EPTCS 300

Proceedings of the

Combined 26th International Workshop on Expressiveness in Concurrency and 16th Workshop on Structural Operational Semantics

Amsterdam, The Netherlands, 26th August 2019

Edited by: Jorge A. Pérez and Jurriaan Rot

Published: 22nd August 2019
DOI: 10.4204/EPTCS.300
ISSN: 2075-2180
Open Publishing Association

Table of Contents

Preface

This volume contains the proceedings of EXPRESS/SOS 2019, the Combined 26th International Workshop on Expressiveness in Concurrency and the 16th Workshop on Structural Operational Semantics. Following a long tradition, EXPRESS/SOS 2019 was held as one of the affiliated workshops of the 30th International Conference on Concurrency Theory (CONCUR 2019), in Amsterdam (The Netherlands).

The EXPRESS/SOS workshop series aims at bringing together researchers interested in the formal semantics of systems and programming concepts, and in the expressiveness of computational models. In particular, topics of interest for the workshop include (but are not limited to):

- expressiveness and rigorous comparisons between models of computation (process algebras, event structures, Petri nets, rewrite systems);
- expressiveness and rigorous comparisons between programming languages and models (distributed, component-based, object-oriented, service-oriented);
- logics for concurrency (modal logics, probabilistic and stochastic logics, temporal logics and resource logics);
- analysis techniques for concurrent systems;
- theory of structural operational semantics (metatheory, category-theoretic approaches, congruence results);
- comparisons between structural operational semantics and other formal semantic approaches;
- applications and case studies of structural operational semantics;
- software tools that automate, or are based on, structural operational semantics.

This year, the Program Committee selected 6 submissions for inclusion in the scientific program—four full papers and the following two short papers:

- *Coherent Resolutions of Nondeterminism*, by Marco Bernardo.
- *A GSOS for Attribute-based Communication*, by Marino Miculan and Matteo Paier.

This volume contains revised versions of the four full papers as well as contributed papers associated to the following three invited presentations, which nicely complemented the scientific program:

- *Cellular monads from Positive GSOS specifications*, by Tom Hirschowitz (CNRS / Savoie Mont Blanc University, France)
- *Comparing Process Calculi Using Encodings*, by Kirstin Peters (TU Berlin/TU Darmstadt, Germany)
- *Semantic Structures for Spatially-Distributed Multi-Agent Systems*, by Frank Valencia (CNRS-LIX, Ecole Polytechnique de Paris, France and Univ. Javeriana Cali, Colombia)

We would like to thank the authors of the submitted papers, the invited speakers, the members of the program committee, and their subreviewers for their contribution to both the meeting and this volume. We also thank the CONCUR 2019 organizing committee for hosting the workshop. Finally, we would like to thank our EPTCS editor Rob van Glabbeek for publishing these proceedings and his help during the preparation.

Jorge A. Pérez and Jurriaan Rot,
August 2019

Program Committee

Additional Reviewers

Cellular Monads from Positive GSOS Specifications

Tom Hirschowitz[*]
Univ. Grenoble Alpes, Univ. Savoie Mont Blanc, CNRS, LAMA
73000 Chambéry, France
tom.hirschowitz@univ-smb.fr

We give a leisurely introduction to our abstract framework for operational semantics based on cellular monads on transition categories. Furthermore, we relate it for the first time to an existing format, by showing that all Positive GSOS specifications generate cellular monads whose free algebras are all compositional. As a consequence, we recover the known result that bisimilarity is a congruence in the generated labelled transition system.

1 Introduction

1.1 Motivation

In the vast majority of foundational research on programming languages, although ideas are thought of as widely applicable, they are presented on *one*, simple example. Typically, there is a tension between simplicity of exposition, leading to the minimal language making the idea relevant, and significance, leading to the most expressive one. Strikingly, the scope of the idea is often mostly clear to the experts, but no attempt is made at stating it precisely. The reason for this is that the mathematical concepts needed for even only making such statements are lacking. Indeed, one needs to be able to say something like: "for all programming languages of such shape, the following holds". But there simply is no widely accepted mathematical notion of programming language.

Such a general notion should account for both

(i) the interaction between syntax and dynamics, as involved in, e.g., structural operational semantics [21], or in the statement of results like type soundness, congruence of program equivalence, or compiler correctness, and

(ii) denotational semantics, in the sense of including not only operational, syntactic models but also others, typically ones in which program equivalence is coarser.

Typically, standard formats [19] elude denotational semantics, and are exclusively syntactic. To our knowledge, the only such proposals meeting all these criteria are *functorial operational semantics*, a.k.a. *bialgebraic semantics* [25], and a few variants [4, 24]. This approach has been deeply developed, and shown to extend smoothly to various settings, e.g., non-deterministic and probabilistic languages. However, two important extensions have proved more difficult.

- The treatment of languages with variable binding is significantly more technical than the basic setting [9, 8, 24].
- More importantly, the bialgebraic study of higher-order languages like the λ-calculus or the higher-order π-calculus is only in its infancy [20].

This leaves some room for exploring potential alternatives.

[*]Thanks to Jorge Pérez and Jurriaan Rot for the invitation, and to the referees for helpful comments.

1.2 Context

In recent work [12], a new approach to abstract operational semantics was proposed, and its expressive power was demonstrated by proving for the first time an abstract soundness result for *bisimulation up to context* in the presence of variable binding. Bisimulation up to context is an efficient technique [23, Chapter 6] for proving program equivalences, which had previously been proved correct in the bialgebraic setting [2], but only without binding.

Its novelty mainly resides in the following two technical features.

Transition categories First, while standard operational semantics is based on *labelled transition systems*, this is both generalised and abstracted over in the framework.

Generalisation Indeed, in the examples, instead of standard labelled transition systems, we use a slight generalisation similar in spirit to [6], essentially from relations to graphs, i.e., possibly with several transitions between two states. This simple, harmless generalisation brings in a lot of useful structure, typically that of a *topos* [17], which is unavailable at this level in bialgebraic operational semantics.

Abstraction In full generality, the framework takes as a parameter a *transition category*, a typical example of which is given by such generalised transition systems. For any object of a given transition category, bisimulation may be defined by lifting, following an idea from [14].

Combinatorial category theory A second technical innovation is the use of advanced combinatorial category theory. To start with, *familial monads* [3], or rather their recent *cellular* variant [10], provide a notion of evaluation context for both programs and transitions, at the abstract level. Standard reasoning by induction on context thus becomes simple algebraic calculation. A second, crucial notion is cofibrantly generated factorisation systems, a notion from homotopy theory [13, 22] which, together with cellularity, allows for a conceptually simple, yet relevant characterisation of well-behaved transition contexts.

Each instance of the framework is then constructed as follows.

Type of transition system The first step is the choice of a type of transition system, which may involve different kinds of states (e.g., initial or final ones), the set of labels to be put on transitions, etc. Technically, this amounts to fixing a transition category $\mathscr{C}$. This also fixes the relevant notion of bisimulation, hence bisimilarity.

Transition rules The second step consists in defining the dynamics of the considered language, which is usually specified through a set of inference rules. This comes in as a monad T on $\mathscr{C}$, whose algebras are essentially the transition systems satisfying the given inference rules. The standard, syntactic transition system is typically the free algebra $T(0)$. This fixes the relevant notion of context closure. In this setting, congruence of bisimilarity $\sim_X$ on a T-algebra X is the fact that $T(\sim_X) \to X^2$ factors through $\sim_X \to X^2$ (see (1) on page 7).

One of the main results [12, Corollary 4.30] is that if the considered algebra is *compositional*, in the sense that its structure map $T(X) \to X$ is a functional bisimulation, and if the monad T satisfies an additional condition, then bisimilarity is indeed a congruence. The latter condition is called $\mathbf{T}_s$-*familiality* in [12], but we will here call it *cellularity*, because it is a specialisation of cellularity in the sense of [10] to familial functors. As mentioned above, a second main result [12, Corollary 5.15] is that under a different condition called $\mathbf{T}_s^{\vee}$-*familiality*, bisimulation up to context is sound.

1.3 Contribution

One of the main issues with cellular monads T on transition categories $\mathscr{C}$ is the lack of an efficient generation mechanism, i.e., a mathematical construction that produces pairs $(\mathscr{C}, T)$ from more basic data. In this paper, we initiate the search for such generating constructions by showing that an existing simple format, *Positive GSOS* [1], always produces cellular monads whose free algebras are compositional. As a consequence, we recover (Theorem 10.4) the known result that bisimilarity is a congruence in all free algebras.

As this is an invited contribution, we briefly introduce the approach at an expository, rather concrete level. In particular, the only considered transition category is the one of generalised labelled transition systems in the sense alluded to above. Finally, our proofs are meant to be instructive rather than fully detailed.

1.4 Plan

In §2, we explain our generalisation of labelled transition systems, and bisimulation by lifting. In §3, we recall Positive GSOS specifications Σ and show how they generate monads T_Σ. In §4, we argue that algebras for the obtained monad T_Σ are a good notion of model for the considered Positive GSOS specification. We then state congruence of bisimilarity in categorical terms, and quickly reduce it to two key properties: (i) compositionality of the considered algebra and (ii) preservation of functional bisimulations by T_Σ.

We deal with (i) in §5, where we show that when T_Σ is obtained from a Positive GSOS specification, all free algebras are compositional. In §6, we then attack (ii), by further reducing it to familiality and cellularity. The remaining sections develop these ideas.

In §7, we define familiality for functors (as opposed to monads), and show that T_Σ is a familial functor. In §8, we establish some factorisation properties of familial functors which were announced and used in §4 to reduce congruence of bisimilarity to compositionality and preservation of bisimulation. We then introduce cellularity in §9, and show that T_Σ is indeed cellular. Finally, we wrap up in §10 by defining familiality for monads (which is slightly more demanding than for mere functors), and showing that T_Σ does form a familial monad. This fills a hole left open in §5, thus allowing us to state the main theorem.

Finally, we conclude and give some perspective in §11.

1.5 Prerequisites

We assume familiarity with basic category theory [16, 15], including categories, functors, natural transformations, monads and their algebras, and the Yoneda lemma.

2 Labelled transition systems as presheaves

2.1 Generalised transition systems

A standard SOS specification is given by a signature, plus a family of transition rules over a fixed set $\mathbb{A}$ of labels. The set $\mathbb{A}$ fixes the relevant kind of transition system, and we interpret this by constructing a corresponding category of (generalised) transition systems. Given any set $\mathbb{A}$, let $\Gamma_\mathbb{A}$ denote the graph with

- vertex set $\mathbb{A}+1$, i.e., vertices are elements of $\mathbb{A}$, denoted by $[a]$ for $a \in \mathbb{A}$, plus a special vertex $\star$,

- two edges $s^a, t^a : \star \to [a]$, for all $a \in \mathbb{A}$.

Pictorially, $\Gamma_{\mathbb{A}}$ looks like this:

$$\begin{tikzcd} \ldots & {[a]} & \ldots & (a \in \mathbb{A}) \\ & \star. \arrow[u, bend left, "s^a"] \arrow[u, bend right, "t^a"'] \end{tikzcd}$$

There are no composable edges in Γ_A, so, adding formal identity arrows, it readily forms a category, which we also denote by Γ_A.

Definition 2.1. The *category of transition systems induced by* $\mathbb{A}$ is $\widehat{\Gamma_{\mathbb{A}}}$, the category of presheaves over $\Gamma_{\mathbb{A}}$.

To see what presheaves over $\Gamma_{\mathbb{A}}$ have to do with transition systems, let us observe that a presheaf $X \in \widehat{\Gamma_{\mathbb{A}}}$ consists of a set $X(\star)$ of *states*, together with, for each $a \in \mathbb{A}$, a set of *transitions* $e \in X[a]$ with source and target maps $X(s^a), X(t^a) : X[a] \to X(\star)$. Our notion is thus only slightly more general than standard labelled transition systems over $\mathbb{A}$, in that it allows several transitions with the same label between two given states.

Remark 2.2. The category $\widehat{\Gamma_{\mathbb{A}}}$ may be viewed as a category of labelled graphs. Indeed, letting $\Omega_{\mathbb{A}}$ denote the one-vertex graph with $\mathbb{A}$ loops on it, we have by well-known abstract nonsense an equivalence $\mathbf{Gph}/\Omega_{\mathbb{A}} \simeq \widehat{\Gamma_{\mathbb{A}}}$ of categories. (This is due to the fact that $\Gamma_{\mathbb{A}}$ is isomorphic to the *category of elements* of $\Omega_{\mathbb{A}}$, see Definition 7.1 below.)

Notation 2.3. For any $X \in \widehat{\Gamma_{\mathbb{A}}}$, we denote the action of morphisms in $\Gamma_{\mathbb{A}}$ with a dot. E.g., if $e \in X[a]$, then $e \cdot s^a \in X(\star)$ is its source. We also sometimes abbreviate s^a and t^a to just s and t.

Example 2.4. For languages like CCS [18], we let $\mathbb{A} = \mathscr{N} + \mathscr{N} + 1$ denote the disjoint union of a fixed set $\mathscr{N}$ of *channel names* with itself and the singleton 1. Elements of the first term are denoted by $\overline{a}$, for $a \in \mathscr{N}$, and are used for output transitions, while elements of the second term, simply denoted by a, are used for input transitions. Finally, the unique element of the third term is denoted by τ and used for silent transitions. E.g., the labelled transition system

$$\begin{tikzcd} x & y \arrow[l, "\overline{a}"'] \arrow[r, bend left, "b"] \arrow[r, bend right, "b"'] & z \arrow[loop right, "a"] \end{tikzcd}$$

is modelled by the presheaf X with

$$X(\star) = \{x, y, z\} \qquad \begin{array}{c} X(\overline{a}) = \{e\} \\ X(b) = \{f, f'\} \\ X(a) = \{g\} \end{array} \qquad \begin{array}{c} x = e \cdot t \\ y = e \cdot s = f \cdot s = f' \cdot s \\ z = f \cdot t = f' \cdot t = g \cdot s = g \cdot t. \end{array}$$

2.2 Bisimulation

Returning to generalised transition systems, we may define bisimulation categorically in the following way. Morphisms $f : X \to Y$, i.e., natural transformations, are the analogue in our setting of standard functional simulations. Indeed, given any transition $e : x \xrightarrow{a} x'$ in X, then $f(x)$ sure has an a transition to some state related to x': this is simply $f(e)$! The next step is to define an analogue of functional bisimulation. For this, let us observe that the base category $\Gamma_{\mathbb{A}}$ embeds into the presheaf category $\widehat{\Gamma_{\mathbb{A}}}$ – this is just the Yoneda embedding $\mathbf{y} : \Gamma_{\mathbb{A}} \to \widehat{\Gamma_{\mathbb{A}}}$, directly specialised to our setting for readability:

- the state object $\star$ embeds as the one-vertex graph $\mathbf{y}_\star$ with no transition;
- any transition object $[a]$ embeds as the graph $\mathbf{y}_{[a]}$ with one a-transition between two distinct vertices;

- the morphisms $s^a, t^a : \star \to [a]$ embed as the morphisms $\mathbf{y}_\star \to \mathbf{y}_{[a]}$ picking up the source and target, respectively, of the given transition.

Notation 2.5. We often omit $\mathbf{y}$, treating it as an implicit coercion.

Definition 2.6. Let $f : X \to Y$ be a *functional bisimulation* whenever all commuting squares as the solid part below, admit a (potentially non-unique) *lifting* k as shown, i.e., a morphism making both triangles commute.

$$\begin{tikzcd}
\star \arrow[r, "x"] \arrow[d, "s^a"'] & X \arrow[d, "f"] \\
{[a]} \arrow[r, "e"'] \arrow[ur, dashed, "k"] & Y
\end{tikzcd}$$

Let us explain why this matches the standard definition. In any such square, x is essentially the same as just a state in X, while e is just an a-transition in Y. Furthermore, the composite $\star \xrightarrow{s^a} [a] \xrightarrow{e} Y$ picks the source of e, so commutation of the square says that the source $e \cdot s^a$ of e is in fact $f(x)$ (a.k.a. $f \circ x$). So we are in the situation described by the solid part below.

$$\begin{tikzcd}
x \arrow[r, mapsto, "f"] \arrow[d, dashed, "k"'] & f(x) \arrow[d, "e"] \\
x' \arrow[r, dashed, mapsto, "f"'] & y'
\end{tikzcd}$$

Finding a lifting k then amounts to finding an antecedent to e whose source is x, as desired.

We finally recover the analogue of standard bisimulation relations.

Definition 2.7. A *bisimulation relation* on X is a subobject $R \hookrightarrow X^2$ (= isomorphism class of monomorphisms into X^2) whose projections $R \to X$ are both functional bisimulations.

In this case, the above diagram specialises to

$$\begin{tikzcd}
(x_1, x_2) \arrow[r, mapsto, "\pi_i"] \arrow[d, dashed, "(e_1,e_2)"'] & x_i \arrow[d, "e_i"] \\
(x'_1, x'_2) \arrow[r, dashed, mapsto, "\pi_i"'] & x'_i,
\end{tikzcd}$$

where $(x_1, x_2), (x'_1, x'_2) \in R(\star)$, and $(e_1, e_2) \in R[a]$.

Now, $\widehat{\Gamma_{\mathbb{A}}}$, as a presheaf category, is very well-behaved, namely it is a Grothendieck topos [17]. In particular, subobjects of X^2 form a (small) complete lattice, in which the union of a family $R_i \hookrightarrow X^2$ is computed by first taking the copairing $\sum_i R_i \to X^2$, which is generally not monic, and then taking its image. Furthermore, bisimulation relations are closed under unions and so admit a maximum element, *bisimilarity* [12, Proposition 3.14].

The presheaf category $\widehat{\Gamma_{\mathbb{A}}}$ is thus only a slight generalisation of standard labelled transition systems over $\mathbb{A}$, in which we have an analogue of bisimulation, conveniently defined by lifting, and bisimilarity. Let us now consider the case where states are terms in a certain language, and transitions are defined inductively by a set of transition rules, i.e., operational semantics.

3 Positive GSOS specifications as monads

Let us briefly recall the Positive GSOS format. We fix a set $\mathbb{A}$ of labels, and start from a *signature* $\Sigma_0 = (O_0, E_0)$ on **Set**, i.e., a set O_0 equipped with a map $E_0 : O_0 \to \mathbb{N}$.

Definition 3.1. A *Positive GSOS rule* over Σ_0 consists of

- an operation $f \in O_0$, say of arity $n = E_0(f)$,
- a label $a \in \mathbb{A}$,
- n natural numbers $m_1, \ldots, m_n$,
- for all $i \in n$, m_i labels $a_{i,1}, \ldots, a_{i,m_i}$, and
- a term t with $n + \sum_{i=1}^{n} m_i$ free variables.

In more standard form, such a rule is just

$$\frac{\ldots \qquad x_i \xrightarrow{a_{i,j}} y_{i,j} \qquad \ldots \qquad (i \in n, j \in m_i)}{f(x_1, \ldots, x_n) \xrightarrow{a} t}$$

where the x_i's and $y_{i,j}$'s are all distinct and denote the potential free variables of t.

Definition 3.2. A *Positive GSOS specification* is a signature Σ_0, together with a set Σ_1 of Positive GSOS rules.

