From Bialgebraic Semantics to Congruence Formats

Bartek Klin [1]

Abstract

A general and abstract framework to defining congruence formats for various process equivalences is presented. The framework extends bialgebraic techniques of Turi and Plotkin with an abstract coalgebraic approach to process equivalence, based on a notion of test suite. The resulting technique is illustrated on the example of completed trace equivalence. Rather than providing formal proofs, the paper is guiding the reader through the process of deriving a congruence format in the test suite approach.

Key words: process algebra, structural operational semantics, congruence format, bialgebraic semantics

1 Introduction

Process algebra is the area of research concerned with formal descriptions of complex computational systems, especially those with communicating, concurrently executing components. Since the 1980s it has been a well-established and intensively studied area of theoretical computer science (e.g. [4,8,18,21,29]). Traditionally, its main goal was to develop formalisms for the specification and verification of concurrent and networked computer systems, and to provide semantics for concurrent programming languages. Among the best-known traditional process algebras are ACP [7], CCS [28] and CSP [11].

More recently, methods of process algebra have been applied in other areas of computer science, in formal approaches to distributed, dynamic and mobile systems (e.g. π-calculus [14,30], calculus of mobile ambients [12]), security of cryptographic protocols (spi-calculus [1]) and even computational molecular biology [13,33].

When describing systems and processes formally, three key aspects must be considered: syntax, behaviour and process (also called behavioural, observational or operational) equivalence.

[1] Email: klin@brics.dk, WWW: http://www.brics.dk/ klin/

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

Syntax refers to the structure of processes, and reflects the fact that it is natural and convenient to describe various systems as composed of smaller subsystems. In simple cases, processes are arbitrary terms over an algebraic signature. More sophisticated syntactic phenomena include variable binding and structural congruences, where two syntactically different terms are identified and represent the same process.

Behaviour refers to the kind of actions the processes may take. In traditional process algebras, processes are allowed to nondeterministically perform externally observable actions from a prescribed set. One also considers deterministic, probabilistic and/or timed behaviour, the ability to perform unobservable actions, features related to state, input and output.

Process equivalence describes those processes whose behaviours should be considered "the same". This notion clearly depends on the chosen notion of behaviour of processes, but even for a single kind of behaviour one might consider many different process equivalences, suitable for different purposes.

The most well-established approach to the formal presentation of process algebras, covering all three aspects mentioned above, is that of *structural operational semantics* [32,2]. There, the behaviour of processes is modelled by means of transition relations on processes presented as terms over some signature. The transition relations in turn are induced by inference rules that follow the syntactic structure of processes. The intuitive appeal of this approach and, importantly, its inherent support for modelling nondeterministic behaviour, made it a natural framework for the formal description of process algebras.

Generally speaking, a set of operational inference rules induces a *transition system*, i.e., a set of processes together with a transition relation on it. Depending on the form of the rules, this can be a simple labelled transition system (LTS), an LTS with unobservable steps, a probabilistic transition system, a timed transition system etc. Based on the structure of the transition relation, one can define many different equivalences and preorders on processes. The variety of possible process equivalences has been studied most intensively in the case of LTSs, and includes bisimulation equivalence [31], simulation equivalence, trace equivalence, testing equivalence and many others (for a comprehensive treatment, see [17]). For other notions of transition system and process behaviour, the most thoroughly studied process equivalences include weak/delay/branching bisimulation equivalences (e.g. [16]), probabilistic bisimulation equivalence [26], timed bisimulation equivalence [3] and many others.

For a notion of process equivalence to be practically useful, it must be *compositional* (or, in other words, it must be a *congruence*), i.e., it must be respected by the syntactic constructs of a chosen process algebra. This is necessary for any kind of inductive reasoning about processes, for example for specification and verification of systems component by component.

Proofs of compositionality of chosen process equivalences with respect to particular process algebras can be quite demanding. It is therefore desirable to show general results of this kind that hold on entire classes of process algebras. In the framework of structural operational semantics, the search for such results led to the development of various *congruence formats*. A congruence format is a restriction on the syntactic form of structural inference rules that guarantees a particular process equivalence compositional.

One of the most popular congruence formats is GSOS [10], a restriction on the form of inference rules that guarantees bisimulation equivalence on the LTS generated from these rules to be compositional. Many other congruence formats have been defined for various notions of behaviour and process equivalence (see [2] and references therein, but also [5,23]). However, to define a general congruence format for a given notion of process equivalence is often a difficult task. Given the growing variety of disparate programming paradigms and process behaviours, it is desirable to have a general framework for constructing congruence formats for given notions of syntax, behaviour and process equivalence. To provide such a framework, one must come up with abstract notions of syntax, behaviour and equivalence, covering most of the well-known examples.

One such framework, developed by Turi and Plotkin in [35], is called *bialgebraic semantics*. It is based on the classical algebraic approach to syntax and a coalgebraic approach to behaviour [34], including a coalgebra span approach to bisimulation equivalence. In [35], it was shown how to derive, in an abstract fashion, GSOS as a congruence format for bisimulation equivalence. Applying the same approach to other kinds of behaviour, Bartels [6] derived a novel congruence format for probabilistic bisimulation equivalence on probabilistic transition systems, and Kick [23] a format for timed bisimulation on timed transition systems.

The generality of the original bialgebraic approach is limited by features of the coalgebra span approach to process equivalence, which covers only a single notion of equivalence for any given notion of behaviour. In [25,24], the bialgebraic approach has been modified to accommodate other well-known equivalences. Instead of the coalgebra span approach, a novel framework for modelling process equivalences was proposed, based on simple notions of tests and test suites. This, when combined with the original bialgebraic framework of Turi and Plotkin, gives a framework for deriving congruence formats for many different process equivalences.

The purpose of this paper is to explain, as intuitively as possible, the original bialgebraic techniques and their extension with test suites. To show some concrete examples, and to demonstrate that this abstract framework does lead to novel results, the paper concentrates on the commonly considered behaviour of labelled nondeterminism, and on two particular process equivalences: bisimulation equivalence and completed trace equivalence. The former is used to explain the original techniques of Turi and Plotkin, and the latter to

illustrate the test suite approach. The choice of completed trace equivalence is motivated by the opinion (see [2]) that it is very difficult to find a general congruence format for it. The successful derivation of a reasonably general format therefore serves as evidence that the abstract bialgebraic approach can bring some concrete and useful results.

The content and style of this paper is motivated by the feeling, expressed to me by several people, that the bialgebraic test suite approach is complicated and hard to understand. In particular, formal proofs of correctness of derived congruence formats shown in [24] are admittedly not very revealing, and they provide little insight into the process of actual derivation of formats. This makes it a bit hard for anyone to adapt that approach to their favourite behaviours and equivalences.

For this reason, I decided to omit almost every formal proof from this presentation of bialgebraic and test suite techniques. Instead, I tried to provide as much intuition as possible into each step of the derivation of a congruence format for completed trace equivalence. I hope that this intuition will make the test suite approach easier to understand and to use. For a thorough, detailed and fully formal presentation of the ideas sketched here, the reader is advised to refer to [24]. A preliminary version of the framework and of the format presented here was also published as joint work with Paweł Sobociński in [25].

Although all relevant definitions are recalled, the reader is assumed to have basic knowledge of structural operational semantics and concurrency theory. Being familiar with [17] and [2] should be more than enough. Throughout the paper, basic notions of category theory are used. The reader is assumed to be familiar with rudimentary notions like category, functor, natural transformation, initial object, final object, product, coproduct and pullback. The first chapters of [27], along many other books, contain definitions of these.

The structure of the paper is as follows. In Sections 2 and 3, some notions and results from the area of structural operational semantics are briefly recalled. In Section 4, the original bialgebraic approach of Turi and Plotkin is sketched, together with its abstract representation of the GSOS format. In Section 5, the abstract test suite approach to various process equivalences is described. In Section 6, this approach is combined with bialgebraic methods to give an abstract theorem about congruence formats. That theorem is specialised to the case of completed trace equivalence, and the reader is guided through the process of derivation of a congruence format for the equivalence. The paper is concluded with a set of exercises, and a list of open problems worth pondering.

Acknowledgements: I am grateful to Piotr Hoffman, Paweł Sobociński and Daniele Varacca for comments on a draft of this paper.

2 Labelled transition systems and process equivalences

Operational descriptions are usually based on the notion of labelled transition system.

Definition 2.1 A *labelled transition system* (LTS) $\langle X, A, \longrightarrow \rangle$ is a set X of *processes*, a set A of *actions*, and a *transition relation* $\longrightarrow \subseteq X \times A \times X$. Usually instead of $\langle x, a, x' \rangle \in \longrightarrow$ one writes $x \stackrel{a}{\longrightarrow} x'$. An LTS $\langle X, A, \longrightarrow \rangle$ is called *finitely branching* if for every process $x \in X$ there are only finitely many processes $x' \in X$ and actions $a \in A$ such that $x \stackrel{a}{\longrightarrow} x'$.

For any $x \in X$, $a \in A$ one writes $x \stackrel{a}{\not\longrightarrow}$ to denote that there is no $x' \in X$ such that $x \stackrel{a}{\longrightarrow} x'$. Moreover, $x \not\longrightarrow$ means that $x \stackrel{a}{\not\longrightarrow}$ for all $a \in A$.

Many preorders and equivalences have been defined on processes in labelled transition systems. For illustration purposes, this paper concentrates on two of them: bisimulation equivalence and completed trace equivalence. Many other equivalences are studied in detail in [17].

Bisimulation equivalence is usually defined based on the classical notion of bisimulation. In the following two definitions, an LTS $\langle X, A, \longrightarrow \rangle$ is assumed.

Definition 2.2 A relation $R \subseteq X \times X$ is a *bisimulation* if xRy implies that for any $a \in A$,

- for any $x' \in X$ if $x \stackrel{a}{\longrightarrow} x'$ then there exists $y' \in X$ such that $y \stackrel{a}{\longrightarrow} y'$ and $x'Ry'$, and
- for any $y' \in X$ if $y \stackrel{a}{\longrightarrow} y'$ then there exists $x' \in X$ such that $x \stackrel{a}{\longrightarrow} x'$ and $x'Ry'$.

Processes $x, y \in X$ are *bisimulation equivalent*, or *bisimilar*, if there exists a bisimulation R such that xRy.

It is straightforward to prove that in any LTS bisimulation equivalence is indeed an equivalence relation, and it is the greatest bisimulation on the LTS.

For finitely branching LTSs, an alternative characterisation of bisimulation can be given using the following *finitary Hennessy-Milner logic*. Processes are considered equivalent if and only if they satisfy exactly the same formulae from the logic.

