mathtt{y}'} \tag{16}$$

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

where it is assumed that $A = A_0 \cup \{\, \bar{a} \mid \, a \in A \,\} \cup \{\tau\}$, and $\bar{\bar{a}}$ denotes a. It is straightforward to see how the above rules represents such a law for

$$\Sigma X = 1 + A \times X + X \times X + X \times X.$$

In particular, the component function $\lambda_. : A \times X \times BX \to B(X + \Sigma X)$ for the prefixing operator is defined by:

$$\lambda_.(a, x, \gamma)(b) = \begin{cases} \{\iota_1(x)\} & \text{if } a = b \\ \emptyset & \text{otherwise} \end{cases}$$

where $\iota_1 : X \to X + \Sigma X$ is the coproduct injection.

However, some useful specifications do not conform to the type of (15). Consider a simple language with sequential composition and binary Kleene star, defined by:

$$t ::= \texttt{nil} \mid a \mid t;t \mid t * t$$

$$\frac{}{a \xrightarrow{a} \texttt{nil}} \qquad \frac{\mathtt{x} \xrightarrow{a} \mathtt{x}'}{\mathtt{x};\mathtt{y} \xrightarrow{a} \mathtt{x}';\mathtt{y}} \qquad \frac{\{\mathtt{x} \not\xrightarrow{b}\}_{b \in A} \quad \mathtt{y} \xrightarrow{a} \mathtt{y}'}{\mathtt{x};\mathtt{y} \xrightarrow{a} \mathtt{y}'} \tag{17}$$

$$\frac{\mathtt{x} \xrightarrow{a} \mathtt{x}'}{\mathtt{x} * \mathtt{y} \xrightarrow{a} \mathtt{x}';(\mathtt{x} * \mathtt{y})} \qquad \frac{\mathtt{y} \xrightarrow{a} \mathtt{y}'}{\mathtt{x} * \mathtt{y} \xrightarrow{a} \mathtt{y}'}$$

Here, the term on the right side of the conclusion of the first rule for $*$ is not flat, therefore the codomain of the corresponding distributive law cannot be $B\Sigma$ or even $B(\mathrm{Id} + \Sigma)$. However, the above rules do define a distributive law of the type

$$\lambda : \Sigma(\mathrm{Id} \times B) \Longrightarrow BT_\Sigma \tag{18}$$

where

$$\Sigma X = 1 + A + X \times X + X \times X$$

and T_Σ is the free monad over Σ, i.e., the functor than maps a set X to the set of Σ-terms with variables from X.

Altogether, the component functions of λ represented by the rules (17) are, on a set X:

$$\begin{aligned}
\lambda_{\texttt{nil}}(\star)(b) &= \emptyset \\
\lambda_A(a)(b) &= \begin{cases} \{\texttt{nil}\} \text{ if } a = b \\ \quad \emptyset \quad\; \text{otherwise} \end{cases} \\
\lambda_{;}((x,\beta),(y,\gamma))(b) &= \begin{cases} \gamma(b) & \text{if } \forall c \in A.\beta(c) = \emptyset \\ \{\, x';y \mid x' \in \beta(b) \,\} & \text{otherwise} \end{cases} \\
\lambda_{*}((x,\beta),(y,\gamma))(b) &= \{\, x';(x * y) \mid x' \in \beta(b) \,\} \cup \gamma(b)
\end{aligned}$$

and these define a function natural in X.

Plenty of interesting SOS specifications represent distributive laws of the type (18) for $B = (\mathcal{P}_\omega -)^A$. In fact, as was observed in [57] and proved in detail in [7], such distributive laws correspond to image finite GSOS specifications (5). For this reason, the type (18) of distributive laws is called *abstract GSOS*.

Abstract GSOS is a generalization of (9) and (15). In [57], a dual generalization was also suggested, to distributive laws of the type

$$\lambda : \Sigma D_B \Longrightarrow B(\mathrm{Id} + \Sigma) \tag{19}$$

where D_B is the *cofree comonad* over B. Cofree comonad is the categorical notion dual to that of free monad: just as $T_\Sigma X$ is the carrier of an initial $(X + \Sigma-)$-algebra, $D_B X$ is the carrier of a final $(X \times B-)$-coalgebra. For example, if $B = (\mathcal{P}_\omega -)^A$ on **Set**, then $D_B X$ is the set of all finitely branching, but possibly infinite, synchronization trees edge-labeled with A and node-labeled with X, quotiented by strong bisimilarity. It turns out that just laws of this type for $B = (\mathcal{P}_\omega -)^A$ correspond to specifications in the image finite safe ntree format (7); therefore the type (19) of distributive law is called *abstract safe ntree*. Following this convention, we will call the type (9) *abstract toy SOS*.

All laws of the type (12), (14), (15), (18) or (19) induce initial bialgebras by constructions similar to (10). This follows from a more general construction from [57], for distributive laws of the free monad T_Σ over the cofree comonad

D_B (these are natural transformations $\lambda : T_\Sigma D_B \Longrightarrow D_B T_\Sigma$ subjects to additional axioms). It is also shown there that, provided that B preserves weak pullbacks, coalgebraic bisimilarity on the induced B-coalgebra is a congruence on the initial Σ-algebra; this specializes to the previously known facts that GSOS and safe ntree are congruence formats for bisimilarity. The advantage of the bialgebraic approach is that these abstract constructions and results apply also to other choices of B and Σ, and even to other underlying categories. Some applications of this are mentioned below.

3.5 Related work on bialgebraic semantics

Since [57], several studies and applications of the bialgebraic framework have been developed. For reference, we briefly list some of them before we proceed to the main contribution of this paper: a combination of the basic framework with a coalgebraic approach to modal logic.

In [57], natural transformations of the type (18) and (19) are considered as special cases of the more general notion of a distributive law of a monad over a comonad. In [42,43,52], various types of distributive laws are studied on the abstract, categorical level. In [7], different kinds of distributive laws are studied and related on the concrete example of LTSs; also a complete proof of one-to-one correspondence between abstract GSOS and concrete GSOS specifications is included there.

In [6,7], the abstract GSOS framework is applied to reactive probabilistic systems and probabilistic automata, represented as coalgebras for suitable functors. A congruence format for probabilistic bisimilarity is derived. In [29,30], the same framework is applied to processes with timed transitions. Congruence results regarding timed bisimilarity are proved, and a congruence format for the case of discrete time is derived. In [31,32], the combination of timing with action is studied more carefully, with insights on combining different behaviours to obtain a modular account of semantics.

In [34], abstract GSOS is studied in a CPO-enriched setting, where recursion is possible to express via straightforward fixpoint constructions. There, it is shown how to combine standard GSOS distributive laws with recursive equations to obtain other well-behaved distributive laws. Another bialgebraic approach to recursive equations is [26].

In [17,18], syntax with variable binding was modeled algebraically in a presheaf category, and the standard SOS description of the π-calculus was shown to fit in the abstract GSOS format there, although no actual format was proposed. Recently [16], such a format, a special case of abstract GSOS, has been proposed in the closely related setting of nominal sets [21], with congruence properties related to a version of open bisimilarity. Interestingly, in nominal settings the syntax and behaviour functors reside in different categories. The basic bialgebraic setting is suitably generalized to accommodate this.

In [39,35], abstract GSOS is interpreted in certain fibered categories. This allows one to derive congruence formats for process equivalences other than the canonical coalgebraic notion of bisimilarity. In particular, novel formats for completed trace and failures equivalences on LTSs were obtained. That work is closely related to the present paper, and indeed the following sections can be seen as a refinement of the approach described in [39,35,33].

One should also mention the approach of structured coalgebras (e.g. [15,13]), used for purposes similar to this work and related to the framework of bialgebraic semantics.

4 Coalgebraic modal logic

To study HML and other modal logics at the level of generality of distributive laws, we will use the recent approach of [38], inspired by earlier results of [41,50,54]. To gain momentum, we begin by considering the familiar setting of sets and functions. Normally, the semantics of a logic is some satisfaction relation $\models \subseteq X \times \Phi$ between the set Φ of tests (formulas) and the set X of tested entities (processes), or equivalently a function:

$$\models : X \times \Phi \to 2$$

(here and in the following, 2 denotes the two-element set $\{\mathtt{tt}, \mathtt{ff}\}$). Its two transposes:

$$[\![\_]\!] : \Phi \to 2^X \qquad [\![\_]\!]^\flat : X \to 2^\Phi \tag{20}$$

define the semantics of processes by sets of formulas that hold for them, and the semantics of formulas by sets of processes that satisfy them. In particular, two processes in X are logically equivalent if they are equated by $[\![\_]\!]^\flat$. This treatment is easily generalized to logics where another set is used for "truth values"; for example, in some probabilistic logics the continuous interval $[0,1]$ is used instead of 2.

More generally, assume a category $\mathcal{C}$ of structures of processes, and a category $\mathcal{D}$ of structures of logical formulas, connected by a contravariant adjunction $F \dashv G^{op} : \mathcal{C} \to \mathcal{D}^{op}$. This means that a bijection $\mathcal{C}(X, G\Phi) \cong \mathcal{D}(\Phi, FX)$ holds for any $X \in \mathcal{C}$, $\Phi \in \mathcal{D}$; slightly abusing the notation, we will denote both directions of this bijection by $-^\flat$. To avoid notational clutter, all *op*-notation for functors and natural transformations is omitted in the following; formally, we see F and G as contravariant functors between $\mathcal{C}$ and $\mathcal{D}$, and compose them with (covariant) functors on $\mathcal{C}$ or $\mathcal{D}$ in the obvious way. In all concrete examples considered in this paper, $\mathcal{C} = \mathcal{D} = \mathbf{Set}$ and $F = G = 2^{-}$; however, our abstract results hold for the general case as well.

Functors F and G provide the infrastructure for linking processes and formulas. Note that GF is a monad on $\mathcal{C}$; denote its unit by $\eta : \mathrm{Id} \Longrightarrow GF$. For any $f : \Phi \to FX$ in $\mathcal{D}$, one has $f^\flat = Gf \circ \eta_X$. Also FG is a monad on $\mathcal{D}$, with the unit denoted by $\epsilon : \mathrm{Id} \Longrightarrow FG$.