Let us now describe how any Positive GSOS specification Σ induces a monad T_Σ on $\widehat{\Gamma_{\mathbb{A}}}$, starting with the action of T_Σ on objects. Given any $X \in \widehat{\Gamma_{\mathbb{A}}}$, the set $T_\Sigma(X)(\star)$ of states consists of all Σ_0-terms with variables in $X(\star)$, as defined by the grammar

$$M, N ::= (\!|u|\!) \mid f(M_1, \ldots, M_n),$$

where u ranges over $X(\star)$. Similarly, each $T_\Sigma(X)[a]$ consists of all transition proofs following the rules in Σ_1, with axioms in all $X[a']$'s. Formally, such proofs are constructed inductively from the following rules,

$$\frac{}{(\!|e|\!) :_X (\!|e \cdot s|\!) \xrightarrow{a} (\!|e \cdot t|\!)}\ (e \in X[a]) \qquad \frac{\ldots \qquad R_{i,j} :_X M_i \xrightarrow{a_{i,j}} M_{i,j} \qquad \ldots \qquad (i \in n, j \in m_i)}{\rho(R_{i,j})_{i \in n, j \in m_i} :_X f(M_1, \ldots, M_n) \xrightarrow{a} t[(x_i \mapsto M_i, (y_{i,j} \mapsto M_{i,j})_{j \in m_i})_{i \in n}]}$$

where in the second rule $f \in O_0$, $E_0(f) = n$, $\rho = (f, a, (m_i, (a_{i,j})_{j \in m_i})_{i \in n}, t) \in \Sigma_1$. When $m_i = 0$, we want to keep track of M_i in the transition proof, so by convention the family $(R_{i,j})_{j \in m_i}$ denotes just M_i. In the sequel we simply call *transitions* such transition proofs.

Example 3.3. Let us consider the following simple CCS transition of depth > 1, in any $T_{CCS}(X)[\tau]$.

$$\frac{\dfrac{\dfrac{}{(\!|e_1|\!) :_X (\!|x_1|\!) \xrightarrow{\overline{a}} (\!|y_1|\!)}}{lpar((\!|e_1|\!), (\!|x_2|\!)) :_X (\!|x_1|\!) | (\!|x_2|\!) \xrightarrow{\overline{a}} (\!|y_1|\!) | (\!|x_2|\!)} \qquad \dfrac{}{(\!|e_2|\!) :_X (\!|x_3|\!) \xrightarrow{a} (\!|y_2|\!)}}{sync(lpar((\!|e_1|\!), (\!|x_2|\!)), (\!|e_2|\!)) :_X ((\!|x_1|\!) | (\!|x_2|\!)) | (\!|x_3|\!) \xrightarrow{\tau} ((\!|y_1|\!) | (\!|x_2|\!)) | (\!|y_2|\!)} \quad ,$$

where *lpar* and *sync* denote the left parallel and synchronisation rules, $(e_1 : x_1 \xrightarrow{\overline{a}} y_1) \in X[\overline{a}]$, $x_2 \in X(\star)$, and $(e_2 : x_3 \xrightarrow{a} y_2) \in X[a]$.

The source and target of a transition $R :_X M \xrightarrow{a} N$ are M and N, respectively, which ends the definition of T_Σ on objects. On morphisms $f : X \to Y$, $T_\Sigma(f)$ merely amounts to renaming variables $(\!|x|\!)$ and $(\!|e|\!)$ to $(\!|f(x)|\!)$ and $(\!|f(e)|\!)$, respectively. It thus remains to show that T_Σ has monad structure. The unit $\eta_X : X \to T_\Sigma(X)$ is obviously given by $(\!|-|\!)$, while multiplication $\mu_X : T_\Sigma(T_\Sigma(X)) \to T_\Sigma(X)$ is given inductively by removing the outer layer of $(\!|-|\!)$'s

$$\begin{array}{lrcl}
\text{on states} & \mu_X(\!|M|\!) & = & M \\
 & \mu_X(f(M_1, \ldots, M_n)) & = & f(\mu_X(M_1), \ldots, \mu_X(M_n)) \\
\text{and on transitions} & \mu_X(\!|R|\!) & = & R \\
 & \mu_X(\rho(R_{i,j})_{i \in n, i \in m_i}) & = & \rho(\mu_X(R_{i,j}))_{i \in n, i \in m_i}.
\end{array}$$

Lemma 3.4. *The natural transformations η and μ equip T_Σ with monad structure.*

Proof. A straightforward induction. □

4 Models as algebras and congruence of bisimilarity

Algebras for T_Σ readily give the right notion of model for the transition rules:

Definition 4.1. An *algebra* for a monad T, or a T*-algebra*, consists of an object X, equipped with a morphism $\alpha : T(X) \to X$ such that the following diagrams commute.

$$\begin{tikzcd} T(T(X)) \arrow[r, "T(\alpha)"] \arrow[d, "\mu_X"'] & T(X) \arrow[d, "\alpha"] \\ T(X) \arrow[r, "\alpha"'] & X \end{tikzcd} \qquad \begin{tikzcd} X \arrow[rr, "\eta_X"] \arrow[dr, equal] & & T(X) \arrow[dl, "\alpha"] \\ & X & \end{tikzcd}$$

Thus, intuitively, a T_Σ-algebra is a transition system which is stable under the given operations and transition rules.

We now would like to show that, under suitable hypotheses, bisimilarity for any given T_Σ-algebra $\alpha : T_\Sigma(X) \to X$ is a congruence. We may state this categorically by saying that the canonical morphism $T_\Sigma(\sim_X) \to X^2$ factors through $m : (\sim_X) \hookrightarrow X^2$, as in

$$\begin{tikzcd} T_\Sigma(\sim_X) \arrow[rr, dashed] \arrow[d, "T_\Sigma(m_X)"'] & & \sim_X \arrow[d, "m"] \\ T_\Sigma(X^2) \arrow[r, "\langle T_\Sigma(\pi_1), T_\Sigma(\pi_2)\rangle"'] & (T_\Sigma(X))^2 \arrow[r, "\alpha^2"'] & X^2. \end{tikzcd} \tag{1}$$

Indeed, an element of $T_\Sigma(\sim_X)$ is a term M whose free variables are pairs of bisimilar elements of X, which we write as $M((x_1,y_1),\ldots,(x_n,y_n))$, with $x_i \sim_X y_i$ for all $i \in n$. The morphism $\langle T_\Sigma(\pi_1), T_\Sigma(\pi_2)\rangle$ maps this to the pair

$$(M(x_1,\ldots,x_n), M(y_1,\ldots,y_n)),$$

which α^2 then evaluates componentwise. The given factorisation thus boils down to

$$\alpha(M(x_1,\ldots,x_n)) \sim_X \alpha(M(y_1,\ldots,y_n))$$

for all M and $x_1 \sim_X y_1, \ldots, x_n \sim_X y_n$, i.e., bisimilarity is a congruence.

In order to prove such a property, it is sufficient to prove that T_Σ preserves *all* bisimulation relations, in the sense that if $m : R \hookrightarrow X^2$ is a bisimulation relation, then so is

$$T_\Sigma(R) \xrightarrow{T_\Sigma(m)} T_\Sigma(X^2) \xrightarrow{\langle T_\Sigma(\pi_1), T_\Sigma(\pi_2)\rangle} (T_\Sigma(X))^2 \xrightarrow{\alpha^2} X^2$$

(in the slightly generalised sense that its image is). Equivalently, an easy diagram chasing shows that it all boils down to

$$T_\Sigma(R) \xrightarrow{T_\Sigma(m)} T_\Sigma(X^2) \xrightarrow{T_\Sigma(\pi_i)} T_\Sigma(X) \xrightarrow{\alpha} X$$

being a functional bisimulation for $i \in \{1,2\}$.

Finally, $\pi_i \circ m$ is a functional bisimulation by definition, and functional bisimulations are stable under composition, so it is sufficient to prove that *(i)* the considered algebra is *compositional*, in the sense that its structure map $\alpha : T_\Sigma(X) \to X$ is a functional bisimulation, and

(ii) T_Σ preserves all functional bisimulations.

Compositionality essentially means that transitions of any $\alpha(M(x_1,\ldots,x_n))$ are all obtained by assembling transitions of the x_i's. This is not always the case, even for free algebras:

Example 4.2. Consider a specification Σ consisting of the unique rule

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

say ρ, where f and g are two unary operations. Then the free algebra $\mu_1 : T_\Sigma(T_\Sigma(1)) \to T_\Sigma(1)$ is not compositional. Indeed, 1 contains a unique vertex, say $\star$, and a transition $b : \star \xrightarrow{b} \star$ for all labels b. Thus, $T_\Sigma(1)$ contains a transition $\rho(\!|a|\!) : f(g(\!|\star|\!)) \xrightarrow{a} f(g(\!|\star|\!))$. But the term $f(g(\!|\star|\!))$ is the image under μ_1 of $f(\!|g(\!|\star|\!)|\!)$, which has no transition.

Summing up, we have proved:

Lemma 4.3. *If T_Σ preserves functional bisimulations, then bisimilarity in any compositional T_Σ-algebra is a congruence.*

5 Compositionality

Let us first consider compositionality. For a general algebra, we cannot do more than taking compositionality as a hypothesis. However, we can say something when the considered algebra is free:

Lemma 5.1. *The multiplication $\mu_X : T_\Sigma(T_\Sigma(X)) \to T_\Sigma(X)$ is a functional bisimulation.*

Proof. We will see below (Lemma 10.3) that all naturality squares of μ are pullbacks. In particular, we have a pullback

$$\begin{tikzcd}
T_\Sigma(T_\Sigma(X)) \arrow[r, "T_\Sigma(T_\Sigma(!))"] \arrow[d, "\mu_X"'] \arrow[dr, phantom, "\lrcorner", very near start] & T_\Sigma(T_\Sigma(1)) \arrow[d, "\mu_1"] \\
T_\Sigma(X) \arrow[r, "T_\Sigma(!)"'] & T_\Sigma(1).
\end{tikzcd}$$

But functional bisimulations are easily seen to be stable under pullback, so it is enough to show that μ_1 is a functional bisimulation. We thus consider any term $\mathbf{M}$ whose free variables are in $T_\Sigma(1)(\star)$, i.e., are themselves terms over a single free variable, say $\star$, together with a transition $R : \mu_1(\mathbf{M}) \xrightarrow{a} N$. And we need to show that there exists a transition $\mathbf{R} : \mathbf{M} \xrightarrow{a} \mathbf{N}$ whose free variables and axioms are in $T_\Sigma(1)$, such that $\mu_{[a]}(\mathbf{R}) = R$. We proceed by induction on $\mathbf{M}$:

- If $\mathbf{M} = (\!|M|\!)$, then taking $\mathbf{R} = (\!|R|\!)$ does the job.
- Otherwise, $\mathbf{M} = f(\mathbf{M}_1,\ldots,\mathbf{M}_n)$, so $M = \mu_1(\mathbf{M}) = f(M_1,\ldots,M_n)$, with $M_i = \mu_1(\mathbf{M}_i)$ for all $i \in n$. But then, R must have the form $\rho(R_{i,j})_{i\in n, j\in m_i}$, for a certain rule $\rho = (f,a,(m_i,(a_{i,j})_{j\in m_j})_{i\in n},t)$ of Σ. By induction hypothesis, we find for all $i \in n$ and $j \in m_i$ a transition

$$\mathbf{R}_{i,j} : \mathbf{M}_i \xrightarrow{a_{i,j}} \mathbf{N}_{i,j},$$

such that $\mu_1(\mathbf{R}_{i,j}) = R_{i,j}$. Thus, $\mathbf{R} = \rho(\mathbf{R}_{i,j})_{i\in n, j\in m_i}$ does have $\mathbf{M}$ as its source, and furthermore satisfies $\mu_1(\mathbf{R}) = R$, as desired. □

6 Preserving bisimulations through familiality and cellularity

Let us now consider *(ii)*, i.e., the fact that T_Σ preserves functional bisimulations. So we need to find a lifting to any commuting square of the form

$$\begin{tikzcd} \star \arrow[r, "M"] \arrow[d, "s^a"'] & T_\Sigma(X) \arrow[d, "T_\Sigma(f)"] \\ {[a]} \arrow[r, "R"'] & T_\Sigma(Y), \end{tikzcd}$$

for any functional bisimulation f.

We will proceed in two steps: we will require T_Σ to be first familial, and then cellular. Familiality will allow us to factor the given square as the solid part below left, while cellularity will ensure existence of a lifting k as on the right.

$$\begin{tikzcd} \star \arrow[r, "M'"] \arrow[d, "s^a"'] \arrow[rr, bend left, "M"] & T_\Sigma(A) \arrow[r, "T_\Sigma(\varphi)"] \arrow[d, "T_\Sigma(\gamma)"'] & T_\Sigma(X) \arrow[d, "T_\Sigma(f)"] \\ {[a]} \arrow[r, "R'"'] \arrow[rr, bend right, "R"'] & T_\Sigma(B) \arrow[r, "T_\Sigma(\psi)"'] \arrow[ur, dashed, "T_\Sigma(k)"] & T_\Sigma(Y) \end{tikzcd} \qquad \begin{tikzcd} A \arrow[r, "\varphi"] \arrow[d, "\gamma"'] & X \arrow[d, "f"] \\ B \arrow[r, "\psi"'] \arrow[ur, dashed, "k"] & Y \end{tikzcd} \tag{2}$$

The composite $T_\Sigma(k) \circ R'$ will thus give the desired lifting for the original square.

At this stage, both steps may seem mysterious to the reader. In fact, as we will see, factorisation as above left follows directly from the fact that T_Σ may be expressed as a sum of representable functors. Let us first explain intuitively why this latter fact holds. We will then prove it more rigorously in §7, to eventually return to factorisation in §8.

To start with, let us observe that the set $T_\Sigma(1)(\star)$ consists of terms over a single free variable, say $\star$. For any such term M, we may count the number of occurrences of $\star$, say n_M. Thus, any term in any $T_\Sigma(X)(\star)$ is entirely determined by an $M \in T_\Sigma(1)(\star)$, together with a map $n_M \to X(\star)$ assigning an element of $X(\star)$ to each occurrence of $\star$ in M. But maps $n_M \to X(\star)$ in **Set** are in 1-1 correspondence with maps $n_M \cdot \mathbf{y}_\star \to X$ in $\widehat{\Gamma_{\mathbb{A}}}$, where $n_M \cdot \mathbf{y}_\star$ denotes the n_M-fold coproduct $\mathbf{y}_\star + \cdots + \mathbf{y}_\star$ of $\mathbf{y}_\star$ with itself. In other words, letting $E^\star(M) = n_M \cdot \mathbf{y}_\star$, we have

$$T_\Sigma(X)(\star) \cong \sum_{M \in T_\Sigma(1)(\star)} \widehat{\Gamma_{\mathbb{A}}}(E^\star(M), X). \tag{3}$$

Clearly, for any $f : X \to Y$, the action of $T_\Sigma(f)$ at $\star$ is given by postcomposing with f, i.e., we have

$$\begin{tikzcd} T_\Sigma(X)(\star) \arrow[r, "\cong"] \arrow[d, "T_\Sigma(f)_\star"'] & \sum_{M \in T_\Sigma(1)(\star)} \widehat{\Gamma_{\mathbb{A}}}(E^\star(M), X) \arrow[d, "\sum_{M \in T_\Sigma(1)(\star)} \widehat{\Gamma_{\mathbb{A}}}(E^\star(M), f)"] \\ T_\Sigma(Y)(\star) \arrow[r, "\cong"] & \sum_{M \in T_\Sigma(1)(\star)} \widehat{\Gamma_{\mathbb{A}}}(E^\star(M), Y). \end{tikzcd}$$

The family (3) of isomorphisms is thus natural in X. We will see shortly that this extends to objects other than $\star$. Indeed, any transition in $T_\Sigma(X)[a]$ may be decomposed into a transition R in $T_\Sigma(1)[a]$, together with a morphism $E^a(R) \to X$, where $E^a(R)$ is obtained from occurrences of term and transition variables in R.

We have seen that our isomorphisms are natural in X, so it seems natural to try to express some naturality constraint in the second argument of T_Σ. But this requires making the right-hand side of (3) functorial in this variable in the first place! In fact, for any transition $R : M \xrightarrow{a} N$, we will construct morphisms

$$E^\star(M) \xrightarrow{E(s^a \upharpoonright R)} E^a(R) \xleftarrow{E(t^a \upharpoonright R)} E^\star(N)$$

(see Notation 7.2 below). Thus, e.g., precomposing by the left-hand map yields the desired functorial action

$$\sum_{R \in T_\Sigma(1)[a]} \widehat{\Gamma_{\mathbb{A}}}(E^a(R), X) \to \sum_{M \in T_\Sigma(1)(\star)} \widehat{\Gamma_{\mathbb{A}}}(E^\star(M), X),$$

of $s^a : \star \to [a]$, sending any $\varphi : E^a(R) \to X$ to the composite

$$E^\star(M) \xrightarrow{E(s^a \upharpoonright R)} E^a(R) \xrightarrow{\varphi} X. \tag{4}$$

7 Familiality for functors

Let us now state more rigorously the definition of familiality and the fact that T_Σ is familial. In the next section, we will explain how this entails the desired factorisation (2).

Definition 7.1. The *category of elements* $el(X)$ of any presheaf $X \in \widehat{\mathscr{C}}$ on any category $\mathscr{C}$ has

- as objects all pairs (c, x) with $c \in \mathbf{ob}(\mathscr{C})$ and $x \in X(c)$,
- and as morphisms $(c, x) \to (c', x')$ all morphisms $f : c \to c'$ such that $x' \cdot f = x$.

Notation 7.2. The morphism f, viewed as a morphism $(c, x) \to (c', x')$, is entirely determined by f and x'. We denote it by $f \upharpoonright x'$.

Definition 7.3. An endofunctor $F : \widehat{\mathbb{C}} \to \widehat{\mathbb{C}}$ on a presheaf category is *familial* iff there is a functor $E : el(F(1)) \to \widehat{\mathbb{C}}$ such that

$$F(X)(c) \cong \sum_{o \in F(1)(c)} \widehat{\mathbb{C}}(E(c, o), X), \tag{5}$$

naturally in $X \in \widehat{\mathbb{C}}$ and $c \in \mathbb{C}$.

And indeed, we have:

Lemma 7.4. *The endofunctor T_Σ is familial.*

Proof. We need to do two things: (1) extend the isomorphisms (3) to objects of the form $[a]$, and (2) define the morphisms $E(s^a \upharpoonright R)$ and $E(t^a \upharpoonright R)$ rendering our isomorphisms natural also in the second argument of T_Σ. In fact, we will do almost everything simultaneously by induction: we define $E^a(R)$ and $E(s^a \upharpoonright R) : E^\star(R \cdot s^a) \to E^a(R)$ by induction on R. By convention, as we did for the unique element $\star \in 1(\star)$, we denote the unique element of $1[a]$ by a itself.