Definition 2.3 Given a set of actions A, consider the set of *modal formulae* $\mathcal{F}_{\mathsf{BS}}$, given by the BNF grammar:

$$\phi ::= \top \mid \bot \mid \langle a \rangle \, \phi \mid [a]\phi \mid \phi \wedge \phi \mid \phi \vee \phi$$

where a ranges over A. Given an LTS $h = \langle X, A, \longrightarrow \rangle$, the satisfaction relation $\models_h \subseteq X \times \mathcal{F}_{\mathsf{BS}}$ is defined inductively as follows:

$$\begin{array}{lll}
x \models_h \top & & \text{always} \\
x \models_h \bot & & \text{never} \\
x \models_h \langle a \rangle \phi & \iff & x' \models_h \phi \text{ for some } x' \text{ such that } x \xrightarrow{a} x' \\
x \models_h [a]\phi & \iff & x' \models_h \phi \text{ for all } x' \text{ such that } x \xrightarrow{a} x' \\
x \models_h \phi_1 \wedge \phi_2 & \iff & x \models_h \phi_1 \text{ and } x \models_h \phi_2 \\
x \models_h \phi_1 \vee \phi_2 & \iff & x \models_h \phi_1 \text{ or } x \models_h \phi_2
\end{array}$$

Finally, the equivalence $\cong_{\mathsf{BS}} \subseteq X \times X$ is defined on a given LTS h by:

$$x \cong_{\mathsf{BS}} x' \iff (\forall \phi \in \mathcal{F}_{\mathsf{BS}}.\ x \models_h \phi \iff x' \models_h \phi)$$

The soundness and completeness results for the Hennessy-Milner logic [19] say that in any finitely branching LTS, the relation $\cong_{\mathsf{BS}}$ is equal to bisimulation equivalence.

Once the Hennessy-Milner logic has been recalled, the easiest way to define other well-known equivalences on LTSs is via suitable fragments of that logic. This is how we proceed to define completed trace equivalence.

Definition 2.4 Given a set of actions A, consider the set of modal formulae $\mathcal{F}_{\mathsf{CTr}}$, given by the BNF grammar:

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

where a ranges over A. Given an LTS $h = \langle X, A, \longrightarrow \rangle$, the satisfaction relation $\models_h \subseteq X \times \mathcal{F}_{\mathsf{CTr}}$ is as in Definition 2.3, with the additional case:

$$x \models_h \varnothing \iff x \not\longrightarrow$$

The equivalence $\cong_{\mathsf{CTr}}$ on X is defined on a given LTS h by:

$$x \cong_{\mathsf{CTr}} x' \iff (\forall \phi \in \mathcal{F}_{\mathsf{CTr}}.\ x \models_h \phi \iff x' \models_h \phi) \tag{2.1}$$

The above logic can be indeed viewed as a fragment of the Hennessy-Milner logic, since in any LTS, a process satisfies the formula $\varnothing$ if and only if it satisfies formulae $[a]\bot$ for all actions a. Formulae in $\mathcal{F}_{\mathsf{CTr}}$ ending with $\bot$ are usually called finite, partial traces (or simply traces), and formulae ending with $\varnothing$ are called completed traces.

3 Structural operational semantics and congruence formats

In the context of structural operational semantics, processes are usually closed terms over some signature and labelled transition systems are induced from sets of inference rules.

A *signature* Σ is a set $\bar{\Sigma}$ of *language constructs*, together with an *arity function* $ar : \bar{\Sigma} \to \mathbb{N}$. For a given set X of *variables*, ΣX is the set of expressions of the form $\mathtt{f}(x_1, \ldots, x_{ar(\mathtt{f})})$, where $\mathtt{f} \in \bar{\Sigma}$ and $x_1, \ldots, x_{ar(\mathtt{f})} \in X$.

Given a signature Σ and a set X, the set of *terms* over Σ with variables X is denoted $T_\Sigma X$. The subscript in $T_\Sigma X$ is omitted if Σ is irrelevant or clear from the context. Elements of $T0$ (throughout this paper, 0 denotes the empty set) are called *closed terms* over Σ.

For a term $t \in TX$ and a function (substitution) $\sigma : X \to Y$, $t\sigma$ denotes the term in TY resulting from t by simultaneously replacing every $x \in X$ with $\sigma(x)$.

To define inference rules, assume a fixed, countably infinite set of variables Ξ, ranged over by $\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{y}_1, \mathtt{y}_2, \ldots$. Terms built over variables from Ξ are typeset $\mathtt{t}$, $\mathtt{t}'$ etc., as opposed to the normal notation t, t' etc.

In the following, fix an arbitrary set of actions A. For a signature Σ, a *positive* Σ*-literal* is an expression $\mathtt{t} \xrightarrow{a} \mathtt{t}'$, and a *negative* Σ*-literal* is an expression $\mathtt{t} \not\xrightarrow{a}$, where $\mathtt{t}, \mathtt{t}' \in T\Xi$ and $a \in A$. An *inference rule* ρ over Σ is an expression $\frac{H}{\alpha}$, where H is a set of Σ-literals and α is a positive Σ-literal. Elements of H are then called *premises* of ρ, and α the *conclusion* of ρ. The left side and the right side of the conclusion of ρ are called the *source* and the *target* of ρ, respectively. If the source of a rule ρ is of the form $\mathtt{f}(\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_n)$, one says that ρ is a rule *for* $\mathtt{f}$.

A *transition system specification* over Σ is a set of rules over Σ.

Example 3.1 An example of a transition system specification is that of *basic process algebra*. Assuming a set A of actions, its syntax Σ is defined by the BNF grammar

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

and the transition system specification **BPA** over Σ is a collection of rules

$$\frac{}{\alpha\mathtt{x} \xrightarrow{\alpha} \mathtt{x}} \qquad \frac{\mathtt{x} \xrightarrow{\alpha} \mathtt{x}'}{\mathtt{x}+\mathtt{y} \xrightarrow{\alpha} \mathtt{x}'} \qquad \frac{\mathtt{y} \xrightarrow{\alpha} \mathtt{y}'}{\mathtt{x}+\mathtt{y} \xrightarrow{\alpha} \mathtt{y}'}$$

where α ranges over A. When presenting terms over the above syntax, the trailing $\mathtt{nil}$s are omitted. This is a rather simple example; in particular, all premises in the above rules are positive.

It is quite clear how to induce a labelled transition system from the above rules. First, a notion of a provable positive literal is defined in a straightforward way. The set of all provable literals forms an LTS with closed terms

over Σ as processes, and with positive closed literals as transitions. Note that when negative premises are around, it might be less clear how to associate an LTS to a transition system specification in a meaningful way (see [2]).

If an LTS can be induced from a set of rules over Σ, one can define various equivalences (e.g., bisimulation equivalence, completed trace equivalence) between closed terms over Σ as the corresponding equivalences on the induced LTS. Such equivalences identify terms whose behaviours as processes are the same in some sense. However, for an equivalence to be useful in practice, it should be a congruence:

Definition 3.2 Let Σ be any signature. An equivalence relation $R \subseteq T0 \times T0$ is a *congruence*, if for any $\mathtt{f} \in \bar{\Sigma}$ with $ar(\mathtt{f}) = n$, and for any $t_1, t'_1, t_2, t'_2, \ldots, t_n, t'_n \in T0$, whenever $t_i R t'_i$ $(i = 1, 2, \ldots, n)$ then $\mathtt{f}(t_1, t_2, \ldots, t_n) R \mathtt{f}(t'_1, t'_2, \ldots, t'_n)$.

As it turns out, many interesting equivalences are not congruences even for quite simple languages. Moreover, if a particular equivalence is a congruence for a given language, the proof of this is can be quite demanding. This is why it is worthwhile to look for *congruence formats* of transition system specifications, i.e., syntactic restrictions on sets of rules that guarantee particular equivalences on the induced LTSs to be congruences.

One of the most popular congruence format is GSOS [10]. The following definition assumes a fixed signature Σ.

Definition 3.3 [GSOS]A transition system specification Λ is in GSOS format if every rule $\rho \in \Lambda$ is of the form

$$\frac{\left\{ \mathtt{x}_i \xrightarrow{a_{ij}} \mathtt{y}_{ij} \ : \ i \leq n, \ j \leq m_i \right\} \cup \left\{ \mathtt{x}_i \overset{b_{ik}}{\not\longrightarrow} \ : \ i \leq n, k \leq n_i \right\}}{\mathtt{f}(\mathtt{x}_1, \ldots, \mathtt{x}_n) \xrightarrow{c} \mathtt{t}}$$

with $\mathtt{f}$ a language construct in $\bar{\Sigma}$ and $n = ar(\mathtt{f})$, such that $\mathtt{x}_i \in \Xi$ and $\mathtt{y}_{ij} \in \Xi$ are all distinct and are the only variables that occur in ρ. If, moreover, for every $\mathtt{f} \in \bar{\Sigma}$, Λ contains only finitely many rules for $\mathtt{f}$, then Λ is *image-finite*.

Any transition system specification Λ in GSOS format induces an LTS as sketched above. If Λ is image-finite, then the induced LTS is finitely branching (for details, see [2]). Moreover, as was proved in [10], the bisimulation equivalence $\cong_{\mathsf{BS}}$ is guaranteed to be a congruence on the induced LTS. In other words, GSOS is a congruence format for bisimulation equivalence.

Several other congruence formats for bisimulation equivalence and for other well-known equivalences have been defined (for a detailed survey, see [2]). However, for some time the task of defining a general format for completed trace equivalence appeared very difficult. That view was supported by the following examples of seemingly simple and innocent rules, for which, however, completed trace equivalence fails to be a congruence.

Example 3.4 Assume $A = \{a, b\}$, and extend **BPA** with an operational rule for the *encapsulation operator* $\partial_{\{b\}}$:

$$\frac{\mathtt{x} \xrightarrow{a} \mathtt{y}}{\partial_{\{b\}}(\mathtt{x}) \xrightarrow{a} \partial_{\{b\}}(\mathtt{y})}$$

It is easy to check that the above extension of **BPA** does not respect completed traces. Indeed, $aa+ab \cong_{\mathsf{CTr}} a(a+b)$ but $\partial_{\{b\}}(aa+ab) \not\sqsubseteq_{\mathsf{CTr}} \partial_{\{b\}}(a(a+b))$, since $\langle a \rangle \varnothing$ is a completed trace of $\partial_{\{b\}}(aa+ab)$, but not of $\partial_{\{b\}}(a(a+b))$.

Example 3.5 Assume $A = \{a, b\}$, and extend **BPA** with a collection of rules for binary *synchronous composition* $\times$:

$$\frac{\mathtt{x} \xrightarrow{\alpha} \mathtt{x}' \quad \mathtt{y} \xrightarrow{\alpha} \mathtt{y}'}{\mathtt{x} \times \mathtt{y} \xrightarrow{\alpha} \mathtt{x}' \times \mathtt{y}'}$$

where α ranges over A.