Assuming a functor B on $\mathcal{C}$, a (*coalgebraic polyadic modal*) *logic* for B-coalgebras is a functor L on $\mathcal{D}$ (the *syntax*) together with a *connection* between L and B, i.e., a natural transformation $\rho : LF \Longrightarrow FB$ (the *semantics*). Such a ρ determines the adjoint connection

$$\rho^\star = GL\epsilon \circ G\rho G \circ \eta_{BG} : BG \Longrightarrow GL; \tag{21}$$

it is not difficult to see that the correspondence between ρ and $\rho^\star$ is bijective.

If L has an initial algebra $a : L\Phi \to \Phi$, then for any coalgebra $h : X \to BX$ the interpretation $[\![\_]\!]_h : \Phi \to FX$ is defined as the unique algebra morphism:

$$\begin{tikzcd} \Phi \arrow[d, dotted, "{[\![\_]\!]_h}"'] & & L\Phi \arrow[ll, "a"'] \arrow[d, "{L[\![\_]\!]_h}"] \\ FX & FBX \arrow[l, "Fh"] & LFX, \arrow[l, "\rho_X"] \end{tikzcd} \tag{22}$$

and the transpose $[\![\_]\!]_h^\flat : X \to G\Phi$ represents the logical equivalence associated with (L, ρ).

Example 1 The logic for completed trace equivalence (4) on finitely branching LTSs, i.e., on B-coalgebras for $B = (\mathcal{P}_\omega -)^A$ on **Set**, is defined by syntax:

$$L\Phi = \{\top\} + \{\varnothing\} + A \times \Phi$$

on **Set**, with semantics $\rho_X : L2^X \to 2^{BX}$ defined by cases:

$$\begin{aligned} \rho_X(\top)(\beta) &= \mathtt{tt} \quad \text{always} \\ \rho_X(\varnothing)(\beta) &= \mathtt{tt} \iff \forall a \in A.\ \beta(a) = \emptyset \\ \rho_X(\langle a\rangle\phi)(\beta) &= \mathtt{tt} \iff \exists y \in \beta(a).\ \phi(y) = \mathtt{tt}. \end{aligned}$$

It is easy to see how L corresponds to the syntax of the logic (4) for completed trace equivalence and ρ to its semantics. Indeed, for any B-coalgebra h, the map $[\![\_]\!]_h$ defined by (22) is the usual semantics of the logic for completed traces, and the kernel of $[\![\_]\!]_h^\flat$ is completed trace equivalence on h. $\square$

Example 2 The finitary HML (3) on B-coalgebras for $B = (\mathcal{P}_\omega -)^A$ on **Set**

is defined by syntax:

$$L\Phi = A \times \coprod_{n\in\mathbb{N}} (2\times\Phi)^n$$

on **Set**, represented with the grammar:

$$\phi ::= \langle a\rangle \bigwedge_{j=1..n} \psi_j \qquad\qquad \psi ::= \phi \mid \neg\phi. \tag{23}$$

Its semantics $\rho : L2^- \Longrightarrow 2^{B-}$ is defined by:

$$\rho_X(\langle a\rangle(\psi_1\wedge\cdots\wedge\psi_n))(\beta) = \mathtt{tt} \iff$$
$$\exists y\in\beta(a).\forall i=1..n. \begin{cases} \psi_i = \phi_i \Rightarrow \phi_i(y) = \mathtt{tt} \\ \psi_i = \neg\phi_i \Rightarrow \phi_i(y) = \mathtt{ff} \end{cases}$$

Note how propositional operators $\wedge$ and $\neg$ are combined with the diamond modalities $\langle a\rangle$ of HML in a single layer of syntax L. This makes our presentation of HML a little more complicated than the classical (3); however, it allows for a formalization of its semantics as a transformation ρ of a simple type. Also note that the logic presented here is, formally speaking, a proper subset of HML: for example, the always true formula $\top$ is not present (although all formulas of the form $\langle a\rangle\top$ are, with $\top$ being, by convention, the empty conjunction). However, the fragment is expressive, i.e., it still characterizes bisimilarity. $\square$

5 Logical distributive laws

A logic (L,ρ) for B-coalgebras lifts B to an endofunctor B^ρ on the category $(\mathcal{D}\downarrow F)$, i.e, the slice category of the contravariant adjunction of F and G. Objects of $(\mathcal{D}\downarrow F)$ are triples (X,r,Φ) where $X\in\mathcal{C}$, $\Phi\in\mathcal{D}$ and $r:\Phi\to FX$ in $\mathcal{D}$, and a morphism $(f,g):(X,r,\Phi)\to(Y,s,\Psi)$ is a pair of maps $f:X\to Y$, $g:\Psi\to\Phi$ such that $Ff\circ s = r\circ g$. The functor B^ρ on $(\mathcal{D}\downarrow F)$ is defined by:

$$\begin{aligned} B^\rho(X,r,\Phi) &= (BX, \rho_X\circ Lr, L\Phi) \\ B^\rho(f,g) &= (Bf, Lg) \end{aligned}$$

and a B^ρ-coalgebra is a B-coalgebra together with an L-algebra interpreted in it according to ρ.

The above suggests that coalgebraic modal logic concerns coalgebras in the category $(\mathcal{D}\downarrow F)$, and to combine it with the bialgebraic approach to SOS one should interpret the latter in that category. To simplify the presentation, we do it first for abstract toy SOS (9), and then show without proof how the approach generalizes to abstract GSOS (18).

Assume that a syntax functor Σ on $\mathcal{C}$ is lifted to a functor Σ^ζ with a functor Γ on $\mathcal{D}$ and a transformation $\zeta : \Gamma F \Longrightarrow F\Sigma$, just as B can be lifted to B^ρ with L and ρ. The following is a characterization of distributive laws of Σ^ζ over B^ρ in terms of more elementary laws satisfying a coherence condition:

Proposition 1 *Every pair (λ, χ) of laws in $\mathcal{C}$ and $\mathcal{D}$:*

$$\lambda : \Sigma B \Longrightarrow B\Sigma \qquad \chi : L\Gamma \Longrightarrow \Gamma L$$

such that the hexagon

$$\begin{tikzcd}
L\Gamma F \arrow[r, Rightarrow, "L\zeta"] \arrow[d, Rightarrow, "\chi F"'] & LF\Sigma \arrow[r, Rightarrow, "\rho\Sigma"] & FB\Sigma \arrow[d, Rightarrow, "F\lambda"] \\
\Gamma LF \arrow[r, Rightarrow, "\Gamma\rho"'] & \Gamma FB \arrow[r, Rightarrow, "\zeta B"'] & F\Sigma B
\end{tikzcd} \tag{24}$$

commutes, gives rise to a distributive law of Σ^ζ over B^ρ.

Proof. Given λ and χ as above, define a distributive law $\kappa : \Sigma^\zeta B^\rho \Longrightarrow B^\rho \Sigma^\zeta$ by $\kappa_{(X,r,\Phi)} = (\lambda_X, \chi_\Phi)$ for any $(X, r, \Phi) \in (\mathcal{D} \downarrow F)$. This has the right type, since $\Sigma^\zeta B^\rho(X, r, \Phi) = (\Sigma BX, \zeta_{BX} \circ \Gamma\rho_X \circ \Gamma Lr, \Gamma L\Phi)$ and $B^\rho\Sigma^\zeta(X, r, \Phi) = (B\Sigma X, \rho_{\Sigma X} \circ L\zeta_X \circ L\Gamma r, L\Gamma\Phi)$. It is also a well-defined morphism in $(\mathcal{D} \downarrow F)$, as the following diagram shows:

$$\begin{tikzcd}
L\Gamma\Phi \arrow[r, "L\Gamma r"] \arrow[d, "\chi_\Phi"'] & L\Gamma FX \arrow[r, "L\zeta_X"] \arrow[d, "\chi_{FX}"'] & LFX\Sigma \arrow[r, "\rho_{\Sigma X}"] & FB\Sigma X \arrow[d, "F\lambda_X"] \\
\Gamma L\Phi \arrow[r, "\Gamma Lr"'] & \Gamma LFX \arrow[r, "\Gamma\rho_X"'] & \Gamma FBX \arrow[r, "\zeta_{BX}"'] & F\Sigma BX
\end{tikzcd}$$

where the left part commutes by naturality of χ, and the right part is (24). Naturality of κ follows from that of λ and χ. $\square$

Remark. The correspondence of κ and (λ, χ) above is actually bijective. In particular, given a $\kappa : \Sigma^\zeta B^\rho \Longrightarrow B^\rho\Sigma^\zeta$, one can extract λ_X as the first component of $\kappa_{(X,\mathrm{id}_{FX},FX)}$. This, however, will not be used in the following development.

The following informal picture shows categories, functors and natural transformations involved in Proposition 1:

$$\begin{tikzcd}
 & \rho{:}LF\Longrightarrow FB & \\
\lambda{:}\Sigma B\Longrightarrow B\Sigma & \mathcal{C} \arrow[loop above, "B"] \arrow[loop below, "\Sigma"'] \arrow[r, bend left, "F"] \arrow[r, phantom, "\perp"] & \mathcal{D} \arrow[loop above, "L"] \arrow[loop below, "\Gamma"'] \arrow[l, bend left, "G"] & \chi{:}L\Gamma\Longrightarrow\Gamma L \; ; \\
 & \zeta{:}\Gamma F\Longrightarrow F\Sigma &
\end{tikzcd}$$

the contravariance of F and G is marked with crossed arrow tails. Σ, B and λ model a language syntax, behaviour and an SOS specification, as described in §3. L and ρ model a modal logic for B-coalgebras, as described in §4. The following theorem says that if the remaining ingredients Γ, ζ and χ can be found, then the logical equivalence induced by the logic on the transition system h_λ induced from the SOS specification is a congruence.