- If $R = (\!|a|\!)$, then its source is $M = (\!|\star|\!)$ and we put $E^a(R) = \mathbf{y}_{[a]}$ and $E(s^a \upharpoonright R) = s^a : \mathbf{y}_\star \to \mathbf{y}_{[a]}$.
- If $R = \rho(R_{i,j})_{i \in n, j \in m_i} :_1 M \xrightarrow{a} N$, then for all i and $j \in m_i$, by induction hypothesis, we get morphisms
$$E(s^{a_{i,j}} \upharpoonright R_{i,j}) : E^\star(M_i) \to E^{a_{i,j}}(R_{i,j}),$$
where $R_{i,j} :_1 M_i \xrightarrow{a_{i,j}} N_{i,j}$ for all $i \in n$ and $j \in m_i$. Let us temporarily fix any $i \in n$. For all $j, j' \in m_i$, we have $R_{i,j} \cdot s^a = R_{i,j'} \cdot s^a = M_i$, so we take the wide pushout $E_i = \bigoplus_{E^\star(M_i)} E^{a_{i,j}}(R_{i,j})$, i.e., the colimit of the following diagram.

$$\begin{tikzcd}
 & E^\star(M_i) \arrow[dl, "E(s^{a_{i,j}} \upharpoonright R_{i,j})"'] \arrow[dr, "E(s^{a_{i,j'}} \upharpoonright R_{i,j'})"] & \\
\ldots E^{a_{i,j}}(R_{i,j}) & \ldots & E^{a_{i,j'}}(R_{i,j'}) \ldots
\end{tikzcd} \tag{6}$$

If $m_i = 0$, this reduces to just $E^\star(M_i)$, which is exactly what we want. Finally, we let $E^a(R)$ be the coproduct $\sum_i E_i$ of all the E_i's, and observe that $E^\star(f(M_1,\ldots,M_n)) = \sum_i E^\star(M_i)$ by definition, so that we may define $E(s^a \restriction R)$ to be the coproduct $\sum_i S_i$ of all canonical injections $S_i : E^\star(M_i) \to E_i$.

This ends the inductive definition of $E^a(R)$ and $E(s^a \restriction R)$. We now need to construct the morphisms $E(t^a \restriction R)$. We again proceed inductively. When $R = (\!|a|\!)$, the desired morphism is clearly t^a itself. When $R = \rho(R_{i,j})_{i\in n, j\in m_i}$, the target is $N = t[x_i \mapsto M_i, (y_{i,j} \mapsto N_{i,j})_{j\in m_i}]$. Now, by construction, occurrences $occ_\star(N)$ of the unique variable $\star$ in N are in 1-1 correspondence with

$$V_N = \sum_i \left((occ_\star(M_i))^{occ_{x_i}(t)} + \sum_{j\in m_i} (occ_\star(N_{i,j}))^{occ_{y_{i,j}}(t)} \right),$$

and our map $E(t^a \restriction R)$ should reflect the intended correspondences. Since $E^\star(N) = V_N \cdot \mathbf{y}_\star$, $E(t^a \restriction R)$ is entirely determined by choosing a map $E_u : \mathbf{y}_\star \to E^a(R)$ for all $u \in V_N$:

- If u denotes an occurrence of $\star$ in M_i, for some occurrence of x_i in t, we let E_u denote the composite

$$\mathbf{y}_\star \to E^\star(M_i) \to E^\star(R),$$

where the latter map denotes injection into the colimit of (6).

- If u denotes an occurrence of $\star$ in $N_{i,j}$, for some occurrence of $y_{i,j}$ in t, we let E_u denote the composite

$$\mathbf{y}_\star \to E^\star(N_{i,j}) \xrightarrow{E(t^{a_{i,j}} \restriction R_{i,j})} E^{a_{i,j}}(R_{i,j}) \to E^\star(R),$$

where the latter map again denotes injection into the colimit of (6). □

Rather than a full formal proof, let us illustrate that our construction satisfies the isomorphisms (3) on a few examples.

Example 7.5. In the case of the transition of Example 3.3, familiality means that this transition is determined by picking the following transition in $T_{CCS}(1)[\tau]$,

$$\dfrac{\dfrac{\dfrac{}{(\!|\overline{a}|\!) :_1 (\!|\star|\!) \xrightarrow{\overline{a}} (\!|\star|\!)}}{lpar((\!|\overline{a}|\!),(\!|\star|\!)) :_1 (\!|\star|\!)|(\!|\star|\!) \xrightarrow{\overline{a}} (\!|\star|\!)|(\!|\star|\!)} \qquad \dfrac{}{(\!|a|\!) :_1 (\!|\star|\!) \xrightarrow{a} (\!|\star|\!)}}{sync(lpar((\!|\overline{a}|\!),(\!|\star|\!)),(\!|a|\!)) :_1 ((\!|\star|\!)|(\!|\star|\!))|(\!|\star|\!) \xrightarrow{\tau} ((\!|\star|\!)|(\!|\star|\!))|(\!|\star|\!)}$$

together with a morphism $E^\tau(sync(lpar((\!|\overline{a}|\!),(\!|\star|\!)),(\!|a|\!))) \to X$. Let us start with $E^\star(lpar((\!|\overline{a}|\!),(\!|\star|\!)))$: it is given by the colimit of

$$\begin{tikzcd} \mathbf{y}_\star \arrow[d, "s^{\overline{a}}"'] & \mathbf{y}_\star \\ \mathbf{y}_{[\overline{a}]} & \end{tikzcd}$$

(one $\mathbf{y}_\star$ for each argument x_i of $lpar$, and for each x_i one $\mathbf{y}_{[a_{i,j}]}$ for each premise $x_i \xrightarrow{a_{i,j}} y_{i,j}$). Equivalently, this is just the coproduct $\mathbf{y}_{[\overline{a}]} + \mathbf{y}_\star$, and $E(s^{\overline{a}} \restriction lpar((\!|\overline{a}|\!),(\!|\star|\!)))$ and $E(t^{\overline{a}} \restriction lpar((\!|\overline{a}|\!),(\!|\star|\!)))$ are given by

$$\mathbf{y}_\star + \mathbf{y}_\star \xrightarrow{s^{\overline{a}}+\mathbf{y}_\star} \mathbf{y}_{[\overline{a}]} + \mathbf{y}_\star \xleftarrow{t^{\overline{a}}+\mathbf{y}_\star} \mathbf{y}_\star + \mathbf{y}_\star.$$

It is then clear that $E(s^\tau \restriction sync(lpar((\!|\overline{a}|\!),(\!|\star|\!)),(\!|a|\!)))$ is given by

$$\mathbf{y}_\star + \mathbf{y}_\star + \mathbf{y}_\star \xrightarrow{s^{\overline{a}}+\mathbf{y}_\star+s^a} \mathbf{y}_{[\overline{a}]} + \mathbf{y}_\star + \mathbf{y}_{[a]}.$$

Now how about $E(t^\tau \restriction sync(lpar((\!|\overline{a}|\!),(\!|\star|\!)),(\!|a|\!)))$? As the term t occurring in the rule is here linear in the $y_{i,j}$'s, an easy computation leads to

$$\mathbf{y}_{[\overline{a}]} + \mathbf{y}_\star + \mathbf{y}_{[a]} \xleftarrow{t^{\overline{a}}+\mathbf{y}_\star+t^a} \mathbf{y}_\star + \mathbf{y}_\star + \mathbf{y}_\star.$$

On this example, the isomorphism (5) thus boils down to the transition $sync(lpar((\!|e_1|\!),(\!|x_2|\!)),(\!|e_2|\!))$ above being entirely determined by picking $R = sync(lpar((\!|\overline{a}|\!),(\!|\star|\!)),(\!|a|\!)) \in T_{CCS}(1)[\tau]$, and giving a morphism

$$\varphi : E^\tau(sync(lpar((\!|\overline{a}|\!),(\!|\star|\!)),(\!|a|\!))) = \mathbf{y}_{[\overline{a}]} + \mathbf{y}_\star + \mathbf{y}_{[a]} \longrightarrow X,$$

which holds by universal property of coproduct and the Yoneda lemma. Naturality of (5) in c says that the source of (R,φ) is given up to this correspondence by $((\!|\star|\!)|(\!|\star|\!))|(\!|\star|\!)$ and the composite

$$E^\star(((\!|\star|\!)|(\!|\star|\!))|(\!|\star|\!)) = \mathbf{y}_\star + \mathbf{y}_\star + \mathbf{y}_\star \xrightarrow{s^{\overline{a}}+\mathbf{y}_\star+s^a} \mathbf{y}_{[\overline{a}]} + \mathbf{y}_\star + \mathbf{y}_{[a]} \longrightarrow X,$$

and likewise for the target.

Example 7.6. Let us now illustrate the treatment of branching, in the sense of a rule having several premises involving the same x_i. An example from CCS is the 'replicated synchronisation' rule

$$\frac{x_1 \xrightarrow{\overline{a}} y_{1,1} \qquad x_1 \xrightarrow{a} y_{1,2}}{!x_1 \xrightarrow{\tau} !x_1|(y_{1,1}|y_{1,2})},$$

say *rsync*. First, $E^\tau(rsync((\!|[\overline{a}]|\!),(\!|[a]|\!)))$ is simply the pushout

$$\begin{tikzcd}
\mathbf{y}_\star \arrow[r, "s^a"] \arrow[d, "s^{\overline{a}}"'] & \mathbf{y}_{[a]} \arrow[d] \\
\mathbf{y}_{[\overline{a}]} \arrow[r] & E^\tau(rsync((\!|[\overline{a}]|\!),(\!|[a]|\!))) \arrow[ul, phantom, "\ulcorner", very near start]
\end{tikzcd},$$

which rightly models the fact that a transition $rsync((\!|e_1|\!),(\!|e_2|\!)) \in T_{CCS}(X)[\tau]$ is entirely determined by picking $rsync((\!|[\overline{a}]|\!),(\!|[a]|\!)) \in T_{CCS}(1)[\tau]$, together with elements e_1 and e_2 of $X[\overline{a}]$ and $X[a]$ with a common source. The morphism $E(s^\tau \restriction rsync((\!|[\overline{a}]|\!),(\!|[a]|\!)))$ is then straightforwardly given by the diagonal. The target morphism $E(t^\tau \restriction rsync((\!|[\overline{a}]|\!),(\!|[a]|\!)))$ is a bit more complex to compute. Indeed, the target $t = !x_1|(y_{1,1}|y_{1,2})$ has three free variables. The first, x_1, should yield a morphism $\mathbf{y}_\star \to E^\tau(rsync((\!|[\overline{a}]|\!),(\!|[a]|\!)))$ that is determined by the source morphism $E^\star(M_1) \to E^{\overline{a}}(R_{1,1})$. Here, we get

$$\mathbf{y}_\star \xrightarrow{s^{\overline{a}}} E^{\overline{a}}((\!|[\overline{a}]|\!)) = \mathbf{y}_{[\overline{a}]} \to E^\tau(rsync((\!|[\overline{a}]|\!),(\!|[a]|\!))).$$

On the other hand, $y_{1,1}$ and $y_{1,2}$ should be determined by the target morphisms $E^\star(M_{1,1}) \to E^{\overline{a}}(R_{1,1})$ and $E^\star(M_{1,2}) \to E^a(R_{1,2})$, in our case

$$\mathbf{y}_\star \xrightarrow{t^{\overline{a}}} E^{\overline{a}}((\!|[\overline{a}]|\!)) = \mathbf{y}_{[\overline{a}]} \to E^\tau(rsync((\!|[\overline{a}]|\!),(\!|[a]|\!))) \quad \text{and} \quad \mathbf{y}_\star \xrightarrow{t^a} E^a((\!|a|\!)) = \mathbf{y}_a \to E^\tau(rsync((\!|[\overline{a}]|\!),(\!|[a]|\!))).$$

8 Familiality and factorisation

Let us now return to our proof sketch (2), and explain the properties of familiality that allow us to factor the original square as indicated. The crucial observation is that elements of the form

$$(R, id_{E^c(R)}) \in \sum_{R \in T_\Sigma(1)(c)} \widehat{\Gamma_{\mathbb{A}}}(E^c(R), E^c(R)) \cong T_\Sigma(E^c(R))(c)$$

have the special property that any other element of the form $(R,\varphi) \in T_\Sigma(X)(c)$ may be obtained uniquely as the image of (R, id) by the action of

$$T_\Sigma(\varphi)_c : T_\Sigma(E^c(R))(c) \to T_\Sigma(X)(c).$$

Having the same first component R is equivalent to having the same image in $T_\Sigma(1)(c)$. So by Yoneda, having two elements of $T_\Sigma(X)(c)$ and $T_\Sigma(Y)(c)$ with common first component is the same as having a commuting square of the form below left.

$$\begin{tikzcd} \mathbf{y}_c \arrow[r] \arrow[d] & T_\Sigma(X) \arrow[d] \\ T_\Sigma(Y) \arrow[r] & T_\Sigma(1) \end{tikzcd} \qquad \begin{tikzcd} \mathbf{y}_c \arrow[r, "p"] \arrow[d, "\xi"'] & T_\Sigma(X) \arrow[d, "T_\Sigma(!)"] \\ T_\Sigma(E^c(R)) \arrow[r, "T_\Sigma(!)"'] \arrow[ur, dashed, "T_\Sigma(k)"] & T_\Sigma(1) \end{tikzcd}$$

The special property of (R, id) is thus equivalently that any commuting square as above right (solid part) admits a unique (dashed) lifting k as shown, making the non-trivial triangle commute. In fact, this holds more generally by replacing $\mathbf{y}_c$ and 1 by arbitrary objects:

Definition 8.1. Given a functor $F : \mathscr{C} \to \mathscr{D}$, a morphism $\xi : D \to F(C)$ is *F-generic*, or *generic* for short, when any commuting square as the solid part of

$$\begin{tikzcd} D \arrow[r, "\chi"] \arrow[d, "\xi"'] & F(B) \arrow[d, "F(h)"] \\ F(C) \arrow[r, "F(k)"'] \arrow[ur, dashed, "F(l)"] & F(A) \end{tikzcd}$$

admits a unique *strong* lifting l as shown, in the sense that $F(l) \circ \xi = \chi$ and $h \circ l = k$.

Lemma 8.2 ([26, Remark 2.12]). *A functor $F : \widehat{\mathbb{C}} \to \widehat{\mathbb{C}}$ is familial iff any morphism $Y \to F(X)$ factors as*

$$Y \xrightarrow{\xi} F(A) \xrightarrow{F(\varphi)} F(X),$$

where ξ is F-generic. This is called a generic-free *factorisation.*

Proof sketch. ($\Rightarrow$) Passing from $\mathbf{y}_c$ to any Y goes by observing that generic morphisms are stable under colimits in the comma category $\widehat{\mathbb{C}} \downarrow F$, remembering that any presheaf Z is a colimit of the composite

$$el(Z) \xrightarrow{\mathbf{p}_Z} \mathbb{C} \xrightarrow{\mathbf{y}} \widehat{\mathbb{C}},$$

where $\mathbf{p}_Z$ denotes the obvious projection functor.
($\Leftarrow$) Conversely, $E(c,o)$ is given by A, for any choice of generic-free factorisation

$$\begin{tikzcd} & F(A) \arrow[dr, "F(\varphi)"] & \\ \mathbf{y}_c \arrow[ur, "\xi"] \arrow[rr, "o"'] & & F(1). \end{tikzcd}$$

□

Lemma 8.2 thus accounts for the factorisation of the original square as on the left in (2): M and R respectively factor as

$$\star \xrightarrow{M'} T_\Sigma(A) \xrightarrow{T_\Sigma(\varphi)} T_\Sigma(X) \qquad \text{and} \qquad \star \xrightarrow{R'} T_\Sigma(B) \xrightarrow{T_\Sigma(\psi)} T_\Sigma(Y),$$

with M' and R' generic. But genericness of M' yields the strong lifting γ in

$$\begin{tikzcd}
\star \arrow[rr, "s^a"] \arrow[d, "M'"'] & & {[a]} \arrow[r, "R'"] & T_\Sigma(B) \arrow[d, "T_\Sigma(\psi)"] \\
T_\Sigma(A) \arrow[rrru, dashed, "T_\Sigma(\gamma)"] \arrow[rr, "T_\Sigma(\varphi)"'] & & T_\Sigma(X) \arrow[r, "T_\Sigma(f)"'] & T_\Sigma(Y).
\end{tikzcd}$$

9 Cellularity

We have now factored the original square as promised, but for the moment we have no guarantee that the 'inner' square

$$\begin{tikzcd}
A \arrow[r, "\varphi"] \arrow[d, "\gamma"'] & X \arrow[d, "f"] \\
B \arrow[r, "\psi"'] & Y
\end{tikzcd} \tag{7}$$

will admit a lifting. The point of cellularity is precisely this. For once, let us start from the abstract viewpoint and explain how directly relevant it is in this case.

The starting point is the observation that our definition of bisimulation by lifting is based on a Galois connection. Indeed, for any class $\mathscr{L}$ of morphisms, let $\mathscr{L}^{\boxslash}$ denote the class of maps $f : X \to Y$ such that for any $l : A \to B$ in $\mathscr{L}$, any commuting square as below left admits a (not necessarily unique) lifting.

$$\begin{tikzcd}
A \arrow[r, "u"] \arrow[d, "\mathscr{L}\ni l"'] & X \arrow[d, "f\in\mathscr{L}^{\boxslash}"] \\
B \arrow[r, "v"'] & Y
\end{tikzcd} \qquad \begin{tikzcd}
X \arrow[r, "u"] \arrow[d, "{}^{\boxslash}\mathscr{R}\ni f"'] & A \arrow[d, "r\in\mathscr{R}"] \\
Y \arrow[r, "v"'] & B
\end{tikzcd}$$

Conversely, given a class $\mathscr{R}$ of morphisms, let ${}^{\boxslash}\mathscr{R}$ denote the class of morphisms $f : X \to Y$ such that for any $r : A \to B$ in $\mathscr{R}$, any commuting square as above right admits a lifting. Clearly, letting $\mathscr{S}$ denote the set of all maps of the form $s^a : \star \to [a]$, $\mathscr{S}^{\boxslash}$ catches exactly all functional bisimulations. But what is ${}^{\boxslash}(\mathscr{S}^{\boxslash})$? In other words, which maps will admit a lifting against all functional bisimulations? This is very relevant to us, because finding a lifting for our inner square (7) is obviously equivalent to showing that $\gamma \in {}^{\boxslash}(\mathscr{S}^{\boxslash})$! Fortunately, the theory of weak factorisation systems gives a precise characterisation [13, Corollary 2.1.15], of which we only need the following very special cases:

Lemma 9.1. *Maps in ${}^{\boxslash}(\mathscr{S}^{\boxslash})$ are closed under composition and pushout, in the sense that*

- *for any composable $f, g \in {}^{\boxslash}(\mathscr{S}^{\boxslash})$, $g \circ f \in {}^{\boxslash}(\mathscr{S}^{\boxslash})$, and*
- *for any $f : X \to Y$ in ${}^{\boxslash}(\mathscr{S}^{\boxslash})$ and $u : X \to X'$, the pushout f' of f along u, as below, is again in ${}^{\boxslash}(\mathscr{S}^{\boxslash})$.*

$$\begin{tikzcd}
X \arrow[r, "f\in{}^{\boxslash}(\mathscr{S}^{\boxslash})"] \arrow[d, "u"'] & Y \arrow[d, "u'"] \\
X' \arrow[r, "f'\in{}^{\boxslash}(\mathscr{S}^{\boxslash})"'] \arrow[ur, phantom, "\ulcorner", very near end] & Y'
\end{tikzcd}$$