Here it is easy to see that $aa \times (aa + ab) \not\sqsubseteq_{\mathsf{CTr}} aa \times a(a+b)$, since $\langle a \rangle \varnothing$ is a completed trace of $aa \times (aa + ab)$, but not of $aa \times (a(a+b))$.

However, completed trace equivalence turns out to be a congruence for **BPA**, and even for some seemingly more complicated sets of rules, involving in particular negative premises:

Example 3.6 Extend **BPA** with a collection of rules for binary *sequential composition* ;:

$$\frac{\mathtt{x} \xrightarrow{\alpha} \mathtt{x}'}{\mathtt{x};\mathtt{y} \xrightarrow{\alpha} \mathtt{x}';\mathtt{y}} \qquad \frac{\mathtt{x} \not\xrightarrow{a} \text{ for all } a \in A \quad \mathtt{y} \xrightarrow{\alpha} \mathtt{y}'}{\mathtt{x};\mathtt{y} \xrightarrow{\alpha} \mathtt{y}'}$$

where α ranges over A.

The proof that completed trace equivalence is a congruence for this language is left as an exercise.

The above examples are a bit discouraging. It seems hard to identify a syntactic feature of the rules for encapsulation and synchronous composition that prevents completed trace equivalence from being a congruence, as these rules are really quite simple. However, in the following sections a congruence format will be derived that excludes these examples, and still is general enough to cover, e.g., the sequential composition operator.

In fact, this paper takes upon a bolder task and provides a method of deriving congruence formats for any given notion of equivalence with enough structure. The new format for completed trace equivalence appears as a special case of this general approach. To do this, one needs to look at labelled transition systems, operational equivalences and transition system specifications a bit more abstractly. Basic category theory proves a useful tool to this end.

4 Towards a mathematical operational semantics

We begin with a brief presentation of the beautiful abstract framework developed by Turi and Plotkin [35] and called *bialgebraic semantics* or *abstract GSOS*. It allowed them to redefine GSOS format and re-prove its congruence properties for bisimulation equivalence in an abstract fashion. In the following sections, that approach is modified to cover other process equivalences, in particular completed trace equivalence.

Any labelled transition system $\langle X, A, \longrightarrow \rangle$ can be viewed as a function

$$h : X \longrightarrow \mathcal{P}(A \times X)$$

(where $\mathcal{P}$ is the powerset construction), via the easy correspondence

$$\langle a, y \rangle \in h(x) \iff x \xrightarrow{a} y$$

With $\mathcal{P}$ viewed as the covariant powerset endofunctor on the category **Set** of sets and functions, a function as above is called a *coalgebra* for the functor $\mathcal{P}(A\times -)$, or $\mathcal{P}(A\times -)$-coalgebra in short. It turns out that replacing $\mathcal{P}(A\times -)$ with other endofunctors (called *behaviour endofunctors* in this context) one can model different kinds of systems, including deterministic, probabilistic, timed, ones with state, input and output and many others. This has led to the development of universal coalgebra as an abstract theory of transition systems (for a detailed introduction, see [34]).

Formally, for any endofunctor B on **Set**, a B-coalgebra with carrier X is a function $h : X \to BX$. A coalgebra morphism from $g : X \to BX$ to $h : Y \to BY$ is a function $f : X \to Y$ such that the following diagram commutes:

$$\begin{array}{ccc} X & \xrightarrow{f} & Y \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle h} \\ BX & \xrightarrow[Bf]{} & BY \end{array}$$

It is easy to see that B-coalgebras and their morphisms form a category, with identities and composition inherited from **Set**.

One of the most important achievements of the coalgebraic approach to processes is the following abstract definition of bisimulation:

Definition 4.1 Let $h : X \to BX$ be a coalgebra. A *(span) bisimulation* on h is a relation $R \subseteq X \times X$ such that there exists a coalgebra structure $r : R \to BR$ that makes the projections π_1 and π_2 into coalgebra morphisms:

$$\begin{array}{ccccc} X & \xleftarrow{\pi_1} & R & \xrightarrow{\pi_2} & X \\ {\scriptstyle h}\downarrow & & \downarrow{\scriptstyle r} & & \downarrow{\scriptstyle h} \\ BX & \xleftarrow[B\pi_1]{} & BR & \xrightarrow[B\pi_2]{} & BX \end{array}$$

It turns out that for $B = \mathcal{P}(A \times -)$, this definition specialises to the standard definition of bisimulation on LTSs. For other choices of B, it specialises to other well-known definitions (for details see [34]).

In many technical developments of the coalgebraic approach it is required that the chosen behaviour functor B preserves weak pullbacks and that in the category of B-coalgebras a final object (a final B-coalgebra) exists. Unfortunately, the endofunctor $\mathcal{P}(A \times -)$ does not admit final coalgebras. Therefore, to use the full power of the coalgebraic approach, it is convenient to restrict attention to finitely branching LTSs. It is easy to see that such systems correspond to coalgebras for the endofunctor $\mathcal{P}_{\mathrm{f}}(A \times -)$, where $\mathcal{P}_{\mathrm{f}}$ is the covariant *finite* powerset functor. This behaviour functor admits final coalgebras and preserves weak pullbacks (for details, see [34]), and the following developments apply without any further assumptions.

The next step is to equip carriers of coalgebras with some kind of syntactic structure, since in the context of structural operational semantics processes in LTSs are terms over some syntax.

A common technique to represent syntax of simple languages is to represent signatures as polynomial (i.e., built only of constants, sums and products) endofunctors on **Set**. Any signature Σ determines an endofunctor (somewhat sloppily denoted Σ as well, which should not lead to any confusion) defined by

$$\Sigma X = \coprod_{\mathtt{f} \in \bar{\Sigma}} \prod_{ar(\mathtt{f})} X$$

where $\coprod$ and $\prod$ denote disjoint unions and cartesian products, respectively. For example, the signature of the basic process algebra **BPA** based on a set of actions A corresponds to the endofunctor

$$\Sigma X = 1 + \underbrace{(X + \ldots + X)}_{|A| \text{ times}} + X \times X$$

where 1 is a singleton set, $+$ denotes disjoint union and $\times$ denotes cartesian product. Then, to equip a set X with algebraic structure over the signature is just to provide a function

$$g : \Sigma X \longrightarrow X$$

which, in categorical terms, is called a Σ-*algebra*. A morphism from an algebra $g : \Sigma X \to X$ to an algebra $h : \Sigma Y \to Y$ is a function $f : X \to Y$ such that the following diagram commutes:

$$\begin{array}{ccc} \Sigma X & \xrightarrow{\Sigma f} & \Sigma Y \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle h} \\ X & \xrightarrow[f]{} & Y \end{array}$$

As in the case of coalgebras, Σ-algebras and their morphisms form a category. If Σ is polynomial (as it is the case when it is derived from a signature

as above), then the initial Σ-algebra

$$\psi : \Sigma T0 \longrightarrow T0$$

always exists, where $T0$ is the set of all closed terms over Σ, and ψ is the obvious algebraic structure "gluing" terms.

The notion of congruence from Definition 3.2 can be generalised to arbitrary Σ-algebras for endofunctors Σ obtained from signatures.

Definition 4.2 Let Σ be a signature and $h : \Sigma X \to X$ an algebra for the corresponding polynomial endofunctor on **Set**. An equivalence relation $R \subseteq X \times X$ is a *congruence on* h if for any coproduct injection $\mathtt{f} : X^n \to \Sigma X$, and for any $x_1, y_1, x_2, y_2, \ldots, x_n, y_n \in X$, whenever $x_i R y_i$ $(i = 1, 2, \ldots, n)$ then $h(\mathtt{f}\,\langle x_1, x_2, \ldots, x_n\rangle) R h(\mathtt{f}\,\langle y_1, y_2, \ldots, y_n\rangle)$.

It is easy to see that when h is the initial Σ-algebra $\psi : \Sigma T0 \to T0$, the above definition specialises to Definition 3.2.

The algebraic framework allows one to represent terms over a signature as well. Formally, given any set X, the endofunctor $X + \Sigma -$ admits initial algebras, and set of terms over Σ with variables from X is the carrier of an initial $(X + \Sigma -)$-algebra. This construction (from X to that carrier) can be extended to an endofunctor on **Set**, denoted T_Σ, or T in short, if Σ is irrelevant or clear from context. In particular, the set $T0$ of closed terms over Σ is the carrier of an initial Σ-algebra. In categorical terminology, the functor T is called the *monad freely generated by* Σ.

In [35], Turi and Plotkin suggested the notion of bialgebra as an abstract representation of transition systems with syntactic structure imposed on processes. A *bialgebra* for functors Σ, B on **Set** is simply a pair of a Σ-algebra and a B-coalgebra with a common carrier:

$$\Sigma X \xrightarrow{g} X \xrightarrow{h} BX$$

They then showed how to induce bialgebras for given Σ and B from *distributive laws*, i.e., natural transformations of the form

$$\lambda : \Sigma(\mathrm{Id} \times B) \Longrightarrow BT$$

where T is the monad freely generated by Σ. For purposes of this paper it is not really important how this abstract construction works technically (for details, see [35]); suffice it to say that any λ as above gives rise to a certain bialgebra with carrier $T0$:

$$\Sigma T0 \xrightarrow{\psi} T0 \xrightarrow{h_\lambda} BT0$$

where ψ is the initial Σ-algebra, and h_λ is a coalgebraic structure derived from λ. Note that the carrier of h_λ is the set of closed terms over the signature Σ.

Recall that when $B = \mathcal{P}_{\mathrm{f}}(A \times -)$, then h_λ is just a finitely branching LTS. Thus one has a method of deriving LTSs from certain abstract entities: distributive laws. A central result of [35] states that these distributive laws in fact correspond to transition system specifications in image-finite GSOS format, and the abstract construction of h_λ from λ specialises to the induction of LTSs from such specifications:

Theorem 4.3 There is an (essentially 1-1) correspondence between transition system specifications Λ in image-finite GSOS format over a signature Σ and with a set of actions A, and natural transformations

$$\lambda : \Sigma(\mathrm{Id} \times \mathcal{P}_{\mathrm{f}}(A \times -)) \Longrightarrow \mathcal{P}_{\mathrm{f}}(A \times T-)$$

For any such transition system specification Λ, the coalgebra h_λ induced from the natural transformation λ corresponding to Λ is just the LTS induced from Λ.

Proof The full proof of this theorem was left implicit in [35], but can be found in [6]. Here it is enough to give some intuition on how to derive a natural transformation λ from a set of rules Λ.

Fix an arbitrary set of actions A, a signature Σ with its corresponding polynomial endofunctor, and an image-finite set Λ of GSOS rules over Σ with actions from A. The task is to define a transformation λ as in the theorem statement.