Theorem 3 *Under the above notation, for given Σ, B, λ, L and ρ, if Σ and L have initial algebras $a_\Sigma : \Sigma P \to P$ and $a_L : L\Phi \to \Phi$, and if some Γ, ζ and χ exist such that* (24) *holds, then the kernel of $\llbracket\text{-}\rrbracket^\flat_{h_\lambda} : P \to G\Phi$ is a congruence, i.e., $\llbracket\text{-}\rrbracket^\flat_{h_\lambda}$ is a Σ-algebra morphism from the initial Σ-algebra.*

Proof. Initial algebras $a_\Sigma : \Sigma P \to P$ and $a_L : L\Phi \to \Phi$ induce initial λ- and χ-bialgebras as in (10):

$$\begin{tikzcd}
\Sigma P \arrow[r, "\Sigma h_\lambda"] \arrow[dd, "a_\Sigma"'] & \Sigma B P \arrow[d, "\lambda_P"] & L\Gamma\Phi \arrow[d, "\chi_\Phi"'] & L\Phi \arrow[l, "Lh_\chi"'] \arrow[dd, "a_L"] \\
& B\Sigma P \arrow[d, "Ba_\Sigma"] & \Gamma L\Phi \arrow[d, "\Gamma a_L"'] & \\
P \arrow[r, dotted, "h_\lambda"'] & BP & \Gamma\Phi & \Phi. \arrow[l, dotted, "h_\chi"]
\end{tikzcd} \tag{25}$$

Then $\llbracket\text{-}\rrbracket_{h_\lambda}$ is a "twisted coalgebra morphism" as below:

$$\begin{tikzcd}
\Gamma F P \arrow[r, "\zeta_P"] & F\Sigma P & FP \arrow[l, "Fa_\Sigma"'] \\
\Gamma\Phi \arrow[u, "\Gamma\llbracket\text{-}\rrbracket_{h_\lambda}"] & & \Phi. \arrow[ll, "h_\chi"] \arrow[u, "\llbracket\text{-}\rrbracket_{h_\lambda}"']
\end{tikzcd} \tag{26}$$

This is proved by L-induction, as both sides of this diagram are algebra morphisms from the initial L-algebra to $F(\lambda_P \circ \Sigma h_\lambda) \circ \rho_{\Sigma P} : LF\Sigma P \to F\Sigma P$. Indeed, in the diagram

$$\begin{tikzcd}
L\Phi \arrow[r, "L\llbracket\text{-}\rrbracket_{h_\lambda}"] \arrow[ddd, "a_L"'] & LFP \arrow[r, "LFa_\Sigma"] \arrow[d, "\rho_P"'] & LF\Sigma P \arrow[d, "\rho_{\Sigma P}"] \\
& FBP \arrow[r, "FBa_\Sigma"] \arrow[dd, "Fh_\lambda"'] & FB\Sigma P \arrow[d, "F\lambda_P"] \\
& & F\Sigma BP \arrow[d, "F\Sigma h_\lambda"] \\
\Phi \arrow[r, "\llbracket\text{-}\rrbracket_{h_\lambda}"'] & FP \arrow[r, "Fa_\Sigma"'] & F\Sigma P
\end{tikzcd}$$

the left part is (22), the upper right part commutes by the naturality of ρ, and the lower right part is the left diagram in (25) mapped along F. On the other hand, in the diagram

$$\begin{tikzcd}
L\Phi \arrow[r, "Lh_\chi"] \arrow[ddd, "a_L"'] & L\Gamma\Phi \arrow[r, "L\Gamma[\![\text{-}]\!]_{h_\lambda}"] \arrow[d, "\chi_\Phi"'] & L\Gamma FP \arrow[r, "L\zeta_P"] \arrow[d, "\chi_{FP}"'] & LF\Sigma P \arrow[d, "\rho_{\Sigma P}"] \\
& \Gamma L\Phi \arrow[r, "\Gamma L[\![\text{-}]\!]_{h_\lambda}"'] \arrow[dd, "\Gamma a_L"'] & \Gamma LFP \arrow[d, "\Gamma\rho_P"'] & FB\Sigma P \arrow[d, "F\lambda_P"] \\
& & \Gamma FBP \arrow[r, "\zeta_{BP}"] \arrow[d, "\Gamma Fh_\lambda"'] & F\Sigma BP \arrow[d, "F\Sigma h_\lambda"] \\
\Phi \arrow[r, "h_\chi"'] & \Gamma\Phi \arrow[r, "\Gamma[\![\text{-}]\!]_{h_\lambda}"'] & \Gamma FP \arrow[r, "\zeta_P"'] & F\Sigma P
\end{tikzcd}$$

the left part is the diagram on the right in (25), the upper middle part commutes by the naturality of χ, the lower middle part is (22) mapped along Γ, the upper right part is (24), and the lower right part commutes by the naturality of ζ.

Mapped along G, (26) is the upper right part of the following diagram, where the upper left part commutes by the naturality of η, the lower left part by (21) and by general properties of adjunctions, and the lower right part is the naturality of $\zeta^\star$ (see (21)):

$$\begin{tikzcd}
P \arrow[r, "\eta_P"] & GFP \arrow[r, "G[\![\text{-}]\!]_{h_\lambda}"] & G\Phi \\
& GF\Sigma P \arrow[u, "GFa_\Sigma"] \arrow[d, "G\zeta_P"] & \\
\Sigma P \arrow[uu, "a_\Sigma"] \arrow[ur, "\eta_{\Sigma P}"] \arrow[dr, "\Sigma\eta_P"'] & G\Gamma FP \arrow[r, "G\Gamma[\![\text{-}]\!]_{h_\lambda}"] & G\Gamma\Phi \arrow[uu, "Gh_\chi"'] \\
& \Sigma GFP \arrow[u, "\zeta^\star_{FP}"'] \arrow[r, "\Sigma G[\![\text{-}]\!]_{h_\lambda}"'] & \Sigma G\Phi \arrow[u, "\zeta^\star_\Phi"']
\end{tikzcd}$$

Thus $[\![\text{-}]\!]^\flat_{h_\lambda} = G[\![\text{-}]\!]_{h_\lambda} \circ \eta_P$ is a Σ-algebra morphism from a_Σ. $\square$

Intuitively, Γ, ζ and χ provide a way of decomposing modal formulas over process syntax. The functor Γ provides a notion of process-syntactic behaviour to logical formulas, with $\zeta : \Gamma F \Longrightarrow F\Sigma$ providing a connection to process syntax, and $\chi : L\Gamma \Longrightarrow \Gamma L$ defining a way to define the behaviour by induction on logical formulas, just as $\lambda : \Sigma B \Longrightarrow B\Sigma$ allows one to define behaviour for processes by induction. Some examples supporting this intuition are described in §6.

To generalize the framework of §5.1 to distributive laws λ of type (15) and to abstract GSOS (18), some technicalities are necessary. Assume both $\mathcal{C}$ and $\mathcal{D}$ have products and coproducts. A connection $\zeta : \Gamma F \Longrightarrow F\Sigma$ induces a connection $\zeta^\times$ between the free pointed functor over Σ and the cofree copointed functor over Γ:

$$\zeta^\times : (\mathrm{Id} \times \Gamma)F \Longrightarrow F(\mathrm{Id} + \Sigma) = F \times F\Sigma$$

defined by $\zeta^\times = \mathrm{id} \times \zeta$. Moreover, a connection $\rho : LF \Longrightarrow FB$ induces a connection

$$\rho^+ : (\mathrm{Id} + L)F \Longrightarrow F(\mathrm{Id} \times B);$$

to define it, define its adjoint (see (21))

$$\rho^{+\star} : (\mathrm{Id} \times B)G \Longrightarrow G(\mathrm{Id} + L) = G \times GL$$

by $\rho^{+\star} = \mathrm{id} \times \rho^\star$.

Theorem 3 can be generalized to distributive laws $\lambda : \Sigma(\mathrm{Id}\times B) \Longrightarrow B(\mathrm{Id}+\Sigma)$ as follows:

Theorem 4 *Under the above notation, for given Σ, B, $\lambda : \Sigma(Id \times B) \Longrightarrow B(Id + \Sigma)$, L and $\rho : LF \Longrightarrow FB$, if Σ and L have initial algebras, and if Γ on $\mathcal{D}$, $\zeta : \Gamma F \Longrightarrow F\Sigma$ and $\chi : L(Id \times \Gamma) \Longrightarrow \Gamma(Id + L)$ exist such that*

$$\begin{tikzcd}
L(Id \times \Gamma)F \arrow[r, Rightarrow, "L\zeta^\times"] \arrow[d, Rightarrow, "\chi F"'] & LF(Id + \Sigma) \arrow[r, Rightarrow, "\rho(Id+\Sigma)"] & FB(Id + \Sigma) \arrow[d, Rightarrow, "F\lambda"] \\
\Gamma(Id + L)F \arrow[r, Rightarrow, "\Gamma\rho^+"'] & \Gamma F(Id \times B) \arrow[r, Rightarrow, "\zeta(Id\times B)"'] & F\Sigma(Id \times B)
\end{tikzcd} \tag{27}$$

(compare with (24)) commutes, then $[\![\_]\!]^\flat_{h_\lambda}$ is a Σ-algebra morphism from the initial Σ-algebra.

The proof of this proceeds as for Theorem 3. Note that χ needs to be generalized to (15) along with λ.