This is useful to us because the map γ that we want to show is in ${}^{\boxslash}(\mathscr{S}^{\boxslash})$ may be obtained as a finite composite of pushouts of maps in $\mathscr{S}$, which allows us to conclude. Indeed, γ occurs in

$$\begin{tikzcd}
\star \arrow[r, "s^a"] \arrow[d, "M'"'] & {[a]} \arrow[d, "R'"] \\
T_\Sigma(E^\star(M)) \arrow[r, "T_\Sigma(\gamma)"'] & T_\Sigma(E^a(R)),
\end{tikzcd}$$

with M' and R' generic. So (M',γ) is the generic-free factorisation of $R' \circ s^a$ as in Lemma 8.2, hence, because generic-free factorisations are unique up to canonical isomorphism, we can actually compute γ. Indeed, letting $M' = (M'', id)$ and $R' = (R'', id)$, for suitable $M'' \in T_\Sigma(1)(\star)$ and $R'' \in T_\Sigma(1)[a]$, by (4) $R' \circ s^a$ is the pair $(R'' \cdot s^a, E(s^a \restriction R''))$, where

$$E(s^a \restriction R'') : E^\star(R'' \cdot s^a) \to E^a(R'')$$

is obtained by familiality of T_Σ. We thus get

$$(M'',\gamma) = (R'' \cdot s^a, E(s^a \restriction R'')),$$

hence in particular

$$\gamma = E(s^a \restriction R'').$$

It is thus sufficient to show that each $E(s^a \restriction R)$ is in ${}^{\boxslash}(\mathscr{S}^{\boxslash})$. This goes by induction on R, following an incremental construction of $E(s^a \restriction R)$. The base case is clear. When $R = \rho(R_{i,j})_{i\in n, j\in m_i}$, remember from the proof of Lemma 7.4 that $E^a(R)$ is the coproduct $\sum_i E_i$ for $i \in n$, each E_i being constructed as the wide pushout of all

$$E(s^{a_{i,j}} \restriction R_{i,j}) : E^\star(M_i) \to E^{a_{i,j}}(R_{i,j}).$$

Coproducts may be constructed by pushout along 0, so it suffices to show that each diagonal $E^\star(M_i) \to E_i$ is in ${}^{\boxslash}(\mathscr{S}^{\boxslash})$ if each $E(s^{a_{i,j}} \restriction R_{i,j})$ is. This in turn boils down to incrementally constructing the diagonal $E^\star(M_i) \to E_i$ by successively pushing out each $E(s^{a_{i,j}} \restriction R_{i,j})$: assuming that we have constructed the diagonal $E^\star(M_i) \to E_i^j$ up until $j < m_i$, we can incorporate $R_{i,j+1}$ by composing with the bottom morphism of

$$\begin{tikzcd}
E^\star(M_i) \arrow[r, "E(s^{a_{i,j+1}} \restriction R_{i,j+1})"] \arrow[d] & E^{a_{i,j+1}}(R_{i,j+1}) \arrow[d] \\
E_i^j \arrow[r] \arrow[ur, phantom, "\ulcorner"] & E_i^{j+1},
\end{tikzcd}$$

which is indeed in ${}^{\boxslash}(\mathscr{S}^{\boxslash})$ by Lemma 9.1. Clearly, the obtained $E_i^{m_i}$ is canonically isomorphic to E_i, so we have shown:

Lemma 9.2. *The monad T_Σ is* cellular, *in the sense that in any commuting square of the form*

$$\begin{tikzcd}
\star \arrow[r, "s^a"] \arrow[d, "\xi"'] & {[a]} \arrow[d, "\chi"] \\
T_\Sigma(A) \arrow[r, "T_\Sigma(\gamma)"'] & T_\Sigma(B),
\end{tikzcd}$$

with ξ and χ generic, we have $\gamma \in {}^{\boxslash}(\mathscr{S}^{\boxslash})$.

10 Familiality for monads

We have now almost proved:

Lemma 10.1. *T_Σ preserves functional bisimulations.*

The only remaining bit is the hole we left in the proof of Lemma 5.1, when we claimed that all naturality squares of μ were pullbacks. Let us prove this now, as part of the following upgrade of Definition 7.3 and Lemma 7.4.

Definition 10.2. A monad is *familial* when its underlying functor is, and its unit and multiplication are *cartesian* natural transformations, i.e., their naturality squares are pullbacks.

In a case like ours, where the underlying category has a terminal object, by the pullback lemma, it is sufficient to verify that squares of the following form are pullbacks.

$$\begin{tikzcd}
T_\Sigma^2(X) \arrow[r, "T_\Sigma^2(!)"] \arrow[d, "\mu_X"'] \arrow[dr, phantom, "\lrcorner", very near start] & T_\Sigma^2(1) \arrow[d, "\mu_1"] \\
T_\Sigma(X) \arrow[r, "T_\Sigma(!)"'] & T_\Sigma(1)
\end{tikzcd}
\qquad
\begin{tikzcd}
X \arrow[r, "T_\Sigma^2(!)"] \arrow[d, "\eta_X"'] \arrow[dr, phantom, "\lrcorner", very near start] & 1 \arrow[d, "\eta_1"] \\
T_\Sigma(X) \arrow[r, "T_\Sigma(!)"'] & T_\Sigma(1)
\end{tikzcd}$$

The following will conclude our proof of congruence of bisimilarity:

Lemma 10.3. *T_Σ is a familial monad.*

Proof. Pullbacks in presheaf categories being pointwise, we just need to check that a few types of squares are pullbacks in **Set**: for μ and η, and for each type of label. Let us treat the most interesting one, namely the left one below, assuming that the right one has already been covered.

$$\begin{tikzcd}
T^2(X)[a] \arrow[r, "T^2(!)_{[a]}"] \arrow[d, "\mu_{X,[a]}"'] & T^2(1)[a] \arrow[d, "\mu_{1,[a]}"] \\
T(X)[a] \arrow[r, "T(!)_{[a]}"'] & T(1)[a]
\end{tikzcd}
\qquad
\begin{tikzcd}
T^2(X)(\star) \arrow[r, "T^2(!)_\star"] \arrow[d, "\mu_{X,\star}"'] & T^2(1)(\star) \arrow[d, "\mu_{1,\star}"] \\
T(X)(\star) \arrow[r, "T(!)_\star"'] & T(1)(\star)
\end{tikzcd}$$

Let $R[!]$ denote $T(!)(R)$ and $\mathbf{R}[\![!]\!]$ denote $T^2(!)(\mathbf{R})$, for all $R \in T(X)[a]$ and $\mathbf{R} \in T^2(X)[a]$. We must show that for any $a \in \mathbb{A}$, given any $\mathbf{R} \in T^2(1)[a]$ and $R \in T(X)[a]$ such that $R[!] = \mu_1(\mathbf{R})$, there exists a unique $\mathbf{R}^0 \in T^2(X)[a]$ satisfying

$$\mu_X(\mathbf{R}^0) = R \qquad \text{and} \qquad \mathbf{R}^0[\![!]\!] = \mathbf{R}.$$

We proceed by induction on $\mathbf{R}$. The base case is easy. For the induction step, if $\mathbf{R} = \rho(\mathbf{R}_{i,j})_{i\in n, j\in m_i}$, then because $\mu_1(\mathbf{R}) = R[!]$, we have $R = \rho(R_{i,j})_{i\in n,j\in m_i}$ with $R_{i,j}[!] = \mu_1(\mathbf{R}_{i,j})$ for all i, j[1]. By induction hypothesis, we find a family $\mathbf{R}^0_{i,j} \in T^2(X)[a_{i,j}]$ such that

$$\mu_X(\mathbf{R}^0_{i,j}) = R_{i,j} \qquad \text{and} \qquad \mathbf{R}^0_{i,j}[\![!]\!] = \mathbf{R}_{i,j} \qquad \text{for all } i, j.$$

Letting now $\mathbf{R}^0 = \rho(\mathbf{R}^0_{i,j})_{i\in n,j\in m_i}$, we get as desired

$$\mu_X(\mathbf{R}^0) = \rho(\mu_X(\mathbf{R}^0_{i,j}))_{i,j} = \rho(R_{i,j})_{i,j} = R \text{ and } \mathbf{R}^0[\![!]\!] = \rho(\mathbf{R}^0_{i,j}[\![!]\!])_{i,j} = \rho(\mathbf{R}_{i,j})_{i,j} = \mathbf{R}. \qquad \square$$

This ends the proof of:

Theorem 10.4. *For all $X \in \widehat{\Gamma_{\mathbb{A}}}$ and Positive GSOS specifications Σ, bisimilarity in the free algebra $T_\Sigma(X)$ is a congruence.*

[1]For all i such that $m_i = 0$, we in fact deal with some term M_i, using the corresponding square. Let us ignore this detail for readability.

11 Conclusion and perspectives

In this paper, we have introduced the familial approach to programming language theory [12] at the rather concrete level of generalised labelled transition systems ($\widehat{\Gamma_{\mathbb{A}}}$). Notably, we have recalled the notions of cellular monad and compositional algebra, and recalled that bisimilarity is always a congruence in a compositional algebra for a cellular monad (Lemma 4.3).

We have also shown that all monads T_Σ generated from a Positive GSOS specification Σ are cellular (Lemma 9.2) and that free T_Σ-algebras are always compositional (Lemma 5.1). Putting all three results together, we readily recover (Theorem 10.4) the known result that bisimilarity is a congruence for all free T_Σ-algebras. In particular, this is the case for the standard, syntactic transition system, which is the initial algebra $T_\Sigma(0)$.

This result constitutes a first generic tool for constructing instances of the framework of [12]. However, its scope is rather limited, and we plan to refine the construction to cover other formats like *tyft/tyxt* [11]. A striking and promising observation here is that the well-foundedness condition demanded of a tyft/tyxt specification for bisimilarity to be a congruence is clearly covered by our approach based on weak factorisation systems (see §9). Cellularity thus provides a semantic criterion for well-foundedness, whose precise relationship with the original, syntactic one seems worth investigating.

Beyond the task of showing by hand that existing formats yield cellular monads whose free algebras are compositional, we also plan to investigate a more categorical understanding of the generating process. The main motivation here is to design a construction that would cover variable binding. The theory developed by Fiore and his colleagues [7, 5] seems like a good starting point.

References

[1] B. Bloom, S. Istrail & A. Meyer (1995): *Bisimulation can't be traced*. Journal of the ACM 42, pp. 232–268, doi:10.1145/200836.200876.

[2] Filippo Bonchi, Daniela Petrisan, Damien Pous & Jurriaan Rot (2014): *Coinduction up-to in a fibrational setting*. In: Proc. 29th Symposium on Logic in Computer Science, ACM, pp. 20:1–20:9, doi:10.1145/2603088.2603149.

[3] Aurelio Carboni & Peter Johnstone (1995): *Connected Limits, Familial Representability and Artin Glueing*. Mathematical Structures in Computer Science 5(4), pp. 441–459, doi:10.1017/S0960129500001183.

[4] Andrea Corradini, Reiko Heckel & Ugo Montanari (2002): *Compositional SOS and beyond: a coalgebraic view of open systems*. Theoretical Computer Science 280(1-2), pp. 163–192, doi:10.1016/S0304-3975(01)00025-1.

[5] Marcelo Fiore & Chung-Kil Hur (2009): *On the construction of free algebras for equational systems*. Theoretical Computer Science 410, pp. 1704–1729, doi:10.1016/j.tcs.2008.12.052.

[6] Marcelo P. Fiore (2000): *Fibred Models of Processes: Discrete, Continuous, and Hybrid Systems*. In: IFIP TCS, LNCS 1872, Springer, pp. 457–473, doi:10.1007/3-540-44929-9_32.

[7] Marcelo P. Fiore (2008): *Second-Order and Dependently-Sorted Abstract Syntax*. In: LICS, IEEE, pp. 57–68, doi:10.1109/LICS.2008.38.

[8] Marcelo P. Fiore & Sam Staton (2006): *A Congruence Rule Format for Name-Passing Process Calculi from Mathematical Structural Operational Semantics*. In: Proc. 21st Symposium on Logic in Computer Science, IEEE, pp. 49–58, doi:10.1109/LICS.2006.7.

[9] Marcelo P. Fiore & Daniele Turi (2001): *Semantics of Name and Value Passing*. In: Proc. 16th Symposium on Logic in Computer Science, IEEE, pp. 93–104, doi:10.1109/LICS.2001.932486.

[10] Richard H. G. Garner & Tom Hirschowitz (2018): *Shapely monads and analytic functors*. Journal of Logic and Computation 28(1), pp. 33–83, doi:10.1093/logcom/exx029.

[11] Jan Friso Groote & Frits Vaandrager (1992): *Structured Operational Semantics and Bisimulation as a Congruence*. Information and Computation 100, pp. 202–260, doi:10.1016/0890-5401(92)90013-6.

[12] Tom Hirschowitz (2019): *Familial monads and structural operational semantics*. PACMPL 3(POPL), pp. 21:1–21:28, doi:10.1145/3290334.

[13] Mark Hovey (1999): *Model Categories*. Mathematical Surveys and Monographs, Volume 63, AMS (1999) 63, American Mathematical Society, doi:10.1090/surv/063.

[14] André Joyal, Mogens Nielsen & Glynn Winskel (1993): *Bisimulation and open maps*. In: Proc. 8th Symposium on Logic in Computer Science, IEEE, pp. 418–427, doi:10.1109/LICS.1993.287566.

[15] Tom Leinster (2014): *Basic Category Theory*. Cambridge Studies in Advanced Mathematics 143, Cambridge University Press, doi:10.1017/CBO9781107360068.

[16] Saunders Mac Lane (1998): *Categories for the Working Mathematician*, 2nd edition. Graduate Texts in Mathematics 5, Springer, doi:10.1007/978-1-4757-4721-8.

[17] Saunders Mac Lane & Ieke Moerdijk (1992): *Sheaves in Geometry and Logic: A First Introduction to Topos Theory*. Universitext, Springer, doi:10.1007/978-1-4612-0927-0.

[18] Robin Milner (1980): *A Calculus of Communicating Systems*. LNCS 92, Springer, doi:10.1007/3-540-10235-3.

[19] MohammadReza Mousavi, Michel A. Reniers & Jan Friso Groote (2007): *SOS Formats and Meta-Theory: 20 Years After*. Theoretical Computer Science 373(3), pp. 238–272, doi:10.1016/j.tcs.2006.12.019.

[20] Marco Peressotti (2017): *Coalgebraic Semantics of Self-Referential Behaviours*. Ph.D. thesis, University of Udine, doi:10.13140/rg.2.2.26899.07203.

[21] Gordon D. Plotkin (1981): *A Structural Approach to Operational Semantics*. DAIMI Report FN-19, Computer Science Department, Aarhus University.

[22] Emily Riehl (2014): *Categorical Homotopy Theory*. New Mathematical Monographs 24, Cambridge University Press, doi:10.1017/CBO9781107261457.

[23] Davide Sangiorgi & Jan Rutten, editors (2011): *Advanced Topics in Bisimulation and Coinduction*. Cambridge Tracts in Theoretical Computer Science 52, Cambridge University Press, doi:10.1017/CBO9780511792588.

[24] Sam Staton (2008): *General Structural Operational Semantics through Categorical Logic*. In: Proc. 23rd Symposium on Logic in Computer Science, pp. 166–177, doi:10.1109/LICS.2008.43.

[25] Daniele Turi & Gordon D. Plotkin (1997): *Towards a Mathematical Operational Semantics*. In: Proc. 12th Symposium on Logic in Computer Science, pp. 280–291, doi:10.1109/LICS.1997.614955.

[26] Mark Weber (2007): *Familial 2-functors and parametric right adjoints*. Theory and Applications of Categories 18(22), pp. 665–732.

Comparing Process Calculi Using Encodings

Kirstin Peters
TU Berlin/TU Darmstadt

Encodings or the proof of their absence are the main way to compare process calculi. To analyse the quality of encodings and to rule out trivial or meaningless encodings, they are augmented with encodability criteria. There exists a bunch of different criteria and different variants of criteria in order to reason in different settings. This leads to incomparable results. Moreover, it is not always clear whether the criteria used to obtain a result in a particular setting do indeed fit to this setting. This paper provides a short survey on often used encodability criteria, general frameworks that try to provide a unified notion of the quality of an encoding, and methods to analyse and compare encodability criteria.

1 Introduction

Process calculi (or process algebra) is one area of formalisations of concurrent systems. Other areas are for example Petri nets [54] or the Actor model [25]. *A brief history of process algebra* can be found in [2]. Over the time different process calculi emerge. Examples are CCS [32], the π-calculus [36, 35], CSP [26], or ACP [4], just to name some of the most prominent ones. Note that each of these calculi denotes rather a family of process calculi. The number of different process calculi is in fact enormous. As discussed in [41] there are many good reasons for this great number of different calculi. The most plausible is maybe that many of these different calculi stem from different practical needs. They are designed as domain-specific calculi capturing exactly the set of primitives necessary to model the desired system at a proper level of abstraction without overloading the theory with (for this domain) unnecessary operations. The large number of calculi calls for methods to compare different calculi or different variants within a family of process calculi with respect to their expressive power. The most prominent such method is language embedding using encodings [5]. Encodings—or the proof of their absence—do not only allow to compare the expressive power of languages but also formalise similarities and differences between the considered languages. So they provide a base for implementations of languages into real systems.

There are basically two ways to measure the expressive power of a language [45].

An *absolute result* is a result about the expressive power of a single process calculus, usually obtained by proving the ability (*positive absolute result*) or inability (*negative absolute result*) to solve some kind of problem (see [45, 22] and even [31]), whereas a *relative result* compares two process calculi.

To *compare* the absolute expressive power of *two* languages, we may simply choose a problem that can be solved in one language, but not in the other language. Note that combining two absolute results that are both positive or both negative usually does not reveal much information, because it proves only that the considered languages do not differ with respect to the respective particular problem. Actually as soon as we compare two languages, it makes sense to use the term *relative expressive power*, as we can now relate the two languages. The terminology of the relative expressive power has also been attributed (see [45, 22]) to the comparison of the expressive power of two languages by means of the existence or non-existence of encodings from one language into the other language, subject to various conditions on the encoding. Indeed, encodings are the main way to relate the expressive power of two process calculi.

A positive relative result, i.e., the proof of the existence of an encoding, is denoted as *encodability result*, whereas a negative result is denoted as *separation result*.

Language comparison by means of encodings is a wide area of research within the context of process calculi. Reasonable and meaningful encodings from one language into another show that the latter is at least as expressive as the former, whereas the absence of such an encoding shows that the former can express some behaviour that is not expressible in the latter, i.e., reveals a difference in the expressive power of the former compared to the latter language. To analyse the quality of encodings and to rule out trivial or meaningless encodings, they are augmented with encodability criteria. However, as stated several times in literature (e.g. in [43, 41, 45, 22]), there is no agreement on what set of criteria makes an encoding reasonable and meaningful. Sometimes it is even stated that such an agreement may not exist or may not be desirable (see e.g. [43]), because many criteria result from different practical needs. They are often derived from the main purpose of the current analysis. From a practical point of view this is meaningful. But, obviously, using different quality criteria for different results, because they were motivated by different practical problems, naturally leads to incomparable results.