For a given set X, the domain of the component function λ_X is the set

$$\Sigma(X \times \mathcal{P}_{\mathrm{f}}(A \times X))$$

Elements of this set are terms r built of a single, principal language construct (say, $\mathtt{f}$) only, with variables from X, and with each variable x equipped with a finite set of pairs that can be viewed as possible transitions originating from x. Given such information, one might determine what transitions can be made from r according to rules Λ, by looking on each rule ρ for $\mathtt{f}$ in Λ in turn, and trying to substitute elements of X for variables from Ξ so that variables the in source of ρ are mapped to variables in r, and that all premises of ρ are satisfied by sets of transitions provided with r. If this can be done, than ρ can be "fired" and the resulting transition (i.e., a pair of a label and a term) is added to the result of λ_X, which is an element of

$$\mathcal{P}_{\mathrm{f}}(A \times TX)$$

More formally, given a language construct $\mathtt{f} \in \bar{\Sigma}$ with arity n and a set X, a rule ρ in the GSOS format

$$\frac{\left\{ \mathtt{x}_i \xrightarrow{a_{ij}} \mathtt{y}_{ij} \ : \ i \leq n, \ j \leq m_i \right\} \cup \left\{ \mathtt{x}_i \not\xrightarrow{b_{ik}} \ : \ i \leq n, k \leq n_i \right\}}{\mathtt{f}(\mathtt{x}_1, \ldots, \mathtt{x}_n) \xrightarrow{c} \mathtt{t}}$$

defines a map $\rho_X : (X \times \mathcal{P}_{\mathrm{f}}(A \times X))^n \to \mathcal{P}_{\mathrm{f}}(A \times TX)$ as follows: $\langle c, t\rangle \in \rho_X \langle x_i, \beta_i\rangle_{i\leq n}$ iff there exists a substitution $\sigma : \Xi \to X$ satisfying

(i) $\sigma(\mathtt{x_i}) = x_i$

(ii) $\forall i \leq n \forall j \leq m_i \ \langle a_{ij}, \sigma(\mathtt{y_{ij}})\rangle \in \beta_i$

(iii) $\forall i \leq n \forall k \leq n_i \forall x \in X \ \langle b_{ij}, x\rangle \notin \beta_i$

(iv) $\mathtt{t}\sigma = t$

Then given a set Λ of rules in the image finite GSOS format one can define a function $\lambda_X : \Sigma(X \times \mathcal{P}_{\mathrm{f}}(A \times X)) \to \mathcal{P}_{\mathrm{f}}(A \times TX)$ by defining for each $\mathtt{f} \in \bar{\Sigma}$ with arity n a function $f_X : (X \times \mathcal{P}_{\mathrm{f}}(A \times X))^n \to \mathcal{P}_{\mathrm{f}}(A \times TX)$ as follows:

$$f_X : \langle x_i, \beta_i\rangle_{i\leq n} \quad \mapsto \quad \bigcup_{\substack{\rho\in\Lambda \\ \rho \text{ a rule for } \mathtt{f}}} \rho_X \langle x_i, \beta_i\rangle_{i\leq n}$$

Image finiteness of Λ ensures that this function is well defined, i.e. that it returns only finite sets. Finally, λ_X is determined uniquely by the f_X's since it is a function from a coproduct. □

For example, consider the transition system specification **BPA** (Example 3.1) as Λ (assuming $a, b, c \in A$), and take $X = \{\spadesuit, \heartsuit, \diamondsuit, \clubsuit\}$. Consider the following $r_1, r_2, r_3 \in \Sigma(X \times \mathcal{P}_{\mathrm{f}}(A \times X))$:

$$\begin{aligned} r_1 &= \langle \clubsuit, \{\langle a, \heartsuit\rangle, \langle b, \spadesuit\rangle\}\rangle + \langle \diamondsuit, \{\langle a, \clubsuit\rangle\}\rangle \\ r_2 &= a\,\langle \heartsuit, \{\langle b, \diamondsuit\rangle, \langle c, \spadesuit\rangle\}\rangle \\ r_3 &= \mathtt{nil} \end{aligned}$$

Then the construction described above yields

$$\begin{aligned} \lambda_X r_1 &= \{\langle a, \heartsuit\rangle, \langle b, \spadesuit\rangle, \langle a, \clubsuit\rangle\} \\ \lambda_X r_2 &= \{\langle a, \heartsuit\rangle\} \\ \lambda_X r_3 &= \emptyset \end{aligned}$$

In [35] it was proved that the congruence property of GSOS is a special case of the following result:

Theorem 4.4 For any natural transformation

$$\lambda : \Sigma(\mathrm{Id} \times B) \Longrightarrow BT$$

(where B admits final coalgebras and preserves weak pullbacks) the largest span bisimulation on $h_\lambda : T0 \to BT0$ is a congruence relation on $T0$.

This, specialised to the case of $B = \mathcal{P}_{\mathrm{f}}(A \times -)$ and combined with Theorem 4.3, means that image-finite GSOS is a congruence format for bisimu-

lation equivalence. The importance of Theorem 4.4, however, is that it can be applied to other behaviour endofunctors B and thus it serves as a general method of deriving congruence formats for equivalences corresponding to span bisimulations for various behaviours. This method has already been applied to probabilistic transition systems in [6], yielding a congruence format for probabilistic bisimulation, and to timed transition systems in [23], where a congruence format for timed bisimulation was obtained.

The generality of this approach, however, is limited by the scope of the span bisimulation approach to process equivalence. For any behaviour endofunctor, only a single, canonical notion of equivalence based on span bisimulations can be covered. One way to circumvent this is to find another, more general coalgebraic approach of process equivalence, that in the case of ordinary LTSs would cover more notions than just bisimulation equivalence (in particular, completed trace equivalence), but still allows one to prove a counterpart of Theorem 4.4 to obtain congruence properties.

5 Test suites

In this section, an alternative to the coalgebra span approach to process equivalence is presented, based on simple notions of tests and test suites. An abstract definition of process equivalence is shown, which specialises to most well-known equivalences on LTSs, in particular to completed trace equivalence. In Section 6 this will be combined with bialgebraic methods, yielding a method of deriving congruence formats for given notions of equivalence.

Conceptually, the test suite approach is based on intuitions taken from modal logic: two processes are considered equivalent if they cannot be distinguished by any test from a given test suite. Varying the test suites considered, one obtains different notions of equivalence. The relevant test suites are obtained from the coalgebraic structure of the LTS in question.

The test suite approach departs from the coalgebra span approach in that to find an equivalence on a coalgebra h, one does not construct a span of coalgebras, but rather enriches h with additional structure and requires h to preserve that structure. This general approach was used in [20,22], where coalgebras were equipped with binary relations on their carriers. In our approach, instead, coalgebras are equipped with test suites.

From now on, denote $2 = \{\mathtt{tt}, \mathtt{ff}\}$. A *test* on a set X is a function $V : X \to 2$. A *test suite* on a set X (denoted $\theta : X \rightrightarrows 2$) is a set of tests on X. The set of all test suites on a set X, partially ordered by inclusion, is denoted X^*.

Obviously, any test V on X can be identified with the following subset of X:

$$\{\, x \in X \,:\, Vx = \mathtt{tt} \,\}$$

Viewed this way, a test suite on X is a family of subsets of X. In particular, any topology is a test suite on its carrier. It is useful to keep the intuitive

set interpretation in mind, but stick to the functional definition for technical convenience in further developments.

The *category of test suites* **TS** is defined as follows:

- objects in **TS** are pairs $\langle X, \theta \rangle$, where X is a set and $\theta : X \rightrightarrows 2$ is a test suite,
- morphisms $f : \langle X, \theta \rangle \to \langle Y, \vartheta \rangle$ are functions $f : X \to Y$ such that

$$\{ V \circ f \ : \ V \in \vartheta \} \subseteq \theta$$

To get some intuition, note that morphisms in **TS** are defined exactly like continuous morphisms between topologies, when topologies are viewed as test suites.

T and F denote the constantly true and the constantly false tests. One also speaks of unions and intersections of tests, denoted with $\vee$ and $\wedge$ and defined in the obvious way.

In Section 6, categorical products and coproducts in **TS** will be used. They are defined as follows:

$$\langle X, \theta \rangle + \langle Y, \vartheta \rangle = \langle X + Y, \theta \oplus \vartheta \rangle$$
$$\langle X, \theta \rangle \times \langle Y, \vartheta \rangle = \langle X \times Y, \theta \otimes \vartheta \rangle$$

where

$$\theta \oplus \vartheta = \{ [V, V'] \ : \ V \in \theta, V' \in \vartheta \}$$
$$\theta \otimes \vartheta = \{ V \circ \pi_1 \ : \ V \in \theta \} \cup \{ V' \circ \pi_2 \ : \ V' \in \vartheta \}$$

(here $[V, V'] : X + Y \to 2$ denotes the copairing of V and V'). Checking the required universal properties is left as an exercise. It will also be useful to consider another construction on test suites. For test suites $\theta : X \rightrightarrows 2$, $\vartheta : Y \rightrightarrows 2$, the test suite $\theta \bowtie \vartheta : X \times Y \rightrightarrows 2$ is defined by

$$\theta \bowtie \vartheta = \{ \wedge \circ (V \times V') \ : \ V \in \theta, V' \in \vartheta \} \tag{5.1}$$

where $\wedge : 2 \times 2 \to 2$ is the logical-and operator. It is easy to see that $\bowtie$ is associative, therefore parentheses around its use are omitted when appropriate.

Intuitively, given test suites $\theta : X \rightrightarrows 2$ and $\vartheta : Y \rightrightarrows 2$, $\theta \oplus \vartheta$ is the test suite on $X + Y$ containing tests defined by cases using tests from θ on X and ϑ on Y, $\theta \otimes \vartheta$ contains the tests on $X \times Y$ which consist of *either* a test from θ on X *or* a test from θ on Y; finally, $\theta \bowtie \vartheta$ is the test suite on $X \times Y$ consisting of tests built by performing a test from θ on X and simultaneously performing a test from ϑ on Y and accepting when *both* tests accept.

Let θ be a test suite on any set X. *Specialisation equivalence* $\cong_\theta$ on X is defined by

$$\cong_\theta = \{ \langle x, y \rangle \in X \times X \mid \forall V \in \theta. \ Vx = Vy \}$$

It is straightforward, and left as an exercise, to show that test suite morphisms preserve specialisation equivalences.

To equip coalgebras for an endofunctor B on **Set** with test suites, it is useful to enrich B to act on the category **TS**. One says that an endofunctor $B^* : \mathbf{TS} \to \mathbf{TS}$ *lifts* an endofunctor $B : \mathbf{Set} \to \mathbf{Set}$ if $p \circ B^* = B \circ p$, where $p : \mathbf{TS} \to \mathbf{Set}$ is the obvious forgetful functor.