Further, assume that Σ freely generates a monad T_Σ on $\mathcal{C}$, i.e., that $T_\Sigma X$ is the carrier of an initial $(X + \Sigma-)$-algebra, and that Γ cofreely generates a comonad D_Γ on $\mathcal{D}$, i.e., that $D_\Gamma\Phi$ is the carrier of a final $(\Phi \times \Gamma-)$-coalgebra. Then $\zeta : \Gamma F \Longrightarrow F\Sigma$ induces a connection $\zeta^\sharp : D_\Gamma F \Longrightarrow FT_\Sigma$. To define it, define its adjoint $\zeta^{\sharp\star} : T_\Sigma G \Longrightarrow GD_\Gamma$ from $\zeta^\star : \Sigma G \Longrightarrow G\Gamma$ pointwise by induction in $\mathcal{C}$, as the unique algebra map:

$$\begin{tikzcd}
G\Phi + \Sigma T_\Sigma G\Phi \arrow[r, "\mathrm{id}_{G\Phi}+\Sigma\zeta^{\sharp\star}_{\Phi}"] \arrow[dd, "a"'] & G\Phi + \Sigma G D_\Gamma \Phi \arrow[d, "\mathrm{id}_{G\Phi}+\zeta^{\star}_{D_\Gamma\Phi}"] \\
 & G\Phi + G\Gamma D_\Gamma \Phi \arrow[d, "{[G\epsilon_\Phi, G\delta_\Phi]}"] \\
T_\Sigma G\Phi \arrow[r, dotted, "\zeta^{\sharp\star}_{\Phi}"'] & G D_\Gamma \Phi
\end{tikzcd} \tag{28}$$

where $\langle \epsilon_\Phi, \delta_\Phi\rangle : D_\Gamma\Phi \to \Phi \times \Gamma D_\Gamma \Phi$ is the final $(\Phi \times \Gamma -)$-coalgebra.

Theorem 3 can be generalized to distributive laws $\lambda : \Sigma(\mathrm{Id} \times B) \Longrightarrow BT_\Sigma$ as follows:

Theorem 5 *Under the above notation, for given Σ (and its freely generated monad T_Σ), B, $\lambda : \Sigma(Id \times B) \Longrightarrow BT_\Sigma$, L and $\rho : LF \Longrightarrow FB$, if Σ and L have initial algebras, and if Γ (and its cofreely generated comonad D_Γ) on $\mathcal{D}$, $\zeta : \Gamma F \Longrightarrow F\Sigma$ and $\chi : LD_\Gamma \Longrightarrow \Gamma(Id + L)$ exist such that*

$$\begin{tikzcd}
LD_\Gamma F \arrow[r, Rightarrow, "L\zeta^\sharp"] \arrow[d, Rightarrow, "\chi F"'] & LFT_\Sigma \arrow[r, Rightarrow, "\rho T_\Sigma"] & FBT_\Sigma \arrow[d, Rightarrow, "F\lambda"] \\
\Gamma(Id + L)F \arrow[r, Rightarrow, "\Gamma\rho^+"'] & \Gamma F(Id \times B) \arrow[r, Rightarrow, "\zeta(Id\times B)"'] & F\Sigma(Id \times B)
\end{tikzcd} \tag{29}$$

(compare with (24), (27)) commutes, then $[\![\_]\!]^\flat_{h_\lambda}$ again is a Σ-algebra morphism from the initial Σ-algebra.

The proof of this proceeds as for Theorem 3. Note that while λ is generalized to abstract GSOS (18), the logical distributive law χ needs to be generalized to abstract safe ntree (19).

6 Examples

In this section the framework developed in §5 is illustrated on four simple examples, aimed at explaining the workings of logical distributive laws rather than at exploring the full scope of our approach. First, a very simple example of an SOS specification in the abstract toy SOS format is explained in detail. Then, trace equivalence is proved compositional for a subset of CCS, followed by a proof that de Simone format is a congruence format for trace equivalence. Finally, completed trace equivalence is proved compositional for a language with binary Kleene star, which does not conform to any previously known congruence format for completed traces.

These examples are aimed at explaining the inner workings of logical distributive laws at a basic level rather than at exploring the full scope of our approach. Therefore, although the abstract developments of previous sections are naturally presented for arbitrary categories, in all examples in this section we shall take $\mathcal{C} = \mathcal{D} = \mathbf{Set}$, $F = G = 2^{-}$ and $B = (\mathcal{P}_\omega -)^A$, for a fixed set A of labels. Unless stated otherwise, A is assumed to be finite.

6.1 A toy SOS specification

Consider the language (8) from §3.3, with synchronous product and no sequential composition. Categorically, its syntax is modeled by the functor

$$\Sigma X = \{\mathtt{nil}\} + A + X \times X$$

on **Set**, and the rules (8) specify a distributive law $\lambda : \Sigma B \Longrightarrow B\Sigma$.

We will apply the framework of §5 to prove that trace equivalence is compositional for this language. The compositionality result is hardly interesting in itself (and indeed easy to prove without any advanced techniques), but it should be useful to explain our approach on such a very simple instance of abstract toy SOS.

The logic for trace equivalence on B-coalgebras (image finite LTSs) is defined as in Example 1, but with syntax restricted to

$$L\Phi = \{\top\} + A \times \Phi.$$

For the required compositionality result, Theorem 3 requires a functor Γ on **Set** and transformations $\zeta : \Gamma 2^{-} \to 2^{\Sigma -}$ and $\chi : L\Gamma \Longrightarrow \Gamma L$ such that (24) commutes.

Initially it might not be clear how to look for the right Γ. To illustrate the role of this functor and explain the process of finding ζ and χ, we begin with a very simple and natural (although, as we shall see, wrong) choice, where $\Gamma = \Sigma$ and ζ is defined as follows:

$$\begin{aligned} \zeta_X(\mathtt{nil})(t) = \mathtt{tt} &\iff t = \mathtt{nil} \\ \zeta_X(a)(t) = \mathtt{tt} &\iff t = a \\ \zeta_X(\phi_1 \otimes \phi_2)(t) = \mathtt{tt} &\iff t = x_1 \otimes x_2, \phi_i(x_i) = \mathtt{tt} \end{aligned}$$

along the lines of §4. Constructors $\mathtt{nil}$, a and $\otimes$ used here will be called *spatial modalities*, as opposed to *behavioural* modalities $\top$ and $\langle a \rangle$ used in the definition of L. This is motivated by "spatial logics" of [14], where similar logical operators based on process syntax are present. Intuitively, formulas built from these spatial modalities can check the structure of Σ-terms.

One might now attempt to define a distributive law $\chi : L\Gamma \Longrightarrow \Gamma L$ such that (24) commutes. Since both L and Γ are polynomial functors, such a law can be defined by cases, separately for each combination of modalities from spatial and behavioural modalities. Then (24) can also be proved by cases. For example, consider the following partial definition of χ:

$$\chi_\Phi(\langle a\rangle(\phi_1 \otimes \phi_2)) = (\langle a\rangle\phi_1)\otimes(\langle a\rangle\phi_2).$$

Note that both the argument on the left side and the right side of this equation have a simple intuitive meaning: the former says "the process can do an a-step to a process of the form $y_1 \otimes y_2$ such that ϕ_1 holds for y_1 and ϕ_2 holds for y_2", and the latter says "the process is of the form $x_1 \otimes x_2$, x_1 can do an a-step to a process for which ϕ_1 holds, and x_2 can do an a-step to a process for which ϕ_2 holds". A quick look on (8) should convince anyone that these conditions are equivalent; formally, the corresponding case of (24) commutes, as the following calculation shows:

$$\begin{aligned}
& 2^{\lambda_X}(\rho_{\Sigma X}(L\zeta_X(\langle a\rangle(\phi_1\otimes\phi_2))))(t) = \mathtt{tt}\\
\Longleftrightarrow\ & \rho_{\Sigma X}(\langle a\rangle(\zeta_X(\phi_1\otimes\phi_2)))(\lambda_X(t)) = \mathtt{tt}\\
\Longleftrightarrow\ & \exists r \in (\lambda_X(t))(a).\ \zeta_X(\phi_1\otimes\phi_2)(r) = \mathtt{tt}\\
\Longleftrightarrow\ & \exists r \in (\lambda_X(t))(a).\ r = x_1 \otimes x_2, \phi_i(x_i) = \mathtt{tt}\\
\overset{\star}{\Longleftrightarrow}\ & t = \beta_1 \otimes \beta_2, \exists y_i \in \beta_i(a).\ \phi_i(y_i) = \mathtt{tt}\\
\Longleftrightarrow\ & t = \beta_1 \otimes \beta_2, \rho_X(\langle a\rangle\phi_i)(\beta_i) = \mathtt{tt}\\
\Longleftrightarrow\ & \zeta_{BX}((\rho_X(\langle a\rangle\phi_1))\otimes(\rho_X(\langle a\rangle\phi_2)))(t) = \mathtt{tt}\\
\Longleftrightarrow\ & \zeta_{BX}(\Gamma\rho_X((\langle a\rangle\phi_1)\otimes(\langle a\rangle\phi_2)))(t) = \mathtt{tt}\\
\Longleftrightarrow\ & \zeta_{BX}(\Gamma\rho_X(\chi_{2^X}(\langle a\rangle(\phi_1\otimes\phi_2))))(t) = \mathtt{tt},
\end{aligned}$$

where the marked equivalence follows from the definition of λ, and other equivalences are straightforward applications of the definitions of ρ, ζ and χ.

Unfortunately, other cases of χ are harder to define. Already the simple behavioural modality $\top$ is problematic: for $\chi_\Phi(\top)$ one would like an element of $\Gamma L\Phi$ that would represent the always true condition. This is, however, impossible with our initial choice of Γ: every test in $\Gamma L\Phi$ imposes some syntactic condition on the tested process. A simple attempt to overcome this problem would be to add a single constant, always true modality $\mathbb{T} \in \Sigma 1 \to 2$ to Γ, with ζ extended by:

$$\zeta_X(\mathbb{T})(t) = \mathtt{tt} \text{ always.}$$

Then, however, it becomes unclear what $\chi_\Phi(\langle a\rangle\mathbb{T})$ should be. To formalize the condition "the process can do an a-step" as an element of $\Gamma L\Phi$, one would like to write something like:

$$\chi_\Phi(\langle a\rangle\mathbb{T}) = a \vee (\langle a\rangle\top)\otimes(\langle a\rangle\top);$$

this is, however, forbidden as Γ does not allow one to write anything like logical disjunction in spatial modalities.