After some technical preliminaries in Section 2, Section 3 provides a short survey of often used encodability criteria. This section is the heart of this paper. Then, Section 4 shortly outlines three approaches that try to overcome the problem of incomparable results and provide a unified notion of the quality of an encoding. Similarly, Section 5 shortly outlines the only way to formally analyse and compare encodability criteria we are aware of. Finally, Section 6 raises some open questions. Note that this paper provides a rather abstract view on encodings and focuses on encodability criteria. A more practical study of encodings, including a discussion of how to obtain an encoding and prove it correct as well as some actual examples of encodings can be found in [49].

2 Process Calculi and Encodings

A *process calculus* is a language $\mathscr{L}_C = (\mathscr{P}_C, \longmapsto_C)$ consisting of a set of terms $\mathscr{P}_C$—its *syntax*—and a relation on terms $\longmapsto_C \subseteq \mathscr{P}_C \times \mathscr{P}_C$—its *semantics*. The elements of $\mathscr{P}_C$ are called *process terms* or shortly processes or terms. We use upper case letters $P, Q, S, T, \ldots$ to range over process terms.

The *syntax* is usually defined by a context-free grammar defining operators. An *operator* is a function from sorts and process terms into a process term. Sorts can be used to introduce data values or structural information such as location names or identities. For simplicity, we restrict our attention to a single such sort. Assume a countably-infinite set $\mathscr{N}$, whose elements are called *names*. Names are the universe of elements of which the processes are constructed within many process calculi such as e.g. in the π-calculus. Then an operator is a function $\mathrm{op} : \mathscr{N}^n \times \mathscr{P}_C^m \to \mathscr{P}_C$ from names and process terms into a process term. In this case, we say op has the *arity m*. Sometimes, process calculi also specify operators that, instead of a fixed number of arguments, accept any finite set of names and/or process terms usually indexed by a finite index set I. In this case, the arity of the operator is not a fixed value but, for a given set of arguments, is determined by the number of process terms among the arguments. An example of such an operator is given by the choice operator in the π-calculus. Operators of arity 0 are denoted as *constants*.

The semantics of a process calculus can be given as a reduction semantics—that describes the interaction within a system of processes—or a labelled semantics—that also describes the interactions of such a system with its environment. Here we assume that the semantics of the language is provided as a reduction semantics, because in the context of encodings the treatment of reductions is simpler. A *step* $P \longmapsto_C P'$ is an element $(P, P') \in \longmapsto_C$. Let $P \longmapsto_C$ denote the existence of a step from P, i.e.,

$\exists P' \in \mathscr{P}_C .\, P \longmapsto_C P'$, let $P \not\longmapsto_C$ denote the absence of a step from P, i.e., $\neg (P \longmapsto_C)$, and let $\Longmapsto_C$ denote the reflexive and transitive closure of $\longmapsto_C$. We write $P \longmapsto_C{}^{\omega}$ if P can do an infinite sequence of steps. A term P such that $P \longmapsto_C{}^{\omega}$ is called *divergent*.

Intuitively, a *context* $\mathscr{C}([\cdot]_1, \ldots, [\cdot]_n) : \mathscr{P}_C^n \to \mathscr{P}_C$ is a process term with n holes. Formally, it is a function from n process terms into a process term, i.e., given n terms $P_1, \ldots, P_n \in \mathscr{P}_C$, the term $\mathscr{C}(P_1, \ldots, P_n)$ is the result of inserting the n terms $P_1, \ldots, P_n$ in that order into the n holes of $\mathscr{C}$. We also consider contexts $\mathscr{C}([\cdot]_1, \ldots, [\cdot]_n, [\cdot]_{n+1}, \ldots, [\cdot]_{n+m}) : \mathscr{N}^n \times \mathscr{P}_C^m \to \mathscr{P}_C$ that have holes for names and terms.

A typical operator is the restriction of scopes of names. A *scope* defines an area in which a particular name is known and can be used. For several reasons, it can be useful to restrict the scope of a name. For instance to forbid interactions between two processes or with an unknown and, hence, potentially untrusted environment. Names whose scope is restricted such that they cannot be used from outside the scope are denoted as *bound names*. The remaining names are called *free names*. Accordingly, we assume three sets—the sets of names $\mathsf{n}(P)$ and its subsets of free names $\mathsf{fn}(P)$ and bound names $\mathsf{bn}(P)$—for each term P. In the case of bound names, their syntactical representation as lower case letter serves as a place holder for any fresh name, i.e., any name that does not occur elsewhere in the term. To avoid name capture or clashes, i.e., to avoid confusion between free and bound names or different bound names, bound names can be mapped to fresh names by *α-conversion*. We write $P \equiv_\alpha Q$ if P and Q differ only by α-conversion. A *substitution* σ is a finite mapping from names to names defined by a set $\{ {}^{y_1}/_{x_1}, \ldots, {}^{y_n}/_{x_n} \}$ of renamings, where the $x_1, \ldots, x_n$ are pairwise distinct. The application of a substitution to a term $P\{ {}^{y_1}/_{x_1}, \ldots, {}^{y_n}/_{x_n} \}$ is defined as the result of simultaneously replacing all free occurrences of x_i by y_i for $i \in \{1, \ldots, n\}$, possibly applying α-conversion to avoid capture or name clashes. For all names $\mathscr{N} \setminus \{x_1, \ldots, x_n\}$ the substitution behaves as the identity mapping.

To compare process terms, process calculi usually come with different well-studied preorders and equivalences (see [16, 14] for an overview and a classification of the most frequent equivalences). A special kind of equivalence with great importance to reason about processes are congruences. A congruence is the closure of an equivalence with respect to contexts, i.e., an equivalence $\mathscr{R} \subseteq \mathscr{P}_C \times \mathscr{P}_C$ is a *congruence* if $(P, Q) \in \mathscr{R}$ implies $(\mathscr{C}(P), \mathscr{C}(Q)) \in \mathscr{R}$ for all terms $P, Q \in \mathscr{P}_C$ and all contexts $\mathscr{C}([\cdot]) : \mathscr{P}_C \to \mathscr{P}_C$. Moreover, let C be a set of $\mathscr{P}_C \to \mathscr{P}_C$-contexts, then $\mathscr{R} \subseteq \mathscr{P}_C \times \mathscr{P}_C$ is a *congruence with respect to C* if $(P, Q) \in \mathscr{R}$ implies $(\mathscr{C}(P), \mathscr{C}(Q)) \in \mathscr{R}$ for all terms $P, Q \in \mathscr{P}_C$ and all contexts $\mathscr{C} \in C$. Moreover, process calculi usually come with a special congruence $\equiv_C \subseteq \mathscr{P}_C \times \mathscr{P}_C$ called *structural congruence*. Its main purpose is to equate syntactically different process terms that model quasi-identical behaviour.

Of special interest are simulation relations; in particular bisimulations [57]. $\mathscr{R}$ is a bisimulation if any two related processes mutually simulate their respective sequences of steps, such that the derivatives are again related.

Definition 2.1 (Bisimulation) *$\mathscr{R}$ is a* (weak reduction) bisimulation *if for each $(P, Q) \in \mathscr{R}$:*

- *$P \Longmapsto_C P'$ implies $\exists Q' .\, Q \Longmapsto_C Q' \wedge (P', Q') \in \mathscr{R}$*
- *$Q \Longmapsto_C Q'$ implies $\exists P' .\, P \Longmapsto_C P' \wedge (P', Q') \in \mathscr{R}$*

Two terms are bisimilar *if there exists a bisimulation that relates them.*

The definition of a *strong (reduction) bisimulation* is obtained by replacing all $\Longmapsto_C$ by $\longmapsto_C$ in the above definition, i.e., a strong bisimulation requires that a step has to be simulated by a single step. Coupled similarity (defined below) is strictly weaker than bisimilarity. As pointed out in [47], in contrast to bisimilarity it allows for intermediate states in simulations: states that cannot be identified with states of the simulated term. Each symmetric coupled simulation is a bisimulation.

Definition 2.2 (Coupled Simulation) *A relation $\mathscr{R}$ is a* (weak reduction) coupled simulation *if, for each $(P,Q) \in \mathscr{R}$ with $P \Longmapsto_C P'$, $(\exists Q'.\, Q \Longmapsto_C Q' \wedge (P',Q') \in \mathscr{R})$ and $(\exists Q'.\, Q \Longmapsto_C Q' \wedge (Q',P') \in \mathscr{R})$. Two terms are* coupled similar *if they are related by a coupled simulation in both directions.*

An *encoding* from $\mathscr{L}_{\mathrm{S}} = (\mathscr{P}_{\mathrm{S}}, \longmapsto_{\mathrm{S}})$ into $\mathscr{L}_{\mathrm{T}} = (\mathscr{P}_{\mathrm{T}}, \longmapsto_{\mathrm{T}})$ relates two process calculi. We call $\mathscr{L}_{\mathrm{S}}$ the *source* and $\mathscr{L}_{\mathrm{T}}$ the *target language*. Accordingly, terms of $\mathscr{P}_{\mathrm{S}}$ are *source terms* and of $\mathscr{P}_{\mathrm{T}}$ *target terms*. In the simplest case an encoding from $\mathscr{L}_{\mathrm{S}}$ into $\mathscr{L}_{\mathrm{T}}$ is an *encoding function* $[\![\cdot]\!] : \mathscr{P}_{\mathrm{S}} \to \mathscr{P}_{\mathrm{T}}$ from source terms into target terms. Sometimes an encoding is defined by several functions, such as the encoding function and the renaming policy (see below). Else we identify an encoding with its encoding function.

Sometimes it is necessary to translate a source term name into a sequence of names or reserve some names for the encoding function. To ensure that there are no conflicts between these reserved names and the source term names, [22] equips an encoding with a *renaming policy* φ, i.e., a substitution from names into sequences of pairwise disjoint names. To keep distinct names distinct, [22] assumes that the sequences of names that result from applying a renaming policy to distinct names have no common name. Moreover, if the renaming policy translates a single name into a sequence of names then the length of such a sequence has to be the same for all names, such that the encoding can not distinguish between different source term names by the length of the sequences to which they are encoded. Obviously, no name should be translated into an infinite sequence of names.

Definition 2.3 (Renaming Policy) *A substitution $\varphi : \mathscr{N} \to \mathscr{N}^n$ from names into sequences of pairwise disjoint names is a renaming policy, if $\forall x,y \in \mathscr{N}.\, x \neq y$ implies $\varphi(x) \cap \varphi(y) = \emptyset$, where $\varphi(z)$ is considered as a set.*

Note that the renaming policy allows us to use the names reserved by the encoding like implicit parameters. It is for instance possible that some part of the encoding introduces a free occurrence of a reserved name within the encoding of a subterm which is bound by the surrounding part of the encoding. Examples can be found in [22, 49]. Accordingly, in [22] an encoding is a pair $([\![\cdot]\!], \varphi(\cdot))$.

An *encodability criterion* is a predicate on encoding functions, used to reason about the quality of encodings. To simplify the presentation we assume henceforth that $\mathscr{P}_{\mathrm{S}} \cap \mathscr{P}_{\mathrm{T}} = \emptyset$ and thus $\mathscr{P}_{\mathrm{S}} \uplus \mathscr{P}_{\mathrm{T}} = \mathscr{P}_{\mathrm{S}} \cup \mathscr{P}_{\mathrm{T}}$. We say that a condition $\mathsf{P} : (\mathscr{P}_{\mathrm{S}} \uplus \mathscr{P}_{\mathrm{T}}) \to \mathbb{B}$ is *preserved* by an encoding if for all source terms S that satisfy P, the condition P also holds for $[\![S]\!]$. A condition is *reflected* by an encoding if whenever $[\![S]\!]$ satisfies it, then so does S. Finally an encoding *respects* a condition if it both preserves and reflects it.

3 Encodability Criteria

Encodings are used to compare process calculi and to reason about their expressive power. Encodability criteria are conditions that limit the existence of encodings. Their main purpose is to rule out trivial or meaningless encodings, but they can also be used to limit attention to encodings that are of special interest in a particular domain or for a particular purpose. These quality criteria are the main tool in *separation results*, saying that one calculus is not expressible in another one; here one has to show that no encoding meeting these criteria exists. To obtain stronger separation results, care has to be taken in selecting quality criteria that are not too restrictive. For *encodability results*, saying that one calculus is expressible in another one, all one needs is an encoding, together with criteria testifying for the quality of the encoding. Here it is important that the criteria are not too weak.

In the literature various different criteria and different variants of the same criteria are employed to achieve separation and encodability results [15, 40, 42, 43, 44, 41, 59, 9, 45, 8, 22, 52, 17]. Some criteria, like full abstraction or operational correspondence, are used frequently. Other criteria are used to enforce a property of encodings that might only be necessary within a certain domain. For instance, the homomorphic translation of the parallel operator—in general a rather strict criterion—was used in [43] to show the absence of an encoding from the synchronous into the asynchronous π-calculus, because this requirement forbids for the introduction of global coordinators. Thus this criterion is useful when reasoning about the concurrent behaviour of processes, although it is in general too strict to reason about their interleaving behaviour. Unfortunately, it is not always obvious or clear whether the criteria used to obtain a result in a particular setting do indeed fit to this setting. Indeed, as discussed in [52], the homomorphic translation of the parallel operator forbids more than global coordinators, i.e., is too strict even in a concurrent setting (compare to Section 3.6).

In the following we shortly revise some of the most commonly used encodability criteria. In [39] also a class of rather quantitative criteria for the effectiveness or efficiency of an encoding is discussed. The efficiency of an encoding can be measured for example by a criterion to count the number of messages or steps of an emulation [7, 1, 29]. However, here we are more focused on semantic and structural criteria.

3.1 Direct Comparison via a Relation

The least debatable criterion is the direct comparison of the source and the target language by a behavioural equivalence (or preorder). This criterion requires that $\forall S \in \mathscr{P}_S.\ [\![S]\!] \mathscr{R} S$ holds, for some behavioural equivalence (or preorder) $\mathscr{R} \subseteq (\mathscr{P}_S \uplus \mathscr{P}_T) \times (\mathscr{P}_S \uplus \mathscr{P}_T)$ that is either defined in exactly the same way on the source and the target language (as e.g. in [45]) or explicitly distinguishes between source and target terms (as e.g. in [50]). Intuitively, it is required that the encoding respects the semantics of the source modulo the chosen relation. Obviously, the quality of the encoding directly depends on the choice of $\mathscr{R}$. A stricter such relation leads to stricter requirements on the encoding function. It is also very clear in this case under which circumstances two different results can be compared. If both results are proven with respect to the same relation then the results can be compared directly. If one of the considered relations is strictly weaker then the results can be compared with respect to the weaker relation. Else, if the relations are incomparable, also the results are incomparable. Hence, a language $\mathscr{L}_1$ can be considered as strictly weaker than the language $\mathscr{L}_2$ with respect to $\mathscr{R}$, if there is an encoding from $\mathscr{L}_1$ into $\mathscr{L}_2$ that satisfies the above requirement but there is no such encoding from $\mathscr{L}_2$ into $\mathscr{L}_1$.

Of course, there may be still some debate on how to choose the relation $\mathscr{R}$. For instance neither the identity nor $\mathscr{R} = (\mathscr{P}_S \uplus \mathscr{P}_T) \times (\mathscr{P}_S \uplus \mathscr{P}_T)$ are intuitively meaningful choices. Moreover, as shown in [16, 14] there are usually very many potential candidates. Variants of bisimulation are usually a rather strong candidate suitable for encodability results, whereas weaker candidates such as trace equivalence are better suited for separation results. For a congruence it is sometimes necessary to restrict the set of considered contexts to contexts that respect the protocol behind the encoding (see e.g. [58, 18]). However, since the choice of the equivalence directly monitors the requirements on the encoding function, this problem is not that serious. There are, however, two serious drawbacks of this criterion. The first is the requirement that $\mathscr{R}$ is either defined in exactly the same way on the source and the target language or explicitly distinguishes between source and target terms. Usually behavioural relations are designed for one specific language, so it is not clear how to define a suitable relation if the two languages have no standard behavioural equivalence (or preorder) in common. A standard candidate for such an equivalence is weak reduction bisimulation, that is defined similarly on all calculi. Unfortunately, the variant of weak bisimulation given in Definition 2.1 is trivial. We need to add a condition that compares the observables of bisimilar states in order to obtain a meaningful relation. But, since the source and target language might specify different standard observables, this equivalence (if it can be constructed at all) will usually be very complex and unreadable. Note that [22] solves this problem by using a special observable called success instead of the standard observables of the languages, whereas [12, 18] use relations to compare observables of different languages. As a consequence, if $\mathscr{R}$ is not a standard equivalence, the direct comparison of source and target terms modulo $\mathscr{R}$ reveals less intuition on the encoding function. The second problem is that a complex $\mathscr{R}$ leads to a hard proof of this criterion. If $\mathscr{R}$ is not a standard equivalence, most of the standard techniques that would ease such a proof may not be applicable.

3.2 Full Abstraction

Whenever source and target can not be compared directly with respect to a standard equivalence, full abstraction might be a way to nonetheless use standard equivalences. *Full abstraction*—denoted as *observational correspondence* in [13]—is probably the most common quality criterion for language comparison. It is used for instance in [55, 38, 56, 60, 42, 3, 48], just to name some. Full abstraction as proof method for language comparison was adapted from the use of full abstraction to show correspondence between a denotational semantics of a program and its operational semantics. An encoding $[\![\cdot]\!]$ is fully abstract if

$$\forall S_1, S_2 \in \mathscr{P}_\mathrm{S}.\, S_1 \mathscr{R}_\mathrm{S} S_2 \quad \text{iff} \quad [\![S_1]\!] \mathscr{R}_\mathrm{T} [\![S_2]\!]$$

for two behavioural equivalences $\mathscr{R}_\mathrm{S} \subseteq \mathscr{P}_\mathrm{S} \times \mathscr{P}_\mathrm{S}$ and $\mathscr{R}_\mathrm{T} \subseteq \mathscr{P}_\mathrm{T} \times \mathscr{P}_\mathrm{T}$, i.e., full abstraction requires that equivalent source terms have to be mapped into equivalent target terms and vice versa. Note that the direction from the left to the right is often called soundness condition and the only if part completeness condition of full abstraction. The soundness condition is usually the most demanding part. Note that some well-known and widely accepted encodings, as e.g. [6, 27, 33, 34], do not satisfy this property with respect to a reasonable combination of standard equivalences. The main advantage of full abstraction is its wide applicability also with respect to (more or less) standard equivalences. It does e.g. not require that source and target share any notion of observable, which is a premise for the use of most of the standard equivalences in the criterion above. However, again there may be a very large number of equivalences on the source as well as equivalences on the target and the strictness of the property expressed by full abstraction strongly relies on the combination of the chosen equivalences. To reduce the strong dependence of full abstraction results on the chosen equivalences, full abstraction is often combined with operational correspondence. In [13] it is even stated that full abstraction is not of much use without operational correspondence. The two papers [23, 46] come to a similar conclusion after discussing potential pitfalls and misunderstandings w.r.t. full abstraction. These papers also hint to earlier such discussions. The possibility to chose a combination of standard equivalence that turn full abstraction trivial is a major drawback of this criterion. Another major drawback is that, because of the various possibilities to choose these two equivalences, it is often not possible to compare different full abstraction results.