Endofunctors on **TS** lifting a given functor $B : \mathbf{Set} \to \mathbf{Set}$ are determined by their *actions*, i.e., families of functions $\{ B_X : X^* \to (BX)^* \ : \ X \in \mathbf{Set} \}$. If such a family satisfies a certain mild condition (see [24] for details) then B^* defined by

$$B^* \langle X, \theta \rangle = \langle BX, B_X \theta \rangle \qquad B^* f = Bf$$

is an endofunctor on **TS**, and it obviously lifts B. In the following, expressions like B_X, Σ_X will denote actions of some corresponding endofunctors B^*, Σ^* on **TS**.

Any functor $B : \mathbf{Set} \to \mathbf{Set}$ can be lifted to an endofunctor on **TS** in possibly many ways, by defining its action B_X on test suites. For our purposes, one particular way is especially useful. This well-structured method of lifting endofunctors is based on notions of test constructors and closures. Intuitively speaking, one might view tests as formulae interpreted on processes. Then test constructors correspond to modal operators in a language of formulae, and closures correspond to propositional connectives.

Definition 5.1 Let B be an endofunctor on **Set**. Tests on the set $B2$ (i.e, functions $w : B2 \to 2$) are called *B-test constructors*, or simply test constructors, if B is irrelevant or clear from context.

Example 5.2 In the running example for this paper, $B2 = \mathcal{P}_{\mathrm{f}}(A \times 2)$. A test on $\mathcal{P}_{\mathrm{f}}(A \times 2)$ can be viewed as a way to combine results of multiple tests. For example, for any $a \in A$ there is a test constructor $w_{\langle a \rangle}$ defined by

$$w_{\langle a \rangle} \beta = \mathtt{tt} \iff \langle a, \mathtt{tt} \rangle \in \beta$$

How is such a test constructor used? Assume an LTS $h : X \to \mathcal{P}_{\mathrm{f}}(A \times X)$, and a test V on X. The function

$$BV \circ h : X \to B2,$$

applied to a process $x \in X$, first calculates the set of successors of x in h (together with the corresponding actions), and then applies the test V to each of the successors. One can then feed the result of this to the test constructor $w_{\langle a \rangle}$, obtaining a test

$$w_{\langle a \rangle} \circ BV \circ h : X \to 2$$

that checks whether, for a given process x, there exists an a-successor of x satisfying the test V.

The careful reader will by now have noticed the intuitive correspondence between the test constructor $w_{\langle a \rangle}$ and the modality $\langle a \rangle$ from Definition 2.3.

Other test constructors mentioned in this paper are $w_{[a]}$ for $a \in A$ and $w_\varnothing$, defined by

$$w_{[a]}\beta = \mathtt{ff} \iff \langle a, \mathtt{ff}\rangle \in \beta$$
$$w_\varnothing\beta = \mathtt{tt} \iff \beta = \emptyset$$

Definition 5.3 A *test suite closure* is a (large) family of monotonic functions $\mathrm{Cl}_X : X^* \to X^*$ indexed by sets X, such that for any function $f : X \to Y$, and for any test suite $\theta : Y \rightrightarrows 2$, one has

$$\mathrm{Cl}_X \{ V \circ f \,:\, V \in \theta \} = \{ V' \circ f \,:\, V' \in \mathrm{Cl}_Y\, \theta \}$$

Example 5.4 Test suite closures are conceptually simpler than test constructors in that they do not depend on any behaviour endofunctor. They merely close test suites under some constructions of choice. Examples of closures are:

$$\mathrm{Cl}^\top_X\, \theta = \theta \cup \{\mathsf{T}\}$$
$$\mathrm{Cl}^\vee_X\, \theta = \left\{ \textstyle\bigvee_{V\in\vartheta} V \,:\, \vartheta \subseteq \theta \right\}$$
$$\mathrm{Cl}^{\vee\wedge}_X\, \theta = \left\{ \textstyle\bigvee_{i=1}^n \bigwedge_{j=1}^m V_{ij} \,:\, n, m \in \mathbb{N}, V_{ij} \in \theta \right\}$$

Obviously $\mathrm{Cl}^\vee$ is the closure under arbitrary unions, and $\mathrm{Cl}^{\vee\wedge}$ under finite unions and intersections. The straightforward proof that the above constructions satisfy the requirements of Definition 5.3 is left as an exercise.

We are now ready to define a structured way of lifting endofunctors from **Set** to **TS**.

Theorem 5.5 Let B be an endofunctor on **Set**. Any set W of B-test constructors, together with any test suite closure Cl, induces a lifting of B to an endofunctor B^W on **TS**, defined by

$$B^\mathsf{W} \langle B, \theta\rangle = \left\langle BX, B^\mathsf{W}_X \theta \right\rangle \qquad B^\mathsf{W} f = Bf$$

where for any set X, the *action* $B^\mathsf{W}_X : X^* \to (BX)^*$ is a monotonic function defined by

$$B^\mathsf{W}_X \theta = \mathrm{Cl}_{BX}\{w \circ BV \mid w \in \mathsf{W}, V \in \theta\}$$

It is straightforward to check that B^W defined as above is indeed a functor, and obviously it lifts B to **TS**.

As an example, consider again $B = \mathcal{P}_\mathrm{f}(A \times -)$ and take the set of test constructors

$$\mathsf{CTr} = \left\{ w_{\langle a\rangle} \,:\, a \in A \right\} \cup \{w_\varnothing\}$$

with closure $\mathrm{Cl}^\top$. The endofunctor on **TS** arising from this choice along the lines of Theorem 5.5 will be denoted B^CTr. To understand the importance of this functor, observe that

$$h : \langle X, \theta \rangle \longrightarrow \left\langle BX, B_X^{\mathsf{CTr}}\theta \right\rangle$$

is a well-defined morphism in **TS**, i.e., a B^{CTr}-coalgebra, if and only if

$$\theta \supseteq \{\mathsf{T}\} \cup \left\{ w_{\langle a \rangle} \circ BV \circ h \ : \ a \in A, V \in \theta \right\} \cup \left\{ w_{\varnothing} \circ BV \circ h \ : \ V \in \theta \right\}$$

This means that θ must contain the totally true test T, and be closed under formation of new tests using test constructors $w_{\langle a \rangle}$ and $w_{\varnothing}$. Intuitively, the least such θ is the set of interpretations of modal formulae $\mathcal{F}_{\mathsf{CTr}}$ on the underlying LTS $h : X \to BX$. (To grasp this, and to understand the intuitive correspondence between test constructors, closures, modal operators and propositional connectives, it is useful to solve Exercise 8). This leads to the following theorem:

Theorem 5.6 For any LTS $h : X \to BX = \mathcal{P}_{\mathrm{f}}(A \times X)$, the least test suite θ on X that lifts h to a well-defined B^{CTr}-coalgebra

$$h : \langle X, \theta \rangle \longrightarrow \left\langle BX, B_X^{\mathsf{CTr}}\theta \right\rangle$$

exists, and its specialisation equivalence $\cong_\theta$ is equal to completed trace equivalence on h.

The full proof of this theorem can be found in [24], together with a family of analogous results for other well-known process equivalences. For example, choosing the set of test constructors

$$\mathsf{BS} = \left\{ w_{\langle a \rangle} \ : \ a \in A \right\} \cup \left\{ w_{[a]} \ : \ a \in A \right\}$$

with closure $\mathrm{Cl}^{\vee\wedge}$, one obtains a similar characterisation of bisimulation equivalence.

Thus the test suite approach allows one to characterise various process equivalences in a coalgebraic fashion: as (the specialisation equivalences of) the least test suites that lift LTSs to coalgebras for suitably lifted behaviour functors. The next section shows how to combine this approach with bialgebraic methods, to obtain congruence formats for these process equivalences.

6 A congruence format for completed traces

Throughout this section, fix the behaviour functor $B = \mathcal{P}_{\mathrm{f}}(A \times -)$.

To reconstruct the bialgebraic framework of Turi and Plotkin in the test suite framework, it is necessary to lift from **Set** to **TS** not only B (which can be done as shown in Section 5), but also a polynomial syntactic endofunctor Σ and the monad T freely generated by Σ. There are several natural ways of lifting polynomial endofunctors to **TS**, and it is convenient not to commit to any of them until later in this section, when the choice of lifting can be better motivated. However, it is useful to remark that whenever a functor Σ is lifted, the monad freely generated by it is lifted automatically:

Theorem 6.1 Assume an endofunctor Σ on **Set** and the monad T freely generated by Σ. If Σ is lifted to a functor Σ^* on **TS**, then Σ^* freely generates a monad T^* and T^* lifts T.

A proof of this theorem, together with a concrete characterisation of T^* in terms of Σ^*, can be found in [24].

We are now ready to state the main result of the test suite approach to congruence formats. The following statement concerns only completed trace equivalence, but it can be immediately modified to cover other process equivalences covered by the coalgebraic test suite approach of Section 5.

Theorem 6.2 Consider a transition system specification Λ in the image finite GSOS format over a signature Σ and a set of actions B. Let

$$\lambda : \Sigma(\mathrm{Id} \times B) \Longrightarrow BT$$

(where $B = \mathcal{P}_{\mathrm{f}}(A \times -)$ and T is the monad freely generated by Σ) be the natural transformation corresponding to Λ along the lines of Theorem 4.3. Moreover, assume a lifting of Σ to an endofunctor Σ^* on **TS** such that for every Σ^*-algebra

$$g : \Sigma^* \langle X, \theta \rangle \longrightarrow \langle X, \theta \rangle ,$$

the specialisation equivalence $\cong_\theta$ is a congruence on $g : \Sigma X \to X$ in the sense of Definition 4.2.

If λ lifts to a well-defined natural transformation

$$\lambda : \Sigma^*(\mathrm{Id} \times B^{\mathsf{CTr}}) \Longrightarrow B^{\mathsf{CTr}} T^*$$

then completed trace equivalence on the LTS generated by Λ is a congruence in the sense of Definition 3.2.

This theorem, proved in [24], served as an abstract basis for a congruence format for completed trace equivalence. Indeed, all one must do to obtain such a format is to choose a lifting Σ^*, and then find a syntactic restriction on image-finite GSOS specifications that, transformed along the correspondence of Theorem 4.3, guarantee the natural transformation lifting condition of Theorem 6.2. The remainder of this section is aimed at describing the process of choosing a Σ^*, and then finding suitable syntactic restrictions. Rather than giving formal proofs (which can be found in [24]), the aim is to provide as much intuition as possible, to help the reader in finding analogous formats for other process equivalences.