A solution to this problem is to extend Γ more substantially by closing spatial modalities under finite disjunctions (this yields modalities of any finite arity, just as closing behavioural modalities in HML did in §4, Example 2). This amounts to taking $\Gamma = \mathcal{P}_\omega\Sigma$ with $\zeta : \Gamma(2^-) \Longrightarrow 2^{\Sigma-}$ defined by:

$$\begin{aligned}\zeta_X(\gamma)(\mathtt{nil}) = \mathtt{tt} &\iff \mathtt{nil} \in \gamma\\ \zeta_X(\gamma)(a) = \mathtt{tt} &\iff a \in \gamma\\ \zeta_X(\gamma)(x_1 \otimes x_2) = \mathtt{tt} &\iff \exists \phi_1 \otimes \phi_2 \in \gamma.\ \phi_i(x_i) = \mathtt{tt}.\end{aligned}$$

One could then define χ with a set of equations as before, writing for example

$$\begin{aligned}\chi_\Phi(\top) &= \mathtt{nil} \vee (\top\otimes\top) \vee \textstyle\bigvee_{a\in A} a,\\ \chi_\Phi(\langle a\rangle(\mathtt{nil} \vee (\phi_1\otimes\phi_2))) &= a \vee (\langle a\rangle\phi_1\otimes\langle a\rangle\phi_2)\end{aligned}$$

and so forth. However, the theory of GSOS in §3 provides another, more elegant method of presenting such distributive laws: inference rules. Note that the functor $\mathcal{P}_\omega\Sigma$ is rather similar to $B = (\mathcal{P}_\omega-)^A$, and it is reasonable to expect that χ could be presented in a manner similar to GSOS rules. Two differences between $\mathcal{P}_\omega\Sigma$-coalgebras and B-coalgebras are that in the former there are no labelled components in transitions, and successors are flat Σ-terms rather than simple elements. This suggests that instead of literals like $\mathtt{x} \xrightarrow{a} \mathtt{y}$, in rules for χ one should use literals such as $\mathtt{x} \longrightarrow \mathtt{y}\otimes\mathtt{z}$. To distinguish the logical rules from the SOS ones, we will use variables such as ϕ, ψ, instead of $\mathtt{x}, \mathtt{y}$, and we will replace the sign $\longrightarrow$ with $\dashv$. Now the following rules:

$$\frac{}{\top \dashv \mathtt{nil}} \qquad \frac{}{\top \dashv a} \qquad \frac{}{\top \dashv \top\otimes\top}$$
$$\frac{\phi \dashv \mathtt{nil}}{\langle a\rangle\phi \dashv a} \qquad \frac{\phi \dashv \psi\otimes\sigma}{\langle a\rangle\phi \dashv (\langle a\rangle\psi)\otimes(\langle a\rangle\sigma)} \tag{30}$$

where a ranges over A, define a distributive law $\chi : L\Gamma \Longrightarrow \Gamma L$ just as (8) defined λ in §3.3:

$$\begin{aligned}\chi_\Phi(\top) &= \{\mathtt{nil}\} \cup A \cup \{\top\otimes\top\}\\ \chi_\Phi(\langle a\rangle\gamma) &= \{\, a \mid \mathtt{nil} \in \gamma \,\} \cup \{\, \langle a\rangle\psi\otimes\langle a\rangle\sigma \mid \psi\otimes\sigma \in \gamma \,\}\end{aligned}$$

where a ranges over A and $\gamma \in \Gamma\Phi$. Here and in the following, $\{\, a \mid \mathtt{nil} \in \gamma \,\}$ means "$\{a\}$ if $\mathtt{nil} \in \gamma$, otherwise $\emptyset$".

Note that behavioural modalities $\top$, $\langle a\rangle$ play the role of *syntax* here, and spatial modalities $\mathtt{nil}$, a and $\otimes$ are a part of the *behaviour*. The sign $\dashv$ might be read "is guaranteed by"; this is justified by the definition of ζ.

It turns out that for this χ the condition (24) holds. This is proved by cases, according to the structure of L. The first case is:

$$\begin{aligned}
& 2^{\lambda_X}(\rho_{\Sigma X}(L\zeta_X(\top)))(t) = \mathtt{tt} \\
\iff & \rho_{\Sigma X}(\top)(\lambda_X t) = \mathtt{tt} \\
\overset{\star}{\iff} & (t = \mathtt{nil}) \text{ or } (t \in A) \text{ or } (\exists \beta_1, \beta_2 \in BX.\ t = x_1 \otimes x_2) \\
\iff & \zeta_{BX}(\{\mathtt{nil}\} \cup A \cup \{\rho_{\Sigma X}(\top) \otimes \rho_{\Sigma X}(\top)\})(t) = \mathtt{tt} \\
\iff & \zeta_{BX}(\Gamma\rho_{\Sigma X}(\{\mathtt{nil}\} \cup A \cup \{\top \otimes \top\})(t) = \mathtt{tt} \\
\iff & \zeta_{BX}(\Gamma\rho_X(\chi_{2^X}(\top))(t) = \mathtt{tt};
\end{aligned}$$

note that both sides of the marked equivalence are always true. The second case is:

$$\begin{aligned}
& 2^{\lambda_X}(\rho_{\Sigma X}(L\zeta_X(\langle a\rangle\gamma)))(t) = \mathtt{tt} \\
\iff & \rho_{\Sigma X}(\langle a\rangle(\zeta_X(\gamma)))(\lambda_X t) = \mathtt{tt} \\
\iff & \exists r \in (\lambda_X t)(a).\ \zeta_X(\gamma)(r) = \mathtt{tt} \\
\iff & (\mathtt{nil} \in \gamma \text{ and } \mathtt{nil} \in (\lambda_X t)(a)) \\
& \quad \text{or } (\exists \phi_1 \otimes \phi_2 \in \gamma, x_1 \otimes x_2 \in (\lambda_X t)(a).\ \phi_i(x_i) = \mathtt{tt}) \\
\overset{\star}{\iff} & (\mathtt{nil} \in \gamma \text{ and } t = a) \\
& \quad \text{or } (t = \beta_1 \otimes \beta_2, \exists \phi_1 \otimes \phi_2 \in \gamma. \exists y_i \in \beta_i(a).\ \phi_i(y_i) = \mathtt{tt}) \\
\iff & \zeta_{BX}(\{\, a \mid \mathtt{nil} \in \gamma \,\} \cup \{\, \rho_\Sigma(\langle a\rangle\psi) \otimes \rho_\Sigma(\langle a\rangle\sigma) \mid \psi \otimes \sigma \in \gamma \,\})(t) = \mathtt{tt} \\
\iff & \zeta_{BX}(\Gamma\rho_\Sigma(\{\, a \mid \mathtt{nil} \in \gamma \,\} \cup \{\, \langle a\rangle\psi \otimes \langle a\rangle\sigma \mid \psi \otimes \sigma \in \gamma \,\}))(t) = \mathtt{tt} \\
\iff & \zeta_{BX}(\Gamma\rho_\Sigma(\chi_{FX}(\langle a\rangle\gamma)))(t) = \mathtt{tt}.
\end{aligned}$$

Here, the marked equivalence is true by the definition of λ.

This calculation, admittedly rather tedious, is a straightforward sequence of simple unfolding of the definitions of ρ, λ, χ and ζ. Rather than performing the calculation immediately, it is not difficult to convince oneself informally that it is worth performing, by looking at (8) and (30) and noting that, for example, if a property ϕ holds for all processes $x \otimes y$ such that ψ holds for x and σ holds for y, then $\langle a\rangle\phi$ holds for all processes $z \otimes w$ such that $\langle a\rangle\psi$ holds for z and $\langle a\rangle\sigma$ holds for w, which means that the last rule in (30) is correct. In the remaining examples in this paper, precise calculations as above will be omitted.

Since the condition (24) holds for the chosen Γ, ζ and χ, by Theorem 3 the map $[\![\_]\!]^\flat_{h_\lambda} : T_\Sigma 0 \to GT_L 0$ is an algebra morphism from the initial Σ-algebra. This means that its kernel is a congruence. On the other hand by the definition of L and ρ, the kernel of $[\![\_]\!]^\flat_{h_\lambda}$ coincides with trace equivalence on h_λ, the LTS induced by the rules (8). This gives the expected compositionality result.

So far the set A has been assumed finite. However, for infinite A very similar constructions of Γ, ζ and χ work, with the only difference that in the definition of Γ, instead of $\mathcal{P}_\omega$, a powerset bounded by a cardinal higher than the cardinality of A must be used; otherwise the definition of χ by (30) would be incorrect (a bound on the powerset construction is necessary for B to have an initial algebra). Obviously, the definitions of Σ, B, λ and ρ remain unchanged for an infinite A.

6.2 A step towards GSOS: recursion-free CCS

For a slightly more complex, but still very simple example, consider the recursion-free fragment of CCS, with syntax and semantics defined by (16) in §3.4, extended with unary operators:

$$t ::= \cdots \mid t[f] \mid t\backslash L$$

for each $L \subseteq A$ and each function $f : A \to A$ such that $f(\tau) = \tau$ and $f(\bar{a}) = \overline{f(a)}$ (denote the set of such functions by $\mathcal{F}(A)$). The functor on **Set** corresponding to this syntax is

$$\Sigma X = \{\mathtt{nil}\} + A \times X + X \times X + X \times X + X \times \mathcal{F}(A) + X \times \mathcal{P}(A)$$

and the rules (16), extended with

$$\frac{\mathtt{x} \xrightarrow{b} \mathtt{x}'}{\mathtt{x}[f] \xrightarrow{a} \mathtt{x}'[f]}\, a = f(b) \qquad \frac{\mathtt{x} \xrightarrow{a} \mathtt{x}'}{\mathtt{x}\backslash L \xrightarrow{a} \mathtt{x}'\backslash L}\, a, \bar{a} \notin L,$$

where L ranges over $\mathcal{P}(A)$ and f over $\mathcal{F}(A)$, define a distributive law of the form (15).