3.3 Operational Correspondence

Intuitively, *operational correspondence* requires executions to be respected. Again, it consists of a completeness and a soundness part. The completeness condition, also called adequacy, requires that for all source term steps $S \longmapsto_\mathrm{S} S'$ or source term executions $S \Longmapsto_\mathrm{S} S'$ there is one emulation in the target language such that $[\![S]\!] \Longmapsto_\mathrm{T} \asymp_\mathrm{T} [\![S']\!]$, where $\asymp_\mathrm{T} \subseteq \mathscr{P}_\mathrm{T} \times \mathscr{P}_\mathrm{T}$ is some equivalence on the target language. Note that there is no difference in the consideration of single source term steps or source term executions.

Intuitively, the completeness condition requires that any source term execution is emulated by the target term modulo some equivalence $\asymp_{\mathrm{T}}$. Again, completeness is usually the easier part.

For the soundness condition we basically find two formulations. The stricter formulation requires that for all executions of the target $[\![S]\!] \Longmapsto_{\mathrm{T}} T$ there exists some execution of the source $S \Longmapsto_{\mathrm{S}} S'$ such that $[\![S']\!] \asymp_{\mathrm{T}} T$. Intuitively, soundness requires that whatever $[\![S]\!]$ can do is a translation of some behaviour of S modulo $\asymp_{\mathrm{T}}$. The weaker formulation requires that for all executions of the target $[\![S]\!] \Longmapsto_{\mathrm{T}} T$ there exists some execution of the source $S \Longmapsto_{\mathrm{S}} S'$ and some execution of the target $T \Longmapsto_{\mathrm{T}} T'$ such that $[\![S']\!] \asymp_{\mathrm{T}} T'$. Intuitively, it states that any execution of the target is some part of the emulation of an execution in the source modulo $\asymp_{\mathrm{T}}$ [47, 22]. The main difference is that the latter formulation allows for intermediate or partial commitment states [47, 49, 24], i.e., for states that do not need to be related directly to the states of the respective source term but that have to belong to some emulation of a source term step. In this sense, an intermediate state results from the partial emulation of a source term step. In particular the last variant, proposed in [22], was used for numerous encodability and separation results.

Definition 3.1 (Operational Correspondence) *An encoding* $[\![\cdot]\!] : \mathscr{P}_{\mathrm{S}} \to \mathscr{P}_{\mathrm{T}}$ *is* strongly operationally corresponding *w.r.t.* $\asymp_{\mathrm{T}} \subseteq \mathscr{P}_{\mathrm{T}}^2$ *if it is:*

Strongly Complete: $\forall S, S'.\ S \longmapsto_{\mathrm{S}} S'$ *implies* $(\exists T.\ [\![S]\!] \longmapsto_{\mathrm{T}} T \wedge ([\![S']\!], T) \in \asymp_{\mathrm{T}})$

Strongly Sound: $\forall S, T.\ [\![S]\!] \longmapsto_{\mathrm{T}} T$ *implies* $(\exists S'.\ S \longmapsto_{\mathrm{S}} S' \wedge ([\![S']\!], T) \in \asymp_{\mathrm{T}})$

$[\![\cdot]\!] : \mathscr{P}_{\mathrm{S}} \to \mathscr{P}_{\mathrm{T}}$ *is* operationally corresponding *w.r.t.* $\asymp_{\mathrm{T}} \subseteq \mathscr{P}_{\mathrm{T}}^2$ *if it is:*

Complete: $\forall S, S'.\ S \Longmapsto_{\mathrm{S}} S'$ *implies* $(\exists T.\ [\![S]\!] \Longmapsto_{\mathrm{T}} T \wedge ([\![S']\!], T) \in \asymp_{\mathrm{T}})$

Sound: $\forall S, T.\ [\![S]\!] \Longmapsto_{\mathrm{T}} T$ *implies* $(\exists S'.\ S \Longmapsto_{\mathrm{S}} S' \wedge ([\![S']\!], T) \in \asymp_{\mathrm{T}})$

$[\![\cdot]\!] : \mathscr{P}_{\mathrm{S}} \to \mathscr{P}_{\mathrm{T}}$ *is* weakly operationally corresponding *w.r.t.* $\asymp_{\mathrm{T}} \subseteq \mathscr{P}_{\mathrm{T}}^2$ *if it is:*

Complete: $\forall S, S'.\ S \Longmapsto_{\mathrm{S}} S'$ *implies* $(\exists T.\ [\![S]\!] \Longmapsto_{\mathrm{T}} T \wedge ([\![S']\!], T) \in \asymp_{\mathrm{T}})$

Weakly Sound: $\forall S, T.\ [\![S]\!] \Longmapsto_{\mathrm{T}} T$ *implies* $(\exists S', T'.\ S \Longmapsto_{\mathrm{S}} S' \wedge T \Longmapsto_{\mathrm{T}} T' \wedge ([\![S']\!], T') \in \asymp_{\mathrm{T}})$

Again different variants of operational correspondence may arise from different requirements on the assumed equivalence $\asymp_{\mathrm{T}}$ on the target language and a wrong choice might turn operational correspondence trivial. In particular, each encoding is operational corresponding w.r.t. the universal relation on target terms.

In [13] a version of operational correspondence is presented, that considers labelled steps instead of a reduction semantics under the assumption that there exists a mapping $\widehat{\cdot}$ from the labels of the source term into the labels of the target term. Hence, the resulting requirement—$[\![S]\!] \xRightarrow{\widehat{\lambda}}_{\mathrm{T}} \asymp_{\mathrm{T}} [\![S']\!]$ whenever $S \xRightarrow{\lambda}_{\mathrm{S}} S'$ and $[\![S]\!] \xRightarrow{\lambda}_{\mathrm{T}} T$ implies $S \xRightarrow{\lambda'}_{\mathrm{S}} S'$ for some λ', S' such that $[\![S']\!] \asymp_{\mathrm{T}} T$ and $\widehat{\lambda'} = \lambda$—can be considered as stricter than the above variant of operational correspondence, because also observables have to be respected modulo $\widehat{\cdot}$. In fact, without this strengthening to labelled semantics, operational correspondence alone can hardly be considered as suitable criterion. Hence, the above version based on reduction semantics is usually combined with other criteria as full abstraction or some requirements on the preservation or reflection of some kind of observable.

3.4 Observables, Testing, and Termination

If source and target terms can not be compared directly by a standard equivalence, e.g. because not all standard observables of the source are standard observables of the target, a natural weaker requirement is to consider preservation or reflection of the remaining observables that are shared by source and target. Typical observables are links used for communication, barbs (communication capabilities), or traces [59, 45]. Moreover, the use of *termination properties* as the possibility of deadlock, livelock, or divergence are popular [40, 45, 22, 13]. Also all kind of *tests* that the process may (or must) pass, for some formal notion of test can be used to compare source and target behaviour [45, 22].

Another kind of termination property is the promptness condition. Intuitively, promptness ensures that an encoding does not introduce preprocessing steps. An encoding $[\![\cdot]\!]$ is *prompt*, if $S \not\longmapsto_{\mathrm{S}}$ implies $[\![S]\!] \not\longmapsto_{\mathrm{T}}$ for all source terms $S \in \mathscr{P}_{\mathrm{S}}$.

In [22] the criterion *success sensitiveness* was proposed. An encoding is success sensitive if it respects reachability of a particular process $\checkmark$ that represents successful termination, or some other form of success, and is added to the syntax of the source as well as the target language. Since $\checkmark$ cannot be further reduced and $\mathsf{n}(\checkmark) = \mathsf{fn}(\checkmark) = \mathsf{bn}(\checkmark) = \emptyset$, the semantics and structural congruence of a process calculus are not affected by this additional constant operator. We write $P\downarrow_{\checkmark}$ to denote the fact that P is successful—however this predicate might be defined in the particular source or target language. Reachability of success is then defined as $P\Downarrow_{\checkmark} \triangleq \exists P'.\, P \Longmapsto P' \wedge P'\downarrow_{\checkmark}$. We use may-testing here, but alternatively e.g. must-testing or fair-testing can also be implemented. An encoding is success sensitive if each source term and its translation answer the test for reachability of success in the same way.

Definition 3.2 (Success Sensitiveness) *Let $\mathscr{L}_{\mathrm{S}}$ and $\mathscr{L}_{\mathrm{T}}$ each define a predicate $\cdot\downarrow_{\checkmark}\colon \mathscr{P} \to \mathbb{B}$. An encoding $[\![\cdot]\!] : \mathscr{P}_{\mathrm{S}} \to \mathscr{P}_{\mathrm{T}}$ is* success sensitive *if, for all $S \in \mathscr{P}_{\mathrm{S}}$, $S\Downarrow_{\checkmark}$ iff $[\![S]\!]\Downarrow_{\checkmark}$.*

Note that $\checkmark$ can be considered as some kind of termination property—describing successful termination in contrast to not successful termination by a term that cannot reduce but is not $\checkmark$—or as the successful pass of some kind of test.

If the source and target language share standard observables, we can easily extend success sensitiveness to barb sensitiveness. Assume that, instead of $\cdot\downarrow_{\checkmark}\colon \mathscr{P} \to \mathbb{B}$, the languages $\mathscr{L}_{\mathrm{S}}$ and $\mathscr{L}_{\mathrm{T}}$ each define a predicate $\cdot\downarrow\cdot : \mathscr{P} \times \mathscr{B} \to \mathbb{B}$. An encoding $[\![\cdot]\!] : \mathscr{P}_{\mathrm{S}} \to \mathscr{P}_{\mathrm{T}}$ *strongly respects barbs* if, for all $S \in \mathscr{P}_{\mathrm{S}}$ and all $\alpha \in \mathscr{B}$, $S\downarrow\alpha$ iff $[\![S]\!]\downarrow\alpha$. A weak variant is obtained by replacing the predicate $\cdot\downarrow\cdot$ for the existence of a barb by a predicate $\cdot\Downarrow\cdot$ that is true if a barb is reachable.

3.5 Structural Requirements

The above discussed criteria describe semantic requirements, i.e., requirements on the behaviour of target terms. To prove the quality of an encoding semantic criteria are often combined with structural criteria. Intuitively, the semantic criteria describe how the encoded terms should behave with respect to the behaviour of the corresponding source term, whereas structural criteria rather describe how the encoded terms have to look like. Moreover, as stated in [45], structural criteria are needed in order to measure expressiveness of operators in contrast to expressiveness of terms.

The most common structural criterion is compositionality with homomorphy as a special case. Intuitively, *compositionality* states that the translation of a compound term must be defined in terms of the translation of the subterms. To mediate between translations of subterms, a context is introduced. Different manifestations result from different requirements on allowed contexts. In the strictest form, often denoted as *homomorphy*, the context has to be the original source term operator again, i.e., an encoding translates the source term operator $\mathrm{op}\,(x_1, \ldots, x_n, S_1, \ldots, S_m)$ homomorphically if it ensures that $[\![\mathrm{op}\,(x_1, \ldots, x_n, S_1, \ldots, S_m)]\!] = \mathrm{op}\,(x_1, \ldots, x_n, [\![S_1]\!], \ldots, [\![S_m]\!])$ holds for all $x_1, \ldots, x_n \in \mathscr{N}$ and all $S_1, \ldots, S_m \in \mathscr{P}_{\mathrm{S}}$. Of course, homomorphy requires that the respective operator is part of the source and the target language. Because of this, homomorphy is often required only for specific common operators such as the parallel operator, because it occurs more or less with the same meaning and comparable syntax in most of the process calculi. If we assume that the parallel operator is a binary operator in the source and the target language, the homomorphic translation of the parallel operator means $[\![S_1 \mid S_2]\!] = [\![S_1]\!] \mid [\![S_2]\!]$ for all $S_1, S_2 \in \mathscr{P}_{\mathrm{S}}$, where $\mid$ is the syntactical representation of the parallel operator. If single names are translated into single names, it makes sense to extend the notion of homomorphic translation to the use of a renaming policy. Thus, if the encoding translates an operator $\mathrm{op}\,(x_1, \ldots, x_n, S_1, \ldots, S_m)$ always into $\mathrm{op}\,(\phi(x_1), \ldots, \phi(x_n), [\![S_1]\!], \ldots, [\![S_m]\!])$, we call the translation of this operator again homomorphic.

Homomorphic translations of operators are e.g. used to analyse the expressive power of a single operator. To show for instance that a set of operators is not minimal the existence of an encoding is analysed that translates all operators homomorphically except for the operator that should be removed. Moreover, homomorphy is a very nice property, because it significantly eases the proof of the correctness of the encoding function. Basically, the homomorphic translation of an operator ensures that for this operator nothing is to show, because in this point the encoding obviously preserves and reflects all properties of that operator. However, even in case the respective operator is part of the source as well as the target language, homomorphy is a very strict requirement. Intuitively, it states that the encoding function is not allowed to touch the respective operator and, hence, is not allowed to simulate its behaviour by some protocol. Such translations are possible only if the compared languages are very close (at least with respect to this operator).

[43, 22, 49] show that not even between calculi that are so close as the full π-calculus (with mixed choice) and its subcalculus with only separate choice an encoding that translates the parallel operator homomorphically exists. Instead, often compositionality is required. Intuitively, an encoding is compositional if the translation of an operator is the same for all occurrences of that operator in a term. Hence, the translation of that operator can be captured by a context that is allowed in [22] to be parametrised on the free names of the respective source term.

Definition 3.3 (Compositionality) *The encoding* $[\![\cdot]\!]$ *is* compositional *if, for every operator* $\mathrm{op} : \mathscr{N}^n \times \mathscr{P}_{\mathrm{S}}^m \to \mathscr{P}_{\mathrm{S}}$ *and for every subset of names N, there exists a context* $\mathscr{C}_{\mathrm{op}}^N([\cdot]_1, \ldots, [\cdot]_{n+m}) : \mathscr{N}^n \times \mathscr{P}_{\mathrm{S}}^m \to \mathscr{P}_{\mathrm{T}}$ *such that, for all* $x_1, \ldots, x_n \in \mathscr{N}$ *and all* $S_1, \ldots, S_m \in \mathscr{P}_{\mathrm{S}}$ *with* $\mathsf{fn}(S_1) \cup \ldots \cup \mathsf{fn}(S_m) = N$*, it holds that* $[\![\mathrm{op}\,(x_1, \ldots, x_n, S_1, \ldots, S_m)]\!] = \mathscr{C}_{\mathrm{op}}^N(x_1, \ldots, x_n, [\![S_1]\!], \ldots, [\![S_m]\!])$.

It does not impose additional restrictions on the introduced context. For many encodings the parameter N is not relevant or can be omitted by using instead a renaming policy on the source term names in the generated context. In contrast to homomorphy, compositionality is a very natural requirement. Intuitively, it states that every occurrence of an operator in the source term is treated by the encoding function in exactly the same way, i.e., is translated into the same term modulo the translation of the respective subterms. Also note that a compositional encoding, i.e., an encoding that translates all source term operators compositionally, implies that also any source term context can be represented as a context in the target language [45]. Moreover, compositionality guides the design of encodings, because it describes how an encoding has to look like.

Another structural criterion is the *preservation of substitutions*, denoted as *name invariance* in [22] and as *stability* in [13]. It usually requires that, for all source terms $S \in \mathscr{P}_{\mathrm{S}}$ and all substitutions σ on source terms, there exists some substitution σ' on target terms such that $[\![\sigma(S)]\!] \asymp_T \sigma'([\![S]\!])$ for some equivalence $\asymp_T \subseteq \mathscr{P}_{\mathrm{T}} \times \mathscr{P}_{\mathrm{T}}$ on target terms. Often additional requirements on the relationship between σ and σ' or on the equivalence $\asymp_T$ are stated. The strictest case is of course that $\sigma = \sigma'$ and $\asymp_T = \equiv_\alpha$. This criterion is based on the idea that names are property-less [13]. Hence, the preservation of substitutions should ensure that encodings of source terms that differ only in their free names can also only differ in free names (modulo the provided equivalence).

In [22] name invariance is defined modulo the introduced renaming policy. Accordingly, an encoding is considered independent of specific names if it preserves all substitutions σ on source terms by a substitution σ' on target terms such that σ' respects the changes made by the renaming policy.

Definition 3.4 (Name Invariance) *The encoding* $[\![\cdot]\!]$ *is* name invariant *if, for every* $S \in \mathscr{P}_{\mathrm{S}}$ *and* σ*, it holds that*

$$[\![\sigma(S)]\!] \begin{cases} \equiv_\alpha \sigma'([\![S]\!]) & \textit{if } \sigma \textit{ is injective} \\ \asymp_{\mathrm{T}} \sigma'([\![S]\!]) & \textit{otherwise} \end{cases}$$

where σ' *is such that* $\phi(\sigma(n)) = \sigma'(\phi(n))$ *for every* $n \in \mathscr{N}$*.*

Moreover, [59] present *link independence*, a condition that prevents encodings from introducing free names. More precisely, link independence means that, for all source terms $S_1, S_2 \in \mathscr{P}_{\mathrm{S}}$, $\mathsf{fn}(S_1) \cap \mathsf{fn}(S_2) = \emptyset$ implies $\mathsf{fn}([\![S_1]\!]) \cap \mathsf{fn}([\![S_2]\!]) = \emptyset$.

We denote the criteria presented in this subsection as *structural criteria* because they focus mainly on structural properties of the encoding function. We should, however, be aware that every criterion limits the existence of encoding functions and that such limitations usually also have a semantic effect. The main purpose of compositionality is to ensure that the encoding function is compositional. But compositionality also forbids for global coordination (see Section 3.6). Because of that, there is a number of well-accepted encodings—as e.g. the encoding of the π-calculus into the join-calculus in [11]—that are not compositional. Instead the encoding in [11] consists of two levels: an outer level that is parametrised on the free names of the source term and an inner compositional encoding.

3.6 Domain-Specific Criteria

Above we presented different kinds of criteria that were used to specify the notion of a "good" encoding in the literature. Thereby the purpose of all criteria introduced so far, is to define a general notion of correctness for encodings. We know that there is no agreement about the choice, combination, or concrete variant of the above criteria for a general notion of correctness or quality of an encoding. But even if we would agree on such a general notion of quality, there would still be the need for domain-specific criteria. A general notion is in particular necessary, if we want to compare different results or build a hierarchy. Accordingly, such a general notion of quality is a good starting point for language comparison. Nonetheless, we will sometimes need to extend it by domain-specific criteria. Domain-specific criteria, as the name suggests, are used to analyse properties of a specific domain that may in general not be interesting. Hence, it is not a good idea to overload a general framework by permanently adding domain-specific criteria. Instead, we add a domain-specific criterion only if it is necessary to answer a particular kind of question. Possible domains in the context of process calculi are e.g. causality, the branching time behaviour of processes, or considerations related to some special features as failures or time constrains.

Note that an additional criterion may strengthen an encodability result, but it weakens separation results. Moreover, already a single additional criterion significantly complicates the comparison of a result with other already established results that do not rely on this additional criterion. Quite often, they even lead to incomparable results; in particular if two results use different additional criteria. Hence, domain-specific criteria should only be used if they are unavoidable to answer a specific kind of question.

As an example for a domain-specific criterion, we discuss how to obtain a criterion that ensures that an encoding preserves the degree of distribution. Given an extension of a process calculus with an explicit notion of distribution or location we can define the degree of distribution of a system as the number of its locations. If locations are not defined explicitly, a process P is distributable into $P_1, \ldots, P_n$, if we find some distribution that extracts $P_1, \ldots, P_n$ from within P onto different locations. Preservation of distribution then means that the target term is as least as distributable as the source term. Note that the operator of process calculi that is usually associated with distribution is the parallel operator. Accordingly, we consider components of a term that are composed in parallel as distributable. More precisely, we understand distribution as the separation of a process into its (sequential) components.