First, observe that for a natural transformation λ to satisfy the assumptions of Theorem 6.2, it is enough that for any object $\langle X, \theta \rangle \in \mathbf{TS}$, the component function λ_X is a well-defined morphism when lifted to **TS**:

$$\lambda_X : \Sigma^*(\langle X, \theta \rangle \times B^{\mathsf{CTr}} \langle X, \theta \rangle) \longrightarrow B^{\mathsf{CTr}} T^* \langle X, \theta \rangle$$

If all these components are well-defined, the naturality of the lifted transformation comes for free, since composition in **TS** is inherited from **Set**. By definition of **TS**, the above is equivalent to the following condition:

$$\left\{ V \circ \lambda_X \;:\; V \in B_{TX}^{\mathsf{CTr}} T_X \theta \right\} \subseteq \Sigma_{X \times BX}(\theta \otimes B_X^{\mathsf{CTr}} \theta) \tag{6.1}$$

which needs to be guaranteed for any $\theta : X \rightrightarrows 2$.

We can now try to choose a lifting Σ^* of Σ to **TS**. The obvious choice is to replace products by products and coproducts by coproducts, i.e., for

$$\Sigma X = \coprod_{\mathtt{f} \in \bar{\Sigma}} \prod_{ar(\mathtt{f})} X$$

take

$$\Sigma^* \langle X, \theta \rangle = \langle \Sigma X, \Sigma_X \theta \rangle$$

where

$$\Sigma_X \theta = \bigoplus_{\mathtt{f} \in \bar{\Sigma}} \underbrace{\theta \otimes \cdots \otimes \theta}_{ar(\mathtt{f}) \text{ times}} \tag{6.2}$$

Σ^* defined this way is indeed a functor and it lifts Σ. However, using this lifting for Theorem 6.2 one obtains a rather severe syntactic restriction for Λ.

To realise this, consider a language with a binary language construct `f` (together with some others, including a constant `nil`), described by a GSOS rule

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

together with some others, chosen so that completed trace equivalence for the language is a congruence. Moreover, consider the test suite $\theta : X \rightrightarrows 2$ consisting of just the totally true test T. It then turns out that the totally true test on TX is an element of $T_X\theta$. Then $B_{TX}^{\mathsf{CTr}} T_X \theta$ contains the test $W = w_{\langle c \rangle} \circ B\mathsf{T} : BTX \to 2$ that checks whether its argument contains any pair with c as the first component.

Now one would like the test $W \circ \lambda_X : \Sigma(X \times BX) \to 2$ to belong to $\Sigma_{X \times BX}(\theta \otimes B_X^{\mathsf{CTr}} \theta)$. This is, however, impossible: the functor Σ^* and its action defined as in (6.2) are not rich enough to contain it. Indeed, given an $r \in \Sigma(X \times BX)$, to check whether $\lambda_X r$ passes W, one needs to check that

- $r = \mathtt{f}(\langle x, \beta \rangle, \langle y, \gamma \rangle)$ for some $x, y \in X$, $\beta, \gamma \in B_X^{\mathsf{CTr}} \theta$,
- $\langle a, x' \rangle \in \beta$ for some x', and
- $\langle b, y' \rangle \in \gamma$ for some y'.

The first condition can be checked, since the action $\Sigma_{X \times BX}$ is a coproduct and thus defined by cases. However, due to the use of the categorical product $\otimes$ in (6.2), one cannot check *both* of the remaining conditions at the same time. This rules out any rules with multiple premises from the format!

Another reason for the above definition of Σ_X to be unsuitable for our purposes is illustrated by a language with a unary construct $\mathtt{g}$ (together with some others, including a constant $\mathtt{nil}$), described by GSOS rules

$$\frac{\mathtt{x} \xrightarrow{a} \mathtt{x}'}{\mathtt{g(x)} \xrightarrow{c} \mathtt{nil}} \qquad \frac{\mathtt{x} \xrightarrow{b} \mathtt{x}'}{\mathtt{g(x)} \xrightarrow{c} \mathtt{nil}}$$

Here, to check whether $\lambda_X r$ passes the same test W as above, one needs to check that

- $r = \mathsf{g}(\langle x, \beta \rangle)$ for some $x \in X$, $\beta \in B_X^{\mathsf{CTr}}\theta$, and
- $\langle a, x' \rangle \in \beta$ for some x', *or*
- $\langle b, x' \rangle \in \beta$ for some x'.

Intuitively, $\lambda_X r$ may pass W for any of two reasons corresponding to two rules above, but the construction of Σ^* does not allow to check for disjunction of those.

Based on these intuitions, another choice of Σ^* is proposed, replacing the action of (6.2) with

$$\Sigma_X\theta = \bigoplus_{\mathtt{f} \in \bar{\Sigma}} \mathrm{Cl}^{\vee}_{X^{ar(\mathtt{f})}} \underbrace{\theta \bowtie \cdots \bowtie \theta}_{ar(\mathtt{f}) \text{ times}} \tag{6.3}$$

where $\bowtie$ is as defined in (5.1), and $\mathrm{Cl}^{\vee}$ is the closure under arbitrary unions, as defined in Example 5.4. This choice of Σ^* will be used as the basis for further considerations.

To proceed with syntactic conditions guaranteeing (6.1) it is necessary to understand the nature of tests in $\Sigma_X\theta : \Sigma X \rightrightarrows 2$, $T_X\theta : TX \rightrightarrows 2$ and $B^{\mathsf{CTr}}_{TX}T_X\theta : BTX \rightrightarrows 2$ for any test suite $\theta : X \rightrightarrows 2$.

From (6.3) it is straightforward to see that tests in $\Sigma_X\theta$ are arbitrary unions of tests that, given a term $r \in \Sigma X$, can

- check the principal language construct in r, and
- perform one test $V_i \in \theta$ on every variable x_i in r, accepting when all tests accept.

Hence tests from $\Sigma_X\theta$ can be represented using terms from $\Sigma\theta$. Formally, for every $t \in \Sigma\theta$, the corresponding test $\upsilon(t) : \Sigma X \to 2$ accepts a term $r \in \Sigma X$ if and only if there is a substitution $\sigma : X \to \theta$ such that $\sigma r = t$ and every x passes $\sigma(x)$. Then $\Sigma_X\theta$ is the set of all unions of tests of the form $\upsilon(t)$.

It turns out that tests in $T_X\theta$ can be described in a very similar way, as unions of tests of the form $\upsilon(t)$ for $t \in T\theta$, where υ is defined as above (for a detailed proof of this, see [24]). In other words, tests in $T_X\theta$ are unions of tests that, given a term $t \in TX$, can

- check the syntactic structure of t, and

- perform a test $V_i \in \theta$ on every variable x_i in t, accepting when all tests accept.

The description of $B^{\mathsf{CTr}}_{TX} T_X\theta : \mathcal{P}_{\mathrm{f}}(A \times TX) \rightrightarrows 2$ is now easy, by definition of B^{CTr}. A test from this test suite can either

(a) accept immediately (the totally true test), or

(b) be a union of tests that choose a term $t \in T\theta$ and an action $a \in A$ and check whether the argument set contains a pair $\langle a, r\rangle$ such that r passes $\upsilon(t)$, or

(c) check for emptiness of the argument set.

We are now ready to tackle the main task of this section and find syntactic restrictions on an image-finite GSOS transition system specification Λ that would guarantee the condition (6.1); in other words, they should allow one to find, for any test $V \in B^{\mathsf{CTr}}_{TX} T_X\theta$, a test $W \in \Sigma_{X\times BX}(\theta \otimes B^{\mathsf{CTr}}_X\theta)$ such that $W = V \circ \lambda_X$. There are three cases to consider, corresponding to three kinds of tests in $B^{\mathsf{CTr}}_{TX} T_X\theta$:

(a) If V is the totally true test, then W is also the totally true test. It turns out that this test is always in $\Sigma_{X\times BX}(\theta \otimes B^{\mathsf{CTr}}_X\theta)$. Indeed, for each language construct $\mathtt{f} \in \bar{\Sigma}$ consider the test $W_{\mathtt{f}}$ that, given a term $r \in \Sigma(X \times BX)$, checks that the principal construct in r is $\mathtt{f}$, and then for each variable $\langle x, \beta\rangle$ in r ignores x and performs the totally true test on β. Clearly all terms in $\Sigma(X \times BX)$ with $\mathtt{f}$ as the principal construct pass $W_{\mathtt{f}}$. Then W defined by cases from all $W_{\mathtt{f}}$. Hence this case does not impose any restrictions on λ.

(b) Since $\Sigma_{X\times BX}(\theta \otimes B^{\mathsf{CTr}}_X\theta)$ is a union by (6.3), it is enough to consider not unions, but single tests of the kind described in case (b) above. Assume V checks whether the argument set contains a pair $\langle a, r\rangle$ such that r passes $\upsilon(t)$, for some fixed term $t \in T\theta$ and $a \in A$. There are several syntactic restriction one should impose on λ to ensure that $W = V \circ \lambda_X \in \Sigma_{X\times BX}(\theta \otimes B^{\mathsf{CTr}}_X\theta)$.

First, no variable on the left side of any premise in any rule can appear in the target of that rule. Consider for example a rule

$$\frac{\mathtt{x} \xrightarrow{a} \mathtt{x}'}{\mathtt{f(x)} \xrightarrow{b} \mathtt{g(x)}}$$

and a test $V \in B^{\mathsf{CTr}}_{TX} T_X\theta$ that checks whether the argument set contains a pair $\langle b, \mathtt{g}(x)\rangle$ with $x \in X$ passing some fixed test $V_0 \in \theta$. To express this as a test on $\Sigma(X \times BX)$, one needs to check that the argument term is of the form $\mathtt{f}(\langle x, \beta\rangle)$ such that x passes V_0 and β contains a pair $\langle a, x'\rangle$ for some $x' \in X$. However, this is impossible by definition of $\otimes$: tests in $\Sigma_{X\times BX}(\theta \otimes B^{\mathsf{CTr}}_X\theta)$ can, for any single variable $\langle x, \beta\rangle$ in the argument term, perform a test either on x or on β, but not on both.

Similarly, no variable can appear on the left side of more than one positive premise in any rule. Consider a rule

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

and a test V that checks whether the argument set contains a pair $\langle c, \mathtt{nil}\rangle$. To express this as a test on $\Sigma(X \times BX)$, one needs to perform two separate tests on the transition set corresponding to a single variable, which is forbidden by the use of $\bowtie$ in (6.3).

Almost exactly the same reasoning shows that no variable can appear more than once in the target of any rule. Consider a rule

$$\frac{}{\mathtt{f(x)} \xrightarrow{a} \mathtt{g(x,x)}}$$

and a test V that checks whether the argument set contains a pair $\langle b, \mathtt{g}(x, y)\rangle$ with $x \in X$ passing some fixed test $V_x \in \theta$, and $y \in Y$ passing some other fixed test $V_y \in \theta$, chosen so that the intersection of V_x and V_y is not in θ. To express this as a test on $\Sigma(X \times BX)$, one needs to perform both tests V_x and V_y on a single variable, which is again forbidden by the use of $\bowtie$ in (6.3).