To prove the compositionality of trace equivalence for this language, consider L and ρ as in §6.1, and use Theorem 4. It requires a functor Γ, $\zeta : \Gamma(2^-) \Longrightarrow 2^{\Sigma-}$ and $\chi : L(\mathrm{Id} \times \Gamma) \Longrightarrow \Gamma(\mathrm{Id} + L)$ such that (27) holds. Take $\Gamma = \mathcal{P}_\omega\Sigma$ and define ζ in analogy with §6.1, and let χ be defined by the following rules:

$$\frac{}{\top \dashv \mathtt{nil}} \qquad \frac{}{\top \dashv a.\top} \qquad \frac{}{\top \dashv \top{+}\top} \qquad \frac{}{\top \dashv \top||\top} \qquad \frac{}{\top \dashv \top[f]} \qquad \frac{}{\top \dashv \top\backslash L}$$

$$\frac{}{\langle a\rangle\phi \dashv a.\phi} \qquad \frac{}{\langle a\rangle\phi \dashv \langle a\rangle\phi{+}\top} \qquad \frac{}{\langle a\rangle\phi \dashv \top{+}\langle a\rangle\phi}$$

$$\frac{\phi \dashv \psi||\sigma}{\langle a\rangle\phi \dashv \langle a\rangle\psi||\sigma} \qquad \frac{\phi \dashv \psi||\sigma}{\langle a\rangle\phi \dashv \psi||\langle a\rangle\sigma} \qquad \frac{\phi \vdash \psi||\sigma}{\langle\tau\rangle\phi \dashv \langle a\rangle\psi||\langle\bar{a}\rangle\sigma}$$

$$\frac{\phi \dashv \psi[f]}{\langle a\rangle\phi \dashv (\langle b\rangle\psi)[f]}\, a = f(b) \qquad \frac{\phi \dashv \psi\backslash L}{\langle a\rangle\phi \dashv (\langle a\rangle\phi)\backslash L}\, a, \bar{a} \notin L$$

as in §6.1. A tedious but straightforward calculation shows that (27) commutes. Again, rather that perform the formal calculation it is easier to convince oneself that the logical rules reflect the SOS rules of (16).

Note that χ is not of the abstract toy SOS type $L\Gamma \Longrightarrow \Gamma L$. For example, in the first rule for $\langle a\rangle$ above, the variable ϕ from the left side of the conclusion is used on the right side of the conclusion (hence at least $L(\mathrm{Id}\times\Gamma)$ is needed in the domain of χ), and it is not put under any behavioural modality (hence at least $\Gamma(\mathrm{Id}+L)$ is needed in the codomain of χ). This corresponds to the reason why λ is not of the type $\Sigma B \Longrightarrow B\Sigma$: look at the SOS rule for $a.$ in (16).

6.3 Abstract GSOS: de Simone format

In §6.1 and §6.2 the framework of logical distributive laws was applied to show compositionality for specific languages. In this section it is used to show that an entire format is a congruence format for a process equivalence. More specifically, we will re-prove the well-known fact that de Simone format (6) guarantees the compositionality of trace equivalence.

To this end consider B, L and ρ as in the previous two examples. For any Σ on **Set** corresponding to an algebraic signature, and for any $\lambda : \Sigma(\mathrm{Id}\times B) \Longrightarrow BT_\Sigma$ represented by a set R of inference rules in de Simone format, one needs to find a Γ on **Set**, $\zeta : \Gamma(2^-) \Longrightarrow 2^{\Sigma-}$ and $\chi : LD_\Gamma \Longrightarrow \Gamma(\mathrm{Id}+L)$ such that (29) commutes. This, by Theorem 5, will imply the required result. To simplify the presentation, assume Σ to be finite.

In this simple example, Γ or ζ can be chosen independently from λ, and indeed we choose $\Gamma = \mathcal{P}_\omega\Sigma$ and ζ as used in §6.1 and §6.2, with the powerset in Γ interpreted as finite disjunction:

$$\zeta_X(\gamma)(\mathtt{f}(x_1,\ldots,x_n)) = \mathtt{tt} \iff \exists \mathtt{f}(\phi_1,\ldots,\phi_n)\in\gamma.\ \forall i\in 1..n.\ \phi_i(x_i)=\mathtt{tt}$$

where $\gamma\in\Gamma(2^X)$.

To proceed with defining χ, it is useful to get some intuition about the cofree comonad D_Γ. First, consider the simpler functor D_Σ. Elements of $D_\Sigma\Phi$ are terms over Σ, possibly infinitely deep, with an element of Φ in every node. A simple intuitive difference between $T_\Sigma X$ and $D_\Sigma\Phi$ is that while an element (term) in $T_\Sigma X$ is *either* a variable from X *or* an operator from Σ with a tuple of subterms, an element in $D_\Sigma\Phi$ is a variable from Φ *and* an operator with a tuple of subterms. Therefore in particular, if there are no constant (0-ary) operators in Σ, all terms in $D_\Sigma\Phi$ are infinitely deep. In turn, in $D_\Gamma\Phi = D_{\mathcal{P}_\omega\Sigma}\Phi$ terms are nondeterministic in the sense that every node contains, in addition to an element of Φ, not a single operator but a finite set of operators (successors) with corresponding tuples of subterms.

In (29), the inductively defined (28) transformation $\zeta^\sharp$ appears. In this case, $\zeta^\sharp_X : D_\Gamma(2^X) \Longrightarrow 2^{T_\Sigma X}$ can be described with an auxiliary *compatibility relation* $\lhd \subseteq T_\Sigma X \times D_\Gamma(2^X)$, defined by induction: a variable $x \in X$ is compatible with a tree $d \in D_\Gamma(2^X)$ iff the root of d is labelled with a test $\phi \in 2^X$ such that $\phi(x) = \mathtt{tt}$, and a term $\mathtt{f}(t_1, \ldots, t_n)$ is compatible with a tree d iff there exists a successor of the root of d that is of the form $\mathtt{f}(d_1, \ldots, d_n)$ such that all t_i's are compatible with their respective d_i's. A similar compatibility relation (also denoted $\lhd$) between $T_\Sigma(2^\Phi)$ and $D_\Gamma\Phi$ can also be defined; the only difference is in the treatment of variables. Looking at (28) and at the definition of ζ above, it is not difficult to check that

$$\zeta^{\star\sharp}_\Phi(t)(d) = \mathtt{tt} \iff t \lhd d$$

for $t \in T_\Sigma(2^\Phi)$ and $d \in D_\Gamma\Phi$; as a result,

$$\zeta^\sharp_X(d)(t) = \mathtt{tt} \iff t \lhd d$$

for $t \in T_\Sigma X$ and $d \in D_\Gamma(2^X)$. In words, for a tree $d \in D_\Gamma(2^X)$, $\zeta^\sharp_X(d)$ is the set of all terms in $T_\Sigma X$ compatible with d.

A concrete presentation of distributive laws of the type $\chi : LD_\Gamma \Longrightarrow \Gamma(\mathrm{Id}+L)$ is similar to the safe ntree format (7), corresponding to the same type (19) of distributive laws for $B = (\mathcal{P}_\omega-)^A$ instead of $\Gamma = \mathcal{P}_\omega\Sigma$: in particular, rules of the form

$$\frac{\phi \dashv \mathtt{f_1}(\phi_1, \ldots, \phi_n) \quad \cdots \quad \phi_i \dashv \mathtt{f_i}(\phi_k, \ldots, \phi_m) \quad \cdots}{\langle a\rangle\phi \dashv \mathtt{g}(\ldots, \psi_i, \ldots, \langle b_j\rangle\psi_j, \ldots)}$$

are allowed, where all arities agree with Σ, the relation on formulas defined by the premises is well-founded, and a condition on variables being distinct analogous to that on (7) is satisfied. In fact more general rules could be allowed, with negative premises involved, but for the purpose of our example only χ's defined by this kind of positive rules will be needed.

We can now give a definition of $\chi : LD_\Gamma \Longrightarrow \Gamma(\mathrm{Id} + L)$ such that the hexagon (29) commutes. Obviously, the definition depends on λ and on the assumption that it corresponds to a set of rules in de Simone format. Based on the set of rules, denoted by R, construct a set of rules in the format of (19) as follows: for any rule

$$\frac{\{\mathtt{x}_i \xrightarrow{a_i} \mathtt{y}_i\}_{i \in I}}{\mathtt{g}(\mathtt{x}_1, \ldots, \mathtt{x}_n) \xrightarrow{b} \mathtt{t}}$$

in R, where $I \subseteq \{1, \ldots, n\}$ and only variables from $\{\, x_j \mid j \notin I \,\} \cup \{\, y_i \mid i \in I \,\}$

occur in t, include the rule

$$\frac{\{\phi_\nu \dashv \mathtt{f}_\nu(\phi_{\nu_1}, \ldots, \phi_{\nu_k})\}_{\nu \in N(\mathtt{t})}}{\langle b \rangle \phi \dashv \mathtt{g}(\psi_1, \ldots, \psi_n)}$$

where:

- $N(\mathtt{t})$ is the set of nodes (operator instances) in t,
- for each $\nu \in N(\mathtt{t}) \cup V(\mathtt{t})$, ϕ_ν is a fresh variable, where $V(\mathtt{t})$ is the set of variable instances in t,
- ϕ is the variable corresponding to the top operator in t (or, if t is a variable, to the variable), i.e., $\phi = \phi_\nu$ for a $\nu \in N(\mathtt{t}) \cup V(\mathtt{t})$,
- $\mathtt{f}_\nu$ is the operator in the node ν, $ar(\mathtt{f}_\nu) = k$, and $\nu_1, \ldots, \nu_k$ are the immediate subterms of ν in t,
- and for $i = 1..n$,

$$\psi_i = \begin{cases} \top & \text{if } i \notin I \text{ and } \mathtt{x}_i \text{ does not occur in } \mathtt{t}, \\ \phi_\nu & \text{if } i \notin I \text{ and } \mathtt{x}_i \text{ occurs in } \mathtt{t} \text{ on the position } \nu, \\ \langle a_i \rangle \top & \text{if } i \in I \text{ and } \mathtt{y}_i \text{ does not occur in } \mathtt{t}, \\ \langle a_i \rangle \phi_\nu & \text{if } i \in I \text{ and } \mathtt{y}_i \text{ occurs in } \mathtt{t} \text{ on the position } \nu. \end{cases}$$

Note that the above is well defined only because the original SOS rule is in de Simone format, and in particular because no variable can occur in t more than once.