Hence, by studying distribution preserving encodings, we analyse the possibilities to implement the operators of a calculus or especially its parallel operator. If it is always possible to preserve the degree of distribution in an encoding of a source language into a target language which is close to an implementation e.g. in a real world scenario, then the corresponding parallel operator can be implemented in this scenario simply as the operator of distribution, i.e., parallel source terms can be implemented in distributed real world processes. If it is not possible to obtain a distribution preserving encoding, then the source language implicitly defines side conditions on the use of the parallel operator usually induced by the defined synchronisation mechanism that forbids for such simple implementations. Thus, the implementation of parallel source terms as distributed processes may be possible only under some side conditions, which are hopefully already paraphrased by the respective separation result. In particular, an encoding that preserves the degree of distribution should not introduce a coordinator for concurrent actions, because if concurrent actions are coordinated by the same instance they are sequentialised, i.e., the implementation is less efficient.

Note that compositionality in Definition 3.3 already prevents from the use of global coordinators. Compositionality requires that all occurrences of a parallel operator have to be translated basically in the same way. Hence, if such an encoding introduces a coordinator then for every parallel operator a coordinator is introduced and there is no possibility to examine which of them is the outermost or to order them, i.e., it is not possible to coordinate the coordinators such that they proceed as a centralised entity. In that view, compositionality can be seen as a minimal criterion to ensure the preservation of distribution. However, compositionality alone is too weak, because it still allows for *local coordinators*, i.e., a compositional encoding may still sequentialise some parts of a source term (see e.g. the encodings in [52, 24]).

Instead the homomorphic translation of the parallel operator, i.e., $[\![P \mid Q]\!] = [\![P]\!] \mid [\![Q]\!]$, was often used as a criterion to measure whether an encoding respects the degree of distribution (see e.g. [43, 10, 30]). The homomorphic translation of the parallel operator forbids the introduction of (global and local) coordinators for the translation of the parallel operator. As discussed in [49, 52, 53] the homomorphic translation of the parallel operator usually implies that the respective encoding indeed preserves the degree of distribution but that the converse is not true, i.e., there are encodings that do not translate the parallel operator homomorphically but preserve nonetheless the degree of distribution of all source terms. In this sense, the homomorphic translation of the parallel operator is too strict—at least for separation results. It rightly forbids the introduction of coordinators that reduce the degree of distribution. But it also forbids protocols that handle communications of parallel components without sequentialising them or reducing the degree of distribution in another sense. Moreover, the homomorphic translation of the parallel operator is not always suited to reason about distribution in process calculi. For example in the join-calculus it is not always possible to separate distributed components by means of a parallel operator.

[49, 52, 53] introduce an alternative criterion for the preservation of the degree of distribution. Compositionality and the homomorphic translation of the parallel operator are both structural criteria. Hence, one may assume that also the preservation of distribution is a structural criterion, but in fact this is not true. A natural first condition is to require that encoded source terms are at least as distributable as the source term itself, i.e., that the degree of distribution has to be preserved by the encoding. However, it does not suffice to reason about the degree of distribution, i.e., about the number of distributable components, without additional requirements on the components. An encoding can always trivially ensure that the encoding has at least as much distributable components by introducing new components without any behaviour. Thus, we require that the encodings of distributable source term parts and their corresponding parts in the encoding are related by $\asymp_{\mathrm{T}}$. By doing so we relate the definition of the preservation of distributability to operational completeness, i.e., a semantic criterion that ensures the preservation of the behaviour of the source term (part). Hence, we require that each target term part can emulate at least all behaviour of the respective source part. As a side effect, we require, that whenever a part of a source term can solve a task independently of the other parts, i.e., it can reduce on its own, then the respective part of its encoding must also be able to emulate this reduction independently of the rest of the encoded term. This reflects our intuition that distribution adds some additional requirements on the independence of parallel terms. Accordingly, we require that not only the source term and its encoding are distributable to the same degree, but also their derivatives, i.e., we do not only consider the *initial* degree of distribution. Because of that, the criterion that is presented in [53] has both a structural as well as a semantic component. Remember that a term P is distributable into $P_1, \ldots, P_n$, if we find some distribution that extracts $P_1, \ldots, P_n$ from within P onto different locations.

Definition 3.5 (Preservation of Distribution) *An encoding* $[\![\cdot]\!] : \mathscr{P}_{\mathrm{S}} \to \mathscr{P}_{\mathrm{T}}$ preserves distribution *if for every* $S \in \mathscr{P}_{\mathrm{S}}$ *and for all terms* $S_1, \ldots, S_n \in \mathscr{P}_{\mathrm{S}}$ *that are distributable within S there are some* $T_1, \ldots, T_n \in \mathscr{P}_{\mathrm{T}}$ *that are distributable within* $[\![S]\!]$ *such that* $T_i \asymp_{\mathrm{T}} [\![S_i]\!]$ *for all* $1 \leq i \leq n$.

In essence, this requirement is a distributability-enhanced adaptation of operational completeness. Whenever a source term is distributable into n terms then its encoding must again be distributable into n terms, i.e., the encoded source term is at least as distributable as the source term itself. Moreover, if some of these n terms, say S_i, can perform some execution independently of the rest then, by operational completeness, this execution has to be emulated by its translation modulo $\asymp_{\mathrm{T}}$, i.e., $S_i \Longmapsto_{\mathrm{S}} S_i'$ implies $T_i \Longmapsto_{\mathrm{T}} \asymp_{\mathrm{T}} [\![S_i']\!]$. This formalisation of the preservation of distributability respects both the intuition on distribution as separation on different locations—captured by the structural requirement that the encoded source term is at least as distributable as the source term itself—as well as the intuition on distribution as independence of processes and their executions—a semantic requirement implemented by the condition $T_i \asymp_{\mathrm{T}} [\![S_i]\!]$.

The main disadvantage of this criterion is its complexity. It is more model-independent, since it does not rely on the notion of the parallel operator, i.e., is applicable also for calculi without a parallel operator or completely different kinds of parallel operators such as in CSP and it can be applied to calculi like the join-calculus, where distributed components are not always separated by a parallel operator. But, in order to apply this criterion, we have to specify in the source and the target language what it means for a process to be distributable into a set of components. In calculi with explicit locations this is usually easy. But it is difficult to find a general, i.e., model-independent, formalisation of distributability (see [49, 52, 53]). Moreover, this criterion was designed in order to be combined with operational correspondence and is strongly connected to this criterion. The advantage of the homomorphic translation of the parallel operator is that it is simple, easy to understand, and described independently of other criteria. But it is too strong for separation results, whereas compositionality is too weak for encodability results.

3.7 Summary

We introduced a number of encodability criteria. We observe that these criteria appear in different variants and that there is no agreement on a set of criteria that should be used; not even if we ignore domain-specific criteria. Moreover, we observe that some criteria are often used but as the discussion on full abstraction (see Section 3.2 and [13, 23, 46]) shows not always well-understood. For domain-specific criteria the situation is usually worse, i.e., it is even more difficult to analyse whether they are suitable. A mechanism to reason about the quality of encodability criteria is discussed in Section 5.

Encodability criteria define properties of the encoding function and should ensure that an encoding can be considered as meaningful. Unfortunately, encodability and separation results that are derived w.r.t. different sets of encodability criteria or different variants of these criteria are hard to compare. Section 4 analyses general frameworks of encodability criteria. Note that the formalisation of a criterion should be as general as possible, because a formalisation that is to close to a specific process calculus may hinder the derivation of similar results for other calculi and, thus, the comparison with other results. Furthermore, it is sometimes easier to define a new criterion with respect to an existing one, but again this may shrink the possibilities to compare to other results that do not satisfy the old criterion.

Finally, note that the criteria pose different kinds of proof obligations on the correctness proofs of encodings. Structural criteria (and also the criteria for the efficiency of an encoding function) are usually easier to verify than the remaining semantic criteria. But the semantic criteria are often considered as more essential. Unfortunately, we are not only lacking mechanisms to analyse the quality of encodability criteria and an agreement of a general notion of a "good" encoding but also general proof techniques for encodability criteria. From the criteria introduced above full abstraction and operational correspondence are usually the most elaborate criteria to prove. In the case of full abstraction the proofs heavily depend on the chosen pair of equivalences. Operational correspondence is usually shown by induction on the nature of source or target term steps. But some more sophisticated study of proof techniques might provide e.g. some hints on how to deal with the intermediate states that weakly operationally corresponding encodings might introduce (compare e.g. to the discussion of pre- and post-processing steps in [49]).

4 A General Notion of Quality

A general notion of quality is important to reason about the general expressive power of languages and to compare different results. In particular, they are essential to build a hierarchy that can only be based on encodability and separation results w.r.t. the same set of criteria. Moreover, if we want to relate the expressive power of two given languages there is no guidance on how to start or whether an obtained result is sufficiently substantiated by the chosen criteria to call it reasonable. There are basically two problems that we have to face in providing such a general notion: (1) We have to choose the nature of criteria. For a general setting the criteria should be model-independent. This also includes to some extent the relations that are used e.g. in full abstraction and operational correspondence, i.e., a general setting has to specify how to choose them. Moreover, the criteria should be designed to capture properties of the encoding that are generally agreed as being useful. (2) We have to decide on the variants of the respective criteria. Weaker criteria allow for more encodability and less separation results, whereas more restrictive criteria allow for less encodability and more separation results. The difficulty is to identify the sweet spot between too restrictive and too weak criteria such that the variants are meaningful for separation as well as encodability results. As discussed in Section 3.6, there is a need for domain-specific criteria. Accordingly, it should be possible to extend a general framework by domain-specific criteria.

4.1 Towards A Unified Approach to Encodability and Separation Results

In order to provide a general framework, [19, 22] suggests five criteria well suited for language comparison, i.e., for positive as well as negative translational results. As claimed in [22], most of the encodings appearing in the literature satisfy this framework and several known separation results can also be derived within this framework but there are also encodings that do not satisfies this framework, i.e., the framework is not trivial. The set of criteria is small and handy but at the same time guides the design of encoding functions and supports the proof of translational results by separating the requirements on different intuitive criteria. This framework specifies an encoding to be “good” if it satisfies the five presented criteria. In [19, 20, 21, 22] a number of encodability and separation results—including some new results—that are derived in this setting can be found.

The five conditions are divided into two structural and three semantic criteria. The structural criteria include (1) *compositionality* and (2) *name invariance*. The semantic criteria include (3) *operational correspondence*, (4) *divergence reflection*, and (5) *success sensitiveness*. Note that for the definition of name invariance and operational correspondence a behavioural equivalence $\asymp_{\mathrm{T}}$ for the target language is assumed. Its purpose is to describe the abstract behaviour of a target process, where abstract refers to the behaviour of the source term.

The two structural criteria were introduced and discussed in Section 3.5. Compositionality is given in Definition 3.3 and name invariance in Definition 3.4. They state that the encoding function should be compositional and independent of specific names. To ensure that there are no conflicts between names that are introduced by the encoding function for technical reasons, i.e., to implement some kind of protocol, and the source term names, the encoding is equipped with a renaming policy (Definition 2.3).

The first semantic criterion and usually the most elaborate one to prove is weak operational correspondence as given in Definition 3.1. The definition of operational correspondence relies on the equivalence $\asymp_{\mathrm{T}}$ to get rid of junks possibly left over within computations of target terms. To deal with infinite computations in encodings, the second semantic criterion requires that the encoding reflects divergence. It ensures that the encoding function cannot introduce new divergent behaviour, i.e., all divergent target terms are due to the encoding of a divergent source term. The last criterion links the behaviour of source terms to the behaviour of their encodings. [22] assumes a *success* operator $\checkmark$ as part of the syntax of both the source and the target language. An encoding preserves the abstract behaviour of the source term if it and its encoding answer the tests for success in exactly the same way (Definition 3.2). This criterion only links the behaviours of source terms and their literal translations, but not of their derivatives. To do so, [22] relates success sensitiveness and operational correspondence by requiring that the equivalence on the target language never relates two processes with different success behaviours.

Definition 4.1 (Success Respecting) *An equivalence $\mathscr{R}$ is* success respecting *if, for every $P, Q \in \mathscr{P}_C$ with $P \Downarrow_{\checkmark}$ and $Q \not\Downarrow_{\checkmark}$, it holds that $(P, Q) \notin \mathscr{R}$. We require that $\asymp_{\mathrm{T}}$ is a success respecting equivalence.*

The combination of success sensitiveness and operational correspondence allows to compare the behaviour of source and target terms even if they do not share common observables. By [22] a “good” equivalence $\asymp_{\mathrm{T}}$ is often defined in the form of a barbed equivalence (as described e.g. in [37]) or can be derived directly from the reduction semantics (as described e.g. in [28]) and is often a congruence, at least with respect to parallel composition.

4.2 Theory of Interaction

In contrast to [22], the general framework in [13] is based on labelled semantics. Intuitively, they combine full abstraction and operational correspondence into a bisimulation-like relation called *subbisimulation*. Subbisimulation in [13] connects labelled executions of the source and the target language. Moreover, preservation and reflection of divergence is required. The approach is then used to compare different variants of CCS and different variants of the π-calculus. For example subbisimilarity is applied to show the independence of the operators of the π-calculus.

The paper [12] extends this approach into a general theory of interaction. The prime motivation for the theory of interaction is to bridge the gap between the computation theory, i.e., what kind of functions can be computed, and the interaction theory. Therefore four fundamental and model-independent principles are presented and from them a general theory of equality and expressiveness are derived. Again subbisimilarity but without labels is used as criterion for encodings.

Definition 4.2 (Subbisimilarity) *A relation $\mathscr{R}$ is a* subbisimilarity *if it is total, sound, equipollent, extensional, codivergent, and bisimilar.*

Intuitively, (1) total means that for every source term there is a target term, (2) soundness is similar to operational soundness, (3) equipollence implies that related terms either both cannot reach any state with an observable or both can reach a state with (potentially different) observables, (4) extensional means congruent w.r.t. the parallel operator and restriction, (5) codivergent means that the relation is divergence respecting, and (6) bisimilarity is defined similar to Definition 2.1.

In comparison to the framework of Gorla the requirements induced by the formulation of subbisimilarity seem to be stricter, although a direct comparison is difficult, because of the different formulations. However, [13] and [12] make use of a stricter variant of operational correspondence that does not allow for intermediate or partially committed states. Moreover, [13, 12] fix a number of assumptions on process calculi, e.g. that all process calculi contain at least the parallel operator and restriction.

4.3 Musings on Encodings and Expressiveness

Also [17, 18] aims at providing a general notion of expressiveness for language comparison. Here an encoding should be *valid* and *correct*. In [18] these concepts are defined up to a semantic equivalence or preorder $\sim$, that is not fixed by the framework.

Intuitively, an encoding is valid if it preserves the meaning of expressions, i.e., such that the meaning of a translated expression is semantically equivalent to the meaning of the original. Therefore, [18] assumes a function for each language that associates the *meaning* to processes by mapping them on values (for simplicity and in contrast to [18] I ignore in the following that processes may contain variables). Different languages may have different domains of values. The semantic relation $\sim$ is used to mediate between these different domains, i.e., it is assumed that it is a relation on values over a universe that contains the domains of the source as well as the target language. Moreover, [18] requires a *semantic translation* $\mathbf{R}$ that is a relation that relates each value of the source with its counterpart (or counterparts) in the target. Then, an encoding is *correct* iff the meaning of the translation of an expression is a counterpart of the meaning of this expression.

Definition 4.3 (Correctness) *An encoding $[\![\cdot]\!]$ is* correct *w.r.t. a semantic translation* $\mathbf{R}$ *if* $\mathbf{R}$ *relates the meaning of S and $[\![S]\!]$ for all $S \in \mathscr{P}_{\mathrm{S}}$.*

An encoding $[\![\cdot]\!]$ is correct *up to $\sim$ iff $\sim$ is an equivalence, the restriction* $\mathbf{R}$ *of $\sim$ to the cross product of the domain of the source and the target is a semantic translation, and $[\![\cdot]\!]$ is correct w.r.t.* $\mathbf{R}$.

Definition 4.4 (Validity) *An encoding $[\![\cdot]\!]$ is* valid *up to $\sim$ iff it is correct w.r.t. some semantic translation* $\mathbf{R} \subseteq \sim$. *Language $\mathscr{L}_{\mathrm{T}}$ is at least as expressive as $\mathscr{L}_{\mathrm{S}}$ up to $\sim$ if an encoding valid up to $\sim$ from $\mathscr{L}_{\mathrm{S}}$ into $\mathscr{L}_{\mathrm{T}}$ exists.*

As discussed in [18], in comparison to [22] the above approach implies a slightly stricter variant of compositionality (without the parameter on a set of names N) but no name invariance criterion. Moreover, the combination of three semantic criteria of [22] are close to an instantiation of the criteria in this approach with a particular preorder $\sim$, namely a success respecting and divergence respecting variant of bisimulation (see Section 5).

5 Formalising and Analysing Encodability Criteria

There exists a bunch of different criteria and different variants of criteria in order to reason in different settings. This leads to incomparable results. Moreover it is not always clear whether the criteria used to obtain a result in a particular setting do indeed fit to this setting. A way to formally reason about and compare encodability criteria is presented in [50]. The main idea of this approach is to map encodability criteria on requirements on a relation between source and target terms that is induced by the encoding function. This way the problem of analysing or comparing encodability criteria is reduced to the better understood problem of comparing relations on processes. An Isabelle/HOL formalisation can be found in [51].

The different purposes of encodability criteria lead to very different kinds of conditions that are usually hard to analyse and compare directly. In fact even widely used criteria—as full abstraction—seem not to be fully understood by the community, as the need for articles as [23, 46] shows. In contrast to that, relations on processes—such as simulations and bisimulations—are a very well studied and understood topic (see for example [16]). Moreover, it is natural to describe the behaviour of terms, or compare them, modulo some equivalence relation. Also many encodability criteria, like operational correspondence, are obviously designed with a particular kind of relation between processes in mind. Therefore, in order to be able to formally reason about encodability criteria, to completely capture and describe their semantic effect, and to analyse side conditions of combinations of criteria, mapping them on conditions on relations between source and target terms seems to be natural.

According to [50] every encoding $[\![\cdot]\!] : \mathscr{P}_{\mathrm{S}} \to \mathscr{P}_{\mathrm{T}}$ induces a relation $\mathscr{R}_{[\![\cdot]\!]} \subseteq (\mathscr{P}_{\mathrm{S}} \uplus \mathscr{P}_{\mathrm{T}})^2$ on the disjoint union of its source and target terms by relating source terms and their literal translations, i.e., $(S, [\![S]\!]) \in \mathscr{R}_{[\![\cdot]\!]}$ for all $S \in \mathscr{P}_{\mathrm{S}}$. Encodability criteria induce further properties on such relations. By analysing the different kinds of such properties the semantic effect of an encodability criterion can be studied and different criteria can be compared in a model-independent and formal way. In order to completely capture the effect of a criterion, [50] aims at iff-results of the form

$$[\![\cdot]\!] \text{ satisfies } \mathbf{C} \text{ iff there exists a relation } \mathscr{R}_{[\![\cdot]\!]} \text{ such that } \forall S.\ (S, [\![S]\!]) \in \mathscr{R}_{[\![\cdot]\!]} \text{ and } \mathsf{P}\big(\mathscr{R}_{[\![\cdot]\!]}\big),$$

where P is the condition that captures the effect of $\mathbf{C}$. For example, an encoding reflects divergence iff there exists a relation $\mathscr{R}_{[\![\cdot]\!]}$ such that $\forall S.\ (S, [\![S]\!]) \in \mathscr{R}_{[\![\cdot]\!]}$ and $\mathscr{R}_{[\![\cdot]\!]}$ reflects divergence. Similarly, an encoding (weakly/strongly) respects barbs iff there exists a relation $\mathscr{R}_{[\![\cdot]\!]}$ such that $\forall S.\ (S, [\![S]\!]) \in \mathscr{R}_{[\![\cdot]\!]}$ and $\mathscr{R}_{[\![\cdot]\!]}$ (weakly/strongly) respects barbs.