The structure of the functor B^{CTr} allows a limited use of negative premises in inference rules. To illustrate this, consider a rule

$$\frac{\mathtt{x} \not\xrightarrow{a} \text{ for all } a \in A}{\mathtt{f(x)} \xrightarrow{b} \mathtt{nil}}$$

and a test V that checks whether the argument set contains a pair $\langle b, \mathtt{nil}\rangle$. One can express this as the test on $\Sigma(X \times BX)$ that, for a given $r \in \Sigma(X \times BX)$, checks whether $r = \mathtt{f}(\langle x, \beta\rangle)$ for some $x \in X$, $\beta \in BX$, and then checks whether β is empty.

On the other hand, a more liberal use of negative premises cannot be allowed. Indeed, consider a rule

$$\frac{\mathtt{x} \not\xrightarrow{a}}{\mathtt{f(x)} \xrightarrow{b} \mathtt{nil}}$$

and the test V as previously described. Here, in the corresponding test on $\Sigma(X \times BX)$, one needs to check that the given r is of the form $\mathtt{f}(\langle x, \beta\rangle)$ and that β does not contain any tuple of the form $\langle a, x'\rangle$. However, this kind of check on β does not belong to $B_X^{\mathsf{CTr}}\theta$.

(c) The final test V from $B_{TX}^{\mathsf{CTr}} T_X \theta$ to be considered is the test checking for the emptiness of the argument set. The task is to represent the test $W = V \circ \lambda_X$ as a test from $\Sigma_{X \times BX}(\theta \otimes B_X^{\mathsf{CTr}}\theta)$.

For a term $r \in \Sigma(X \times BX)$, the set $\lambda_X r$ is obtained as described in Theorem 4.3. For this set to be empty, one requires that the procedure described informally there does not yield any transitions from r. In other words, no rule for the principal construct of r can be fired according to the data contained in r. This property of r must be expresses as a test from $\Sigma_{X\times BX}(\theta \otimes B_X^{\mathsf{CTr}}\theta)$.

To analyse the fact that no rule for a construct $\mathtt{f}$ can be fired, it is convenient to speak about *minimal blocking sets* for $\mathtt{f}$. Such a minimal blocking set is a set of literals obtained by choosing a single premise from every rule for $\mathtt{f}$. A minimal blocking set can have less elements than there are rules, if some literals appear as premises of more than one rule.

Intuitively, no rule for $\mathtt{f}$ can be fired based on data contained in $r \in \Sigma(X \times BX)$ if and only if for some minimal blocking set $\mathbb{B}$ for $\mathtt{f}$, no literals from $\mathbb{B}$ are "satisfied" by r. If the latter property can be expressed, for every $\mathbb{B}$, as a test $W_{\mathbb{B}}$ in $\Sigma_{X\times BX}(\theta \otimes B_X^{\mathsf{CTr}}\theta)$, then the required test W can be obtained as the union of all such tests. However, to achieve this, some restrictions on the form of $\mathbb{B}$ are needed.

First, if $\mathbb{B}$ contains some positive literal $\mathtt{x} \xrightarrow{a} \mathtt{y}$, then it has to contain positive literals $\mathtt{x} \xrightarrow{b} \mathtt{y}_b$ for every $b \in A$. Only then one can contribute to checking that no literals from $\mathbb{B}$ are satisfied by r, checking for emptiness of the transition set provided in r with the variable from X corresponding to $\mathtt{x}$.

On the other hand, if $\mathbb{B}$ contains some negative literal $\mathtt{x} \not\xrightarrow{a}$, then it cannot contain any other negative literal $\mathtt{x} \not\xrightarrow{b}$ with the same variable on the left side. Indeed, to check that $\mathtt{x} \not\xrightarrow{a}$ is not satisfied, one wants to test the tuple $\langle x, \beta\rangle \in X \times BX$ corresponding to $\mathtt{x}$ in r, by checking that β contains a tuple $\langle a, x'\rangle$ for some $x' \in X$. But, because of the use of $\bowtie$ in (6.3), only one such test can be performed on this tuple.

Finally, note that if $\mathbb{B}$ contains *both* a positive literal and a negative literal with the same variable on the left side and the same action as a label, then some literals from $\mathbb{B}$ surely must be satisfied, no matter what r is. If this happens, the required test $W_{\mathbb{B}}$ is the totally false test on $\Sigma(X \times BX)$. This test is always an element of $\Sigma_{X\times BX}(\theta \otimes B_X^{\mathsf{CTr}}\theta)$, since it is the empty union of tests.

The above considerations do not imply at all that the restrictions mentioned are *sufficient* for λ_X to satisfy (6.1) for any X and θ. However, it turns out that they are, as is proved in detail in [24]. This leads to the following definition:

Format 1 A set of image finite GSOS rules Λ is in CTr-*format*, if:

(i) For each rule $\rho \in \Lambda$:
- if ρ has a negative premise $\mathtt{x} \not\xrightarrow{a}$, then for every label $b \in A$, ρ has also the negative premise $\mathtt{x} \not\xrightarrow{b}$,
- no variable occurs more than once in the target of ρ,

- no variable occurs simultaneously in the left-hand side of a premise and in the target of ρ,
- no variable occurs simultaneously in the left-hand side of a positive premise and in the left-hand side of any other premise of ρ.

(ii) For each construct $\mathtt{f}$ of the language, for every minimal blocking set for $\mathtt{f}$, either
- for some $\mathtt{x}, \mathtt{y} \in \Xi$, $a \in A$, both $\mathtt{x} \xrightarrow{a} \mathtt{y}$ and $\mathtt{x} \not\xrightarrow{a}$ belong to $\mathbb{B}$, or
- both conditions below hold:
  - for every $\mathtt{x}, \mathtt{y} \in \Xi$, $a \in A$, if $\mathtt{x} \xrightarrow{a} \mathtt{y} \in \mathbb{B}$ then for every $b \in A$ there is some $\mathtt{y}' \in \Xi$ such that $\mathtt{x} \xrightarrow{b} \mathtt{y}' \in \mathbb{B}$, and
  - for every $\mathtt{x} \in \Xi$, $a \in A$, if $x \not\xrightarrow{a} \in \mathbb{B}$ then for every $b \in A$ different from a, $x \not\xrightarrow{b} \notin \mathbb{B}$.

To illustrate the use of CTr-format, consider again the examples described in Section 3.

It is easy to see that **BPA** (Example 3.1) is in CTr-format. It is clear that all rules of **BPA** satisfy condition 1 of the format. For condition 2, consider a language construct $a-$ corresponding to $a \in A$. Since the only rule for $a-$:

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

has no premises, it does not have any minimal blocking set. For the binary construct $+$, the only minimal blocking set is

$$\left\{ \mathtt{x} \xrightarrow{a} \mathtt{x}' \,:\, a \in A \right\} \cup \left\{ \mathtt{y} \xrightarrow{a} \mathtt{y}' \,:\, a \in A \right\}$$

and satisfies condition 2 of the format.

On the other hand, the semantics for the encapsulation operator ∂ (Example 3.4) is not in CTr-format if $A = \{a, b\}$. Indeed, the encapsulation operator fails to satisfy the format: the set $\{\mathtt{x} \xrightarrow{a} \mathtt{y}\}$ is a minimal blocking set for $\partial_{\{b\}}$, but it does not satisfy condition 2.

Similarly, the semantics for the synchronous composition (Example 3.5) is not in CTr-format already if $A = \{a, b\}$. Here, the rules for synchronous composition fail to satisfy the format. Indeed, the set $\{\mathtt{x} \xrightarrow{a} \mathtt{x}', \mathtt{y} \xrightarrow{b} \mathtt{y}'\}$ is a minimal blocking set for the language construct $\times$, but does not satisfy condition 2.

However, **BPA** extended with sequential composition (Example 3.6) is in CTr-format. Condition 1 of the format is checked easily. For condition 2, it is enough to check it for the sequential composition operator. First, observe that for any minimal blocking set $\mathbb{B}$ for the construct ; one has

$$\left\{ \mathtt{x} \xrightarrow{a} \mathtt{x}' \,:\, a \in A \right\} \subseteq \mathbb{B}$$

Then realise that the only minimal blocking set $\mathbb{B}$ that does not contain $\mathtt{x} \stackrel{a}{\not\longrightarrow}$ for any $a \in A$ (and if it does, it necessarily satisfies condition 2) is

$$\left\{ \mathtt{x} \stackrel{a}{\longrightarrow} \mathtt{x}' \,:\, a \in A \right\} \cup \left\{ \mathtt{y} \stackrel{a}{\longrightarrow} \mathtt{y}' \,:\, a \in A \right\}$$

which also satisfies condition 2.

7 Concluding remarks

This paper describes a general, abstract framework for derivation of congruence formats for given notions of syntax, behaviour and process equivalence. The general theory, presented in Sections 4-6, provides a rather abstract condition (Theorem 6.2) that guarantees congruence properties of transition system specifications. Section 6 contains a recipe for specialising this abstract condition to a concrete format for completed trace equivalence on labelled transition systems. The intuition provided there will hopefully assist the reader who would want to apply the abstract framework to other notions of behaviour and process equivalence.

The recipe for the derivation of a new format is as follows: first, find a behaviour endofunctor appropriate for the kind of transition system considered. This paper concentrates only on ordinary LTSs, but other kinds of systems can be modelled coalgebraically as described in [34]. Then, find a concrete representation of the abstract GSOS format. For LTSs, this is provided in Theorem 1. For probabilistic and timed systems, analogous characterisations were given in [6,23]. Having done this, one needs to model the process equivalence in question in the test suite framework. This paper shows how to do it for completed trace equivalence, and mentions also bisimulation equivalence. In [24], various other equivalences on LTSs are modelled this way. Later, one needs to experiment a bit and choose the best notion of syntax lifting, and finally find additional restrictions on the concrete representation of GSOS that guarantee the counterpart of the condition (6.1).

8 Exercises

The following exercises are not very difficult, and they can be used as warm-ups while reading this paper.

1. In addition to process equivalences, many researchers have defined various *process preorders*. For example, completed trace preorder is defined by

$$x \sqsubseteq_{\mathsf{CTr}} x' \iff (\forall \phi \in \mathcal{F}_{\mathsf{CTr}}.\ x \models_h \phi \Longrightarrow x' \models_h \phi)$$

(compare with (2.1)). In general, a process equivalence is the symmetric closure of its corresponding preorder. A preorder R satisfying the compositionality condition of Definition 3.2 is called a *precongruence*.

Show that if a process preorder is a precongruence on some language, than the corresponding process equivalence is a congruence.

2. Write a transition system specification such that bisimulation equivalence is not a congruence on the generated LTS.
3. Show that completed trace equivalence $\cong_{\mathsf{CTr}}$ is a congruence for **BPA** extended with sequential composition (Example 3.6).