A few examples of de Simone rules and their logical counterparts should clarify the above construction:

$$\frac{}{\mathtt{g(x)} \xrightarrow{b} \mathtt{x}} \qquad \frac{}{\langle b \rangle \phi \dashv \mathtt{g}(\phi)}$$

$$\frac{}{\mathtt{g(x)} \xrightarrow{b} \mathtt{f}} \qquad \frac{\phi \dashv \mathtt{f}}{\langle b \rangle \phi \dashv \mathtt{g}(\top)}$$

$$\frac{}{\mathtt{g(x)} \xrightarrow{b} \mathtt{h(x)}} \qquad \frac{\phi \dashv \mathtt{h}(\phi')}{\langle b \rangle \phi \dashv \mathtt{g}(\phi')}$$

$$\frac{}{\mathtt{g}(\mathtt{x}_1,\mathtt{x}_2) \xrightarrow{b} \mathtt{h}(\mathtt{x}_1)} \qquad \frac{\phi \dashv \mathtt{h}(\phi')}{\langle b\rangle\phi \dashv \mathtt{g}(\phi',\top)}$$

$$\frac{}{\mathtt{g}(\mathtt{x}_1,\mathtt{x}_2) \xrightarrow{b} \mathtt{h}(\mathtt{x}_1,\mathtt{k}(\mathtt{x}_2))} \qquad \frac{\phi \dashv \mathtt{h}(\phi',\phi'') \quad \phi'' \dashv \mathtt{k}(\phi''')}{\langle b\rangle\phi \dashv \mathtt{g}(\phi',\phi''')}$$

$$\frac{\mathtt{x} \xrightarrow{a} \mathtt{y}}{\mathtt{g}(\mathtt{x}) \xrightarrow{b} \mathtt{y}} \qquad \frac{}{\langle b\rangle\phi \dashv \mathtt{g}(\langle a\rangle\phi)}$$

$$\frac{\mathtt{x} \xrightarrow{a} \mathtt{y}}{\mathtt{g}(\mathtt{x}) \xrightarrow{b} \mathtt{f}} \qquad \frac{\phi \dashv \mathtt{f}}{\langle b\rangle\phi \dashv \mathtt{g}(\langle a\rangle\top)}$$

$$\frac{\mathtt{x}_2 \xrightarrow{a} \mathtt{y} \quad \mathtt{x}_4 \xrightarrow{c} \mathtt{z}}{\mathtt{g}(\mathtt{x}_1,\mathtt{x}_2,\mathtt{x}_3,\mathtt{x}_4) \xrightarrow{b} \mathtt{h}(\mathtt{x}_1,\mathtt{k}(\mathtt{y}))} \qquad \frac{\phi \dashv \mathtt{h}(\phi',\phi'') \quad \phi'' \dashv \mathtt{k}(\phi''')}{\langle b\rangle\phi \dashv \mathtt{g}(\phi',\langle a\rangle\phi''',\top,\langle c\rangle\top)}$$

For another example, see the logical rules for the recursion-free fragment of CCS in §6.2.

The above defines the logical behaviour for unary modalities $\langle b\rangle$ for $b \in A$. To complete the definition of χ, define the logical behaviour of the constant $\top$ by adding, for each $\mathtt{f} \in \Sigma$, the rule

$$\frac{}{\top \dashv \mathtt{f}(\top,\ldots,\top)}$$

with $ar(\mathtt{f})$ arguments on the right side of the conclusion.

Note that logical rules in §6.1 and §6.2 are obtained from their respective SOS specifications using this procedure. It is not difficult to check that the set of rules obtained this way defines a distributive law $\chi : LD_\Gamma \Longrightarrow \Gamma(\mathrm{Id}+L)$. A somewhat lengthy, but straightforward calculation, based on the definitions of λ, χ, ζ, $\zeta^\sharp$ and ρ proves that (29) commutes, hence trace equivalence is a congruence on LTSs induced by specifications in de Simone format.

The above is based on the assumption of Σ being finite. If it is infinite, a very similar construction works, with the only difference being that $\Gamma = \mathcal{P}_\omega\Sigma$ needs to be replaced with $\Gamma = \mathcal{P}_\kappa\Sigma$ for a cardinal κ larger than the cardinality of Σ. Otherwise, in particular, the rules for $\top$ given above would fail to represent a distributive law of the appropriate type. Note, however, that $\Gamma = \mathcal{P}\Sigma$ cannot be used, since it does not generate a cofree comonad for cardinality reasons.

We will now prove the compositionality of completed trace equivalence for a language which does not conform to any previously known format for that equivalence: the language with sequential composition and binary Kleene star, with syntax and semantics defined by (17). Its syntax is modeled by

$$\Sigma X = \{\mathtt{nil}\} + A + X \times X + X \times X$$

on **Set**. The distributive law defined by (17) is not in the form of (15), since a complex term is used in the conclusion of the first rule for $*$. However, since the rules are in the GSOS format, the law is of the form (18).

Consider the logic for completed trace equivalence with L and ρ as in Example 1. According to Theorem 5, to prove that completed trace equivalence is compositional one needs to find Γ, ζ and $\chi : LD_\Gamma \Longrightarrow \Gamma(\mathrm{Id}+L)$ such that (29) holds.

A tempting choice is $\Gamma = \mathcal{P}_\omega\Sigma$ which worked so well in previous examples, with an analogous definition of ζ interpreting the finite powerset construction as finite disjunctions of spatial modalities. In this example however, this choice does not work. Indeed, for a logical rule for the modality $\langle a\rangle$ that would reflect the first SOS rule for $*$ in (17), one is tempted to write something like:

$$\frac{\phi \dashv \psi;\sigma \qquad \sigma \dashv \kappa{*}\theta}{\langle a\rangle\phi \dashv (\langle a\rangle\psi \wedge \kappa){*}\theta}$$

since the variable $\mathtt{x}$ is indirectly duplicated in the conclusion of the SOS rule. This is, however, forbidden: the structure of L and Γ does not allow any use of conjunctions. An obvious solution is to extend Γ and allow finite conjunctions on the left side of the spatial modality $*$. Formally, consider

$$\Gamma\Phi = \mathcal{P}_\omega(1 + A + \Phi \times \Phi + \mathcal{P}_\omega\Phi \times \Phi) \tag{31}$$

with ζ defined by analogy to that in §6.1, with one difference:

$$\begin{gathered}\zeta(\gamma)(x * y) = \mathtt{tt} \\ \Updownarrow \\ \exists(\delta{*}\phi) \in \gamma.(\phi(y) = \mathtt{tt} \wedge \forall\psi \in \delta.\ \psi(x) = \mathtt{tt})\end{gathered}$$

where the universal quantifier justifies the understanding of the inner powerset in Γ as conjunction. Now χ can be defined by the following rules:

$$\frac{}{\top \dashv \mathtt{nil}} \qquad \frac{}{\top \dashv a} \qquad \frac{}{\top \dashv \top;\top} \qquad \frac{}{\top \dashv \top{*}\top}$$

$$\frac{}{\varnothing \dashv \mathtt{nil}} \qquad \frac{}{\varnothing \dashv \varnothing;\varnothing} \qquad \frac{}{\varnothing \dashv \varnothing{*}\varnothing}$$

$$\frac{\phi \dashv \mathtt{nil}}{\langle a\rangle\phi \dashv a} \qquad \frac{\phi \dashv \psi;\sigma}{\langle a\rangle\phi \dashv (\langle a\rangle\psi);\sigma} \qquad \frac{}{\langle a\rangle\phi \dashv \varnothing;\langle a\rangle\phi}$$

$$\frac{\phi \dashv \psi;\sigma \quad \sigma \dashv K{*}\theta}{\langle a\rangle\phi \dashv (\langle a\rangle\psi \wedge K){*}\theta} \qquad \frac{}{\langle a\rangle\phi \dashv \top{*}\langle a\rangle\phi}$$

where a ranges over A and K is a special variable denoting an arbitrary finite conjunction of formulas. The use of such variables in rules is justified by the structure of Γ, a little more complex than in the examples before. Again, a rather tedious calculation shows that (29) holds for this choice of χ, hence completed trace equivalence is compositional for our language. Here the calculation is a bit more complex than in §6.1 and §6.2, as it involves the inductively defined $\zeta^\sharp$.

Note that the logical rules above are not in the abstract GSOS format, as the first rule in the last row involves lookahead. However, it is in the abstract safe ntree format. This corresponds to the fact that SOS rules for our language are not in safe ntree format, as the first rule for $*$ has a complex term in the conclusion; however, they are in the GSOS format. Note how the lookahead in the logical rule stems from the complex conclusion in the SOS rule.

Note also that the behavioural modality $\varnothing$ is necessarily used in a logical rule for the modality $\langle a\rangle$. This suggests that there is no logical distributive law for the logic for traces for our language. Indeed, trace equivalence is not compositional for the sequential composition operator ;.

7 Conclusions and future work

We have presented a novel, general technique for proving the compositionality of process equivalences on languages defined by SOS. Our framework is a combination of the bialgebraic approach to SOS with a coalgebraic abstraction of modal logics that characterize interesting equivalences. Our main result, Theorem 3 and its generalizations in §5.2, can be seen as a very general "meta-congruence format", parametrized both by process equivalence and by the kind of transition system in question. The framework can be used either to prove compositionality for specific languages or to define congruence formats for various equivalences defined by modal logics.

The approach presented here is a refined and extended version of the framework of *test suites* [35], also aimed at deriving congruence formats. There, distributive laws are defined between fibered functors derived from notions of process equivalences, in total categories of certain fibrations. When $\mathcal{C} = \mathcal{D} = \mathbf{Set}$ and $F = G = \mathcal{V}^-$ for some set $\mathcal{V}$ of truth values, the present framework technically coincides with that of **Set**; note that $(\mathcal{C} \downarrow G)$ is always fibered over $\mathcal{C}$. However, in other cases the present approach provides a better treatment of equivalences. Moreover, it provides a clear connection to modal logic, and the presentation of distributive laws as SOS-like rules hopefully makes the entire framework easier to understand and to apply.