   Hint: Show how the set of all formulae in $\mathcal{F}_{\mathsf{CTr}}$ satisfied by a process $\mathtt{t};\mathtt{t}'$ can be calculated from the corresponding sets for processes $\mathtt{t}$ and $\mathtt{t}'$.
4. *Trace equivalence* $\cong_{\mathsf{Tr}}$ is defined as completed trace equivalence, but using a restricted logic
$$\phi ::= \top \mid \langle a \rangle \, \phi$$
with semantics as in Definition 2.3. *Failures equivalence* $\cong_{\mathsf{Fl}}$ is defined by an extended logic
$$\phi ::= \top \mid \langle a \rangle \, \phi \mid \tilde{Q}$$
(where Q ranges over subsets of A), with semantics as in Definition 2.3, extended with an additional case
$$x \models_h \tilde{Q} \iff x \not\xrightarrow{a} \text{ for all } a \in Q$$
Show that
$$\cong_{\mathsf{Fl}} \subseteq \cong_{\mathsf{CTr}} \subseteq \cong_{\mathsf{Tr}}$$
on any LTS. Then show that neither $\cong_{\mathsf{Fl}}$ nor $\cong_{\mathsf{Tr}}$ is a congruence for **BPA** extended with sequential composition (Example 3.6).
5. Check the universal properties of products and coproducts in **TS** as defined in Section 5.
6. Show that morphisms in **TS** preserve specialisation equivalences of test suites. Show that the converse is not true, i.e., that there exist functions preserving specialisation equivalences that are not test suite morphisms.
7. Prove that $\mathrm{Cl}^{\top}$, $\mathrm{Cl}^{\vee}$ and $\mathrm{Cl}^{\vee\wedge}$ are test suite closures according to Definition 5.3.
8. For any LTS $h : X \to BX$, the set of formulae $\mathcal{F}_{\mathsf{CTr}}$ defined in Section 3 can be interpreted as a test suite $\theta_{\mathsf{CTr}} : X \rightrightarrows 2$:
$$\theta_{\mathsf{CTr}} = \{\, V_\phi \,:\, \phi \in \mathcal{F}_{\mathsf{CTr}} \text{ and } Vx = \mathtt{tt} \iff x \models_h \phi \,\}$$
Prove that
$$h : \langle X, \theta_{\mathsf{CTr}} \rangle \longrightarrow \left\langle BX, B_X^{\mathsf{CTr}} \theta_{\mathsf{CTr}} \right\rangle$$
is a well-defined coalgebra in **TS**.
9. From the experience of choosing a lifting Σ^* and its action Σ_X in Section 6, it seems that choosing a richer lifting (i.e., such that gives a larger test suite on a given argument test suite) one gets a more general congruence format. Following this idea, replace the closure $\mathrm{Cl}^{\vee}$ with the closure under

arbitrary unions and intersections. Take for granted that descriptions of tests from $\Sigma_X\theta$ and $T_X\theta$ remain as described in Section 6, with the only difference that instead of unions of tests represented by terms, it contains all unions of intersections of such tests. Try to find restrictions on λ that guarantee (6.1) in this case, and realise what goes wrong.

Hint: It is enough to consider a rule as simple as

$$\frac{\mathtt{x} \xrightarrow{a} \mathtt{y}}{\mathtt{f(x)} \xrightarrow{a} \mathtt{g(y)}}$$

9 Problems to tackle

This section contains some open problems and possible ways to improve or generalise the test suite approach.

1. The entire test suite approach seems closely related to modal logic decomposition techniques of, e.g., [15]. Tests correspond to modal formulae, test constructors and closures to modal operators and propositional connectives, and the condition (6.1) seems to say something about the possibility to decompose modal formulae along inference rules. This correspondence must be investigated.
2. CTr-format cannot be compared with any other congruence format for completed trace equivalence, since it is, to the author's best knowledge, the first such format published. Note, however, that there are natural examples of GSOS specifications that behave well with respect to completed trace semantics, but are not in CTr-format, as the following example (pointed out by Rob van Glabbeek) shows, extending **BPA** with a collection of rules for binary *Kleene star* $\star$:

$$\frac{\mathtt{x} \xrightarrow{\alpha} \mathtt{x'}}{\mathtt{x} \star \mathtt{y} \xrightarrow{\alpha} \mathtt{x'}; (\mathtt{x} \star \mathtt{y})} \qquad \frac{\mathtt{y} \xrightarrow{\alpha} \mathtt{y'}}{\mathtt{x} \star \mathtt{y} \xrightarrow{\alpha} \mathtt{y'}}$$

where α ranges over A. It is straightforward to check that completed trace equivalence is a congruence for the above rules. However, **BPA** extended with the Kleene star is not in CTr-format. The first rule in Example 5.25 does not satisfy the third part of condition 1 of the format. Indeed, the variable x occurs both on the left side of the premise and in the target of the rule.

This means that there is some improvement to be made concerning the generality of the formats derived with the test suite approach.

On the concrete level, the solution would be to apply the frozen/liquid position trick of [9]. There, variables in inference rules are divided into two sets, treated rather differently in the format, which allows for more generality of the format. The "failure trace format" of [9] states that if there

exists a division of all variables satisfying certain conditions, then failure trace equivalence (another well-known process equivalence) is a congruence.

The test suite approach as described here does not cover any techniques of this kind. In particular, in both liftings Σ^* from Section 6 all positions in terms are treated equally.

The solution to this problem might rely on an almost immediate generalisation of Theorem 6.2. In the statement of the theorem, the lifting Σ^* is arbitrary but fixed. It is easy to see that one can quantify universally over all such liftings in the definition of a format. Theorem 6.2 then takes the form "for any Λ, if there exists a lifting Σ (satisfying the algebra congruence condition) such that λ lifts, then completed trace (or another) equivalence is a congruence". This gives an additional degree of freedom in the definition of a format, and it seems to correspond somehow to the quantification over divisions of variables in the frozen/liquid position technique.

3. An obvious direction of future work is to apply the test suite approach to other well-known process equivalences on LTSs and compare the results obtained with other known congruence formats. In [25,24], this has been done for trace equivalence and failures equivalence. Then one should want to apply the technique to other behaviour endofunctors. Two good starting points are [6] and [23], where concrete descriptions of abstract GSOS for probabilistic and timed transition systems are given.
4. There is a common agreement in the literature on coalgebraic techniques that bisimulation equivalence is a canonical process equivalence. This canonicity is presently not reflected in the test suite approach; it is not clear what is special about the choice of test constructors and a test suite closure leading to a test suite description of bisimulation equivalence. One should try to fill this gap, defining a canonical choice of test constructors and a closure for any behaviour endofunctor (and even any underlying category in place of **Set**), providing a notion of bisimulation equivalence parametrised by behaviour (as it is done in the coalgebra span approach). An expected result would be that the corresponding condition (6.1) would be vacuous, related to the fact that GSOS is a congruence format for bisimulation equivalence.

References

[1] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *4th ACM Conference on Computer and Communications Security*, pages 36–47. ACM Press, 1997.

[2] L. Aceto, W. J. Fokkink, and C. Verhoef. Structural operational semantics. In J. A. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*. Elsevier, 2002.

[3] H. H. Andersen and M. Mendler. An asynchronous process algebra with multiple clocks. In *Proc. ESOP '94*, volume 788 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.

[4] J. C. M. Baeten and W. P. Weijland. *Process Algebra*. Cambridge University Press, 1990.

[5] F. Bartels. GSOS for probabilistic transition systems. In L. Moss, editor, *Proc. CMCS'02*, volume 65 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2002.

[6] F. Bartels. *On Generalised Coinduction and Probabilistic Specification Formats*. PhD dissertation, CWI, Amsterdam, 2004.

[7] J. A. Bergstra and J. W. Klop. Process algebra for synchronous communication. *Information and Control*, 60:109–137, 1984.

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

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

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

[11] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31:560–599, 1995.

[12] L. Cardelli and A. D. Gordon. Mobile ambients. In *Proc. FOSSACS '98*, volume 1378 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[13] V. Danos and J. Krivine. Formal molecular biology done in CCS. In *Proc. BioCONCUR'03*, 2003.

[14] U. Engberg and M. Nielsen. A calculus of communicating systems with name-passing. Technical Report DAIMI PB-208, Computer Science Department, Aarhus University, 1986.

[15] W. J. Fokkink, R. J. van Glabbeek, and P. de Wind. Compositionality of Hennessy-Milner logic through structural operational semantics. In *Proc. FCT'03*, volume 2751 of *Lecture Notes in Computer Science*, pages 412–422. Springer Verlag, 2003.

[16] R. J. van Glabbeek. The linear time – branching time spectrum II. In E. Best, editor, *Proc. CONCUR'93*, volume 715 of *Lecture Notes in Computer Science*, pages 66–81, 1993.

[17] R. J. van Glabbeek. The linear time – branching time spectrum I. In J. A. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*. Elsevier, 1999.

[18] M. Hennessy. *Algebraic Theory of Processes*. MIT Press, 1988.

[19] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32:137–161, 1985.

[20] C. Hermida and B. Jacobs. Structural induction and coinduction in a fibrational setting. *Information and Computation*, 145(2):107–152, 1998.

[21] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[22] B. Jacobs and J. Hughes. Simulations in coalgebra. *Electronic Notes in Theoretical Computer Science*, 82, 2003.

[23] M. Kick. Rule formats for timed processes. In *Proc. CMCIM'02*, volume 68 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2002.

[24] B. Klin. *Abstract Coalgebraic Approach to Process Equivalence for Well-Behaved Operational Semantics*. PhD thesis, BRICS, Aarhus University, 2004.

[25] B. Klin and P. Sobocinski. Syntactic formats for free: An abstract approach to process equivalence. In *Proc. CONCUR 2003*, volume 2671 of *Lecture Notes in Computer Science*, 2003.

[26] K. G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94:1–28, 1991.

[27] S. Mac Lane. *Categories for the Working Matematician*. Springer-Verlag, second edition, 1998.

[28] R. Milner. A calculus of communicating systems. *Journal of the ACM*, 1980.

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

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

[31] D. M. Park. Concurrency and automata on infinite sequences. *Lecture Notes in Computer Science*, 140:195–219, 1981.

[32] G. D. Plotkin. The origins of structural operational semantics. *Journal of Logic and Algebraic Programming*, 60-61:17–139, 2004.

[33] A. Regev, W. Silverman, and E. Shapiro. Representation and simulation of biochemical processes using the π-calculus process algebra. In *Proc. Pacific Symposium of Biocomputing*, 2001.

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

[35] D. Turi and G. D. Plotkin. Towards a mathematical operational semantics. In *Proc. LICS'97*, pages 280–291. IEEE Computer Society Press, 1997.