Some existing work on specific SOS formats and their properties can be rephrased in terms of the present framework. For example, the technique of frozen/liquid positions [9] used to derive formats for decorated trace equivalences corresponds exactly to extending the functors Γ with finite conjunctions as in in §6.4. More interestingly, the SOS-like presentation of logical distributive laws suggests a connection to compositional proof systems as in [56] and to techniques for modal logic decomposition as in [20]. Also, the notion of spatial modality used here seems to be related to spatial logics for process calculi as in [4,14], with a coalgebraic treatment suggested in [47]. The precise nature of these connections needs to be investigated.

Several other problems are left open. Importantly, some guidelines for finding the right logical behaviour Γ, and for checking whether it exists, are much needed instead of informal guessing used in §6. Also, the present framework does not deal with structural axioms (equations), which are an important ingredient in modern process specification formalisms. This is because in Theorems 3-5, logical equivalences are proved congruences on initial algebras, and other algebras, unlike e.g. in [13], are not treated. To model equations accurately, one might treat process syntax as arbitrary monads, as opposed to simple signature functors, with insights taken from e.g. [13,15]. Another common feature of process algebras not covered by our approach is recursion, typically described by rules that are not in GSOS format. To model (unguarded) recursion, techniques such as those used in [34] can be applied. Moreover, a treatment of quantitative logics, where e.g. metric spaces are used instead of equivalences, is presently missing. Last but not least, more examples involving various equivalences and kinds of transition systems, also in categories other than **Set** (e.g., presheaf categories for name-passing calculi), need to be developed in detail.

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] P. Aczel, N. Mendler, A final coalgebra theorem, in: Proc. CTCS'89, vol. 389 of LNCS, Springer, 1989.

[3] J. Adámek, H. Herrlich, G. E. Strecker, Abstract and Concrete Categories, Wiley-Interscience, 1990.

[4] H. R. Andersen, C. Stirling, G. Winskel, A compositional proof system for the modal μ-calculus, in: Procs. LICS'94, IEEE Computer Society Press, 1994.

[5] J. C. M. Baeten, C. A. Middelburg, Process algebra with timing: real time and discrete time, in: J. A. Bergstra, A. Ponse, S. Smolka (eds.), Handbook of Process Algebra, Elsevier, 2002, pp. 627–684.

[6] F. Bartels, GSOS for probabilistic transition systems, in: Proc. CMCS'02, vol. 65 of ENTCS, Elsevier, 2002.

[7] F. Bartels, On generalised coinduction and probabilistic specification formats, PhD dissertation, CWI, Amsterdam (2004).

[8] J. A. Bergstra, A. Ponse, S. Smolka, Handbook of Process Algebra, Elsevier, 2002.

[9] B. Bloom, W. J. Fokkink, R. J. van Glabbeek, Precongruence formats for decorated trace semantics, ACM Transactions on Computational Logic 5 (2004) 26–78.

[10] B. Bloom, S. Istrail, A. Meyer, Bisimulation can't be traced, Journal of the ACM 42 (1995) 232–268.

[11] M. Bonsangue, A. Kurz, Duality for logics of transition systems, in: Proc. FOSSACS'05, vol. 3441 of LNCS, 2005.

[12] M. Bonsangue, A. Kurz, Presenting functors by operations and equations, in: Proc. FOSSACS'06, vol. 3921 of LNCS, 2006.

[13] M. G. Buscemi, U. Montanari, A first order coalgebraic model of pi-calculus early observational equivalence, in: Procs. CONCUR, 2002.

[14] L. Caires, L. Cardelli, A spatial logic for concurrency (part I), Information and Computation 186 (2003) 194–235.

[15] A. Corradini, R. Heckel, U. Montanari, Compositional SOS and beyond: a coalgebraic view of open systems, Theoretical Computer Science 280 (2002) 163–192.

[16] 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.

[17] M. P. Fiore, G. D. Plotkin, D. Turi, Abstract syntax with variable binding, in: Proc. LICS'99, IEEE Computer Society Press, 1999, pp. 193–202.

[18] M. P. Fiore, D. Turi, Semantics of name and value passing, in: Proc. LICS'01, IEEE Computer Society Press, 2001, pp. 93–104.

[19] W. Fokkink, R. J. van Glabbeek, Ntyft/ntyxt rules reduce to ntree rules, Information and Computation 126 (1996) 1–10.

[20] W. J. Fokkink, R. J. van Glabbeek, P. de Wind, Compositionality of Hennessy-Milner logic through structural operational semantics, in: Proc. FCT'03, vol. 2751 of LNCS, Springer, 2003.

[21] M. J. Gabbay, A. M. Pitts, A new approach to abstract syntax with variable binding, Formal Aspects of Computing 13 (2001) 341–363.

[22] R. J. v. 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.

[23] J. F. Groote, M. Mousavi, M. A. Reniers, A hierarchy of SOS rule formats, in: Proc. SOS'05, 2005, Elsevier, 2005.

[24] M. Hennessy, R. Milner, Algebraic laws for nondeterminism and concurrency, Journal of the ACM 32 (1985) 137–161.

[25] B. Jacobs, Introduction to coalgebra: towards mathematics of states and observations, draft available from
`http://www.cs.ru.nl/B.Jacobs/CLG/JacobsCoalgebraIntro.pdf`.

[26] B. Jacobs, Distributive laws for the coinductive solution of recursive equations, Information and Computation 204 (2006) 561–587.

[27] B. Jacobs, J. J. M. M. Rutten, A tutorial on (co)algebras and (co)induction, Bulletin of the EATCS 62.

[28] B. Jonsson, W. Yi, K. G. Larsen, Probabilistic extensions of process algebras, in: J. A. Bergstra, A. Ponse, S. Smolka (eds.), Handbook of Process Algebra, Elsevier, 2002, pp. 685–710.

[29] M. Kick, Bialgebraic modelling of timed processes, in: Proc. ICALP'02, vol. 2380 of LNCS, Springer, 2002.

[30] M. Kick, Rule formats for timed processes, in: Proc. CMCIM'02, vol. 68 of ENTCS, Elsevier, 2002.

[31] M. Kick, J. Power, Modularity of behaviours for mathematical operational semantics, in: Procs. CMCS'04, vol. 106 of ENTCS, Elsevier, 2004.

[32] M. Kick, J. Power, A. Simpson, Coalgebraic semantics for timed processes, Information and Computation 204 (2006) 588–609.

[33] B. Klin, Abstract coalgebraic approach to process equivalence for well-behaved operational semantics, Ph.D. thesis, BRICS, Aarhus University (2004).

[34] B. Klin, Adding recursive constructs to bialgebraic semantics, Journal of Logic and Algebraic Programming 60-61 (2004) 259–286.

[35] B. Klin, From bialgebraic semantics to congruence formats, in: Proc. SOS 2004, vol. 128 of ENTCS, Elsevier, 2005.

[36] B. Klin, Bialgebraic metods in structural operational semantics, in: Proc. SOS'06, vol. 175 of ENTCS, Elsevier, 2007.

[37] B. Klin, Bialgebraic semantics and modal logic, in: Proc. LiCS'07, IEEE Computer Society Press, 2007.

[38] B. Klin, Coalgebraic modal logic beyond sets, in: Proc. MFPS 2007, vol. 173 of ENTCS, 2007.

[39] B. Klin, P. Sobocinski, Syntactic formats for free: An abstract approach to process equivalence, in: Proc. CONCUR 2003, vol. 2761 of LNCS, Springer, 2003.

[40] A. Kurz, Logics for coalgebras and applications to computer science, Ph.D. thesis, Universität München (2000).

[41] A. Kurz, Coalgebras and their logics, ACM SIGACT News 37.

[42] M. Lenisa, J. Power, H. Watanabe, Distributivity for endofunctors, pointed and co-pointed endofunctors, monads and comonads, in: Proc. CMCS'00, vol. 33 of ENTCS, Elsevier, 2000, pp. 230–260.

[43] M. Lenisa, J. Power, H. Watanabe, Category theory for operational semantics, Theoretical Computer Science 327 (1-2) (2004) 135–154.

[44] S. Mac Lane, Categories for the Working Mathematician, 2nd ed., Springer, 1998.

[45] R. Milner, Communication and Concurrency, Prentice Hall, 1988.

[46] R. Milner, Communicating and Mobile Systems: the π-Calculus, Cambridge University Press, 1999.

[47] L. Monteiro, A noninterleaving model of concurrency based on transition systems with spatial structure, ENTCS 106 (2004) 261–277.

[48] L. Moss, Coalgebraic logic, Annals of Pure and Applied Logic 96 (1999) 177–317.

[49] D. Pattinson, Expressivity results in the modal logic of coalgebras, Ph.D. thesis, Universität München (2001).

[50] D. Pavlovic, M. Mislove, J. B. Worrell, Testing semantics: connecting processes and process logics, in: Proc. AMAST'05, vol. 4019 of LNCS, Springer, 2005.

[51] G. D. Plotkin, A structural approach to operational semantics, DAIMI Report FN-19, Computer Science Department, Aarhus University (1981).

[52] J. Power, H. Watanabe, Distributivity for a monad and a comonad, in: Procs. CMCS'99, vol. 19 of ENTCS, Elsevier, 1999.

[53] J. J. M. M. Rutten, Universal coalgebra: a theory of systems, Theoretical Computer Science 249 (2000) 3–80.

[54] L. Schröder, Expressivity of coalgebraic modal logic: the limits and beyond, in: Proc. FOSSACS'05, vol. 3441 of LNCS, 2005.

[55] R. d. Simone, Higher-level synchronising devices in Meije-SCCS, Theoretical Computer Science 37 (1985) 245–267.

[56] A. Simpson, Sequent calculi for process verification: Hennessy-Milner logic for an arbitrary GSOS, Journal of Logic and Algebraic Programming 60–61 (2004) 287–322.

[57] D. Turi, G. D. Plotkin, Towards a mathematical operational semantics, in: Proc. LICS'97, IEEE Computer Society Press, 1997, pp. 280–291.