Using this technique, [50] shows that without further requirements on the source and target relations $\mathscr{R}_{\mathrm{S}}$ and $\mathscr{R}_{\mathrm{T}}$, that are used in the formulation of full abstraction, the semantic effect of full abstraction is very small. Full abstraction is mapped on a relation that relates at least each source term to its literal translation and includes the relations $\mathscr{R}_{\mathrm{S}}$ and $\mathscr{R}_{\mathrm{T}}$. Let us additionally add pairs of the form $([\![S]\!], S)$ for all $S \in \mathscr{P}_{\mathrm{S}}$. Then, an encoding is fully abstract w.r.t. the preorders $\mathscr{R}_{\mathrm{S}}$ and $\mathscr{R}_{\mathrm{T}}$ iff there exists a transitive relation $\mathscr{R}_{[\![\cdot]\!]}$ that relates at least each source term to its literal translation in both directions, such that the restriction of $\mathscr{R}_{[\![\cdot]\!]}$ to source terms $\mathscr{R}_{[\![\cdot]\!]}\restriction_{\mathscr{P}_{\mathrm{S}}}$ is $\mathscr{R}_{\mathrm{S}}$ and $\mathscr{R}_{[\![\cdot]\!]}\restriction_{\mathscr{P}_{\mathrm{T}}} = \mathscr{R}_{\mathrm{T}}$.

Lemma 5.1 (Full Abstraction, [50]) $[\![\cdot]\!] : \mathscr{P}_{\mathrm{S}} \to \mathscr{P}_{\mathrm{T}}$ *is* fully abstract *w.r.t. the preorders* $\mathscr{R}_{\mathrm{S}} \subseteq \mathscr{P}_{\mathrm{S}}^2$ *and* $\mathscr{R}_{\mathrm{T}} \subseteq \mathscr{P}_{\mathrm{T}}^2$ *iff* $\exists \mathscr{R}_{[\![\cdot]\!]}.\ \big(\forall S.\ (S, [\![S]\!]), ([\![S]\!], S) \in \mathscr{R}_{[\![\cdot]\!]}\big) \land \mathscr{R}_{\mathrm{S}} = \mathscr{R}_{[\![\cdot]\!]}\restriction_{\mathscr{P}_{\mathrm{S}}} \land \mathscr{R}_{\mathrm{T}} = \mathscr{R}_{[\![\cdot]\!]}\restriction_{\mathscr{P}_{\mathrm{T}}} \land \mathscr{R}_{[\![\cdot]\!]}$ *is transitive.*

Thus an encoding is fully abstract w.r.t. $\mathscr{R}_{\mathrm{S}}$ and $\mathscr{R}_{\mathrm{T}}$ if the encoding function combines the relations $\mathscr{R}_{\mathrm{S}}$ and $\mathscr{R}_{\mathrm{T}}$ in a transitive way. This underpins the discussions in [13, 23, 46] showing that full abstraction without a clarification of the respective equivalences is not meaningful. It also shows the need for a formal analysis of encodability criteria.

The formulation of operational correspondence (in all its variants) strongly reminds us of simulation relations on processes, such as bisimilarity. Obviously this criterion is designed in order to establish a simulation-like relation between source and target terms. By mapping this criterion on properties of a relation, we can determine the exact nature of this relation. The first two variants of Definition 3.1 exactly describe strong and weak bisimilarity up to $\asymp_{\mathrm{T}}$.

Lemma 5.2 (Operational Correspondence, [50]) *An encoding* $[\![\cdot]\!] : \mathscr{P}_{\mathrm{S}} \to \mathscr{P}_{\mathrm{T}}$ *is operational corresponding w.r.t. a preorder* $\asymp_{\mathrm{T}} \subseteq \mathscr{P}_{\mathrm{T}}^2$ *that is a bisimulation iff* $\exists \mathscr{R}_{[\![\cdot]\!]}.\ \left(\forall S.\ (S, [\![S]\!]) \in \mathscr{R}_{[\![\cdot]\!]}\right) \wedge \asymp_{\mathrm{T}} = \mathscr{R}_{[\![\cdot]\!]} {\restriction_{\mathscr{P}_{\mathrm{T}}}} \wedge \left(\forall S, T.\ (S,T) \in \mathscr{R}_{[\![\cdot]\!]}\ \textit{implies}\ ([\![S]\!], T) \in \asymp_{\mathrm{T}}\right) \wedge \mathscr{R}_{[\![\cdot]\!]}$ *is a preorder and a bisimulation.*

Here, the conditions $\mathscr{R}_{\mathrm{T}} = \mathscr{R}_{[\![\cdot]\!]} {\restriction_{\mathscr{P}_{\mathrm{T}}}}$ and $\left(\forall S, T.\ (S,T) \in \mathscr{R}_{[\![\cdot]\!]} \to ([\![S]\!], T) \in \mathscr{R}_{\mathrm{T}}\right)$ are technical side conditions for the proof of the only-if-part that ensure that $\asymp_{\mathrm{T}}$ is a bisimulation. We obtain the same result if we replace operational correspondence by strong operational correspondence and bisimulation by strong bisimulation. Accordingly, operational correspondence ensures that source terms and their translations are reduction bisimilar.

To obtain a similar result for weak operational correspondence, [50] had to introduce a new kind of simulation relation denoted as correspondence simulation.

Lemma 5.3 (Weak Operational Correspondence, [50]) $[\![\cdot]\!]$ *is weakly operational corresponding w.r.t. a preorder* $\asymp_{\mathrm{T}} \subseteq \mathscr{P}_{\mathrm{T}}^2$ *that is a correspondence simulation iff* $\exists \mathscr{R}_{[\![\cdot]\!]}.\ \left(\forall S.\ (S, [\![S]\!]) \in \mathscr{R}_{[\![\cdot]\!]}\right) \wedge \asymp_{\mathrm{T}} = \mathscr{R}_{[\![\cdot]\!]} {\restriction_{\mathscr{P}_{\mathrm{T}}}} \wedge \left(\forall S, T.\ (S,T) \in \mathscr{R}_{[\![\cdot]\!]}\ \textit{implies}\ ([\![S]\!], T) \in \asymp_{\mathrm{T}}\right) \wedge \mathscr{R}_{[\![\cdot]\!]}$ *is a preorder and a correspondence simulation.*

We omit the definition of correspondence simulation but point out that it is a simulation relation that is in between coupled similarity (Definition 2.2) and bisimulation. Accordingly, weak operational correspondence ensures that source terms and their literal translations are coupled similar.

As stated in [50], the combination of the above results implies that the three semantic criteria of [22] ensure that any "good" encoding in this framework relates source and target terms by a coupled simulation that reflects divergence and respects success. The approach presented in [13, 12] was not addressed in [50] but it requires itself a simulation relation between source and target terms. Subbisimilarity is a variant of bisimulation. Accordingly, the requirements of [13, 12] are more restrictive than [22]. The analysis of structural criteria is left for further research, because structural criteria need more assumptions on the considered languages. As discussed in [50], the formal analysis of encodability criteria can also help to derive proof methods for the respective criteria.

6 Conclusions

As stated in the end of Section 3, there are basically three lines of further research: (1) We need techniques to reason about the quality of encodability criteria to ensure that our encodings are indeed meaningful. Section 5 revises one way to do that, but raises itself some open questions as e.g. how to deal with structural criteria. (2) In order to compare results and build hierarchies we need a general notion of the quality of an encoding. Section 4 outlines three different frameworks for this purpose. All three approaches have certain basic ideas in common, e.g. all three variants imply structural and semantic criteria. But the existence of three frameworks shows that there is still no general agreement and indeed these three frameworks differ not only w.r.t. the semantic equivalence they induce (see Section 5) they also use quite different terminologies and basic definitions. With that they impose different proof techniques. (3) Besides the analysis of encodability criteria, we are also lacking a study of proof techniques for particular criteria.

References

[1] S. Arun-Kummar & M. Hennessy (1992): *An efficiency preorder for processes*. Acta Informatica 29(8), pp. 737–760, doi:10.1007/BF01191894.

[2] J.C.M. Baeten (2005): *A brief history of process algebra*. Theoretical Computer Science 335(2–3), pp. 131–146, doi:10.1016/j.tcs.2004.07.036.

[3] M. Baldamus, J. Parrow & B. Victor (2005): *A Fully Abstract Encoding of the π-Calculus with Data Terms (Extended Abstract)*. In: Proc. of ICALP, LNCS 3580, Springer, pp. 1202–1213, doi:10.1007/11523468_97.

[4] J.A. Bergstra & J.W. Klop (1982): *Fixed point semantics in process algebra*. Technical Report IW 206/82, Mathematical Centre, Amsterdam.

[5] F.S. Boer & C. Palamidessi (1991): *Embedding as a tool for Language Comparison: On the CSP hierarchy*. In: Proc. of CONCUR, LNCS 527, Springer, pp. 127–141, doi:10.1007/3-540-54430-5_85.

[6] G. Boudol (1992): *Asynchrony and the π-calculus (note)*. Note, INRIA.

[7] G.N. Buckley & A. Silberschatz (1983): *An Effective Implementation for the Generalized Input-Output Construct of CSP*. ACM Transactions on Programming Languages and Systems (TOPLAS) 5(2), pp. 223–235, doi:10.1145/69624.357208.

[8] N. Busi, M. Gabbrielli & G. Zavattaro (2009): *On the expressive power of recursion, replication and iteration in process calculi*. Mathematical Structures in Computer Science 19(6), pp. 1191–1222, doi:10.1017/S096012950999017X.

[9] D. Cacciagrano, F. Corradini, J. Aranda & F.D. Valencia (2008): *Linearity, Persistence and Testing Semantics in the Asynchronous Pi-Calculus*. Electronic Notes in Theoretical Computer Science 194(2), pp. 59–84, doi:10.1016/j.entcs.2007.11.006.

[10] M. Carbone & S. Maffeis (2003): *On the Expressive Power of Polyadic Synchronisation in π-Calculus*. Nordic Journal of Computing 10(2), pp. 70–98, doi:10.1016/S1571-0661(05)80361-5.

[11] C. Fournet & G. Gonthier (1996): *The Reflexive CHAM and the Join-Calculus*. In: Proc. of POPL, SIGPLAN-SIGACT, ACM, pp. 372–385, doi:10.1145/237721.237805.

[12] Y. Fu (2016): *Theory of Interaction*. Theoretical Computer Science 611, pp. 1–49, doi:10.1016/j.tcs.2015.07.043.

[13] Y. Fu & H. Lu (2010): *On the expressiveness of interaction*. Theoretical Computer Science 411(11-13), pp. 1387–1451, doi:10.1016/j.tcs.2009.11.011.

[14] R.J. van Glabbeek (1993): *The Linear Time – Branching Time Spectrum II*. In: Proc. of CONCUR, LNCS 715, pp. 66–81, doi:10.1007/3-540-57208-2_6.

[15] R.J. van Glabbeek (1994): *On the expressiveness of ACP (extended abstract)*. In: Proc. of ACP, Workshops in Computing, pp. 188–217, doi:10.1007/978-1-4471-2120-6_8.

[16] R.J. van Glabbeek (2001): *The Linear Time – Branching Time Spectrum I: The Semantics of Conrete, Sequential Processes*. Handbook of Process Algebra, pp. 3–99, doi:10.1016/B978-044482830-9/50019-9.

[17] R.J. van Glabbeek (2012): *Musings on Encodings and Expressiveness*. In: Proc. of EXPRESS/SOS, EPTCS 89, pp. 81–98, doi:10.4204/EPTCS.89.7.

[18] R.J. van Glabbeek (2018): *A Theory of Encodings and Expressiveness (Extended Abstract)*. In: Proc. of FoSSaCS, LNCS 10803, pp. 183–202, doi:10.1007/978-3-319-89366-2_10.

[19] D. Gorla (2008): *Towards a Unified Approach to Encodability and Separation Results for Process Calculi*. In: Proc. of CONCUR, LNCS 5201, pp. 492–507, doi:10.1007/978-3-540-85361-9_38.

[20] D. Gorla (2009): *On the Relative Expressive Power of Calculi for Mobility*. In: Proc. of MFPS, ENTCS 249, pp. 269–286, doi:10.1016/j.entcs.2009.07.094.

[21] D. Gorla (2010): *A taxonomy of process calculi for distribution and mobility*. Distributed Computing 23(4), pp. 273–299, doi:10.1007/s00446-010-0120-6.

[22] D. Gorla (2010): *Towards a Unified Approach to Encodability and Separation Results for Process Calculi*. Information and Computation 208(9), pp. 1031–1053, doi:10.1016/j.ic.2010.05.002.

[23] D. Gorla & U. Nestmann (2014): *Full abstraction for expressiveness: history, myths and facts*. Mathematical Structures in Computer Science, pp. 1–16, doi:10.1017/S0960129514000279.

[24] M. Hatzel, C. Wagner, K. Peters & U. Nestmann (2015): *Encoding CSP into CCS*. In: Proc. of EXPRESS/SOS, EPTCS 7, pp. 61–75, doi:10.4204/EPTCS.190.5.

[25] C. Hewitt, P. Bishop & R. Steiger (1973): *A universal modular ACTOR formalism for artificial intelligence*. In: Proc. of IJCAI, ACM, pp. 235–245.

[26] C.A.R. Hoare (1978): *Communicating Sequential Processes*. Communications of the ACM 21(8), pp. 666–677, doi:10.1145/359576.359585.

[27] K. Honda & M. Tokoro (1991): *An Object Calculus for Asynchronous Communication*. In: Proc. of ECOOP, LNCS 512, pp. 133–147, doi:10.1007/BFb0057019.

[28] K. Honda & N. Yoshida (1995): *On Reduction-Based Process Semantics*. Theoretical Computer Science 151(2), pp. 437–486, doi:10.1016/0304-3975(95)00074-7.

[29] F. Knabe (1993): *A Distributed Protocol for Channel-Based Communication with Choice*. Computers and Artificial Intelligence 12(5), pp. 475–490.

[30] C. Laneve & A. Vitale (2010): *The Expressive Power of Synchronizations*. In: Proc. of LICS, IEEE, pp. 382–391, doi:10.1109/LICS.2010.15.

[31] R.J. Lipton, L. Snyder & Y. Zalcstein (1974): *A Comparative Study of Models of Parallel Computation*. In: Proc. of SWAT, IEEE, pp. 145–155, doi:10.1109/SWAT.1974.2.

[32] R. Milner (1989): *Communication and Concurrency*. Prentice-Hall, Inc.

[33] R. Milner (1992): *Functions as Processes*. Mathematical Structures in Computer Science 2(2), pp. 119–141, doi:10.1017/S0960129500001407.

[34] R. Milner (1993): *The Polyadic π-Calculus: a Tutorial*. Logic and Algebra of Specification 94, pp. 203–246, doi:10.1007/978-3-642-58041-3_6.

[35] R. Milner (1999): *Communicating and Mobile Systems: The π-Calculus*. Cambridge University Press, New York.

[36] R. Milner, J. Parrow & D. Walker (1992): *A Calculus of Mobile Processes, Part I and II*. Information and Computation 100(1), pp. 1–77, doi:10.1016/0890-5401(92)90008-4.

[37] R. Milner & D. Sangiorgi (1992): *Barbed Bisimulation*. In: Proc. of ICALP, LNCS 623, pp. 685–695, doi:10.1007/3-540-55719-9_114.

[38] J.C. Mitchell (1993): *On abstraction and the expressive power of programming languages*. Science of Computer Programming 21(2), pp. 141–163, doi:10.1016/0167-6423(93)90004-9.

[39] U. Nestmann (1996): *On Determinacy and Nondeterminacy in Concurrent Programming*. Ph.D. thesis, Universität Erlangen-Nürnberg.

[40] U. Nestmann (2000): *What is a "Good" Encoding of Guarded Choice?* Information and Computation 156(1-2), pp. 287–319, doi:10.1006/inco.1999.2822.

[41] U. Nestmann (2006): *Welcome to the Jungle: A subjective Guide to Mobile Process Calculi*. In: Proc. of CONCUR, LNCS 4137, pp. 52–63, doi:10.1007/11817949_4.

[42] U. Nestmann & B.C. Pierce (2000): *Decoding Choice Encodings*. Information and Computation 163(1), pp. 1–59, doi:10.1006/inco.2000.2868.

[43] C. Palamidessi (2003): *Comparing the Expressive Power of the Synchronous and the Asynchronous π-calculi*. Mathematical Structures in Computer Science 13(5), pp. 685–719, doi:10.1017/S0960129503004043.

[44] C. Palamidessi, V.A. Saraswat, F.D. Valencia & B. Victor (2006): *On the Expressiveness of Linearity vs Persistence in the Asychronous Pi-Calculus*. In: Proc. of LICS, IEEE Computer Society, pp. 59–68, doi:10.1109/LICS.2006.39.

[45] J. Parrow (2008): *Expressiveness of Process Algebras*. Electronic Notes in Theoretical Computer Science 209, pp. 173–186, doi:10.1016/j.entcs.2008.04.011.

[46] J. Parrow (2014): *General conditions for full abstraction*. Mathematical Structures in Computer Science 26(4), pp. 655–657, doi:10.1017/S0960129514000280.

[47] J. Parrow & P. Sjödin (1992): *Multiway Synchronization Verified with Coupled Simulation*. In: Proc. of CONCUR, LNCS 630, pp. 518–533, doi:10.1007/BFb0084813.

[48] J.A. Perez (2009): *Higher-Order Concurrency: Expressiveness and Decidability Results*. Ph.d. thesis, University of Bologna.

[49] K. Peters (2012): *Translational Expressiveness*. Ph.D. thesis, TU Berlin. Available at `http://opus.kobv.de/tuberlin/volltexte/2012/3749/`.

[50] K. Peters & R. van Glabbeek (2015): *Analysing and Comparing Encodability Criteria*. In: Proc. of EXPRESS/SOS, EPTCS 190, pp. 46–60, doi:10.4204/EPTCS.190.4.

[51] K. Peters & R. van Glabbeek (2015): *Analysing and Comparing Encodability Criteria for Process Calculi*. Archive of Formal Proofs. `http://isa-afp.org/entries/Encodability_Process_Calculi.shtml`.

[52] K. Peters & U. Nestmann (2012): *Is It a "Good" Encoding of Mixed Choice?* In: Proc. of FoSSaCS, LNCS 7213, pp. 210–224, doi:10.1007/978-3-642-28729-9_14.

[53] K. Peters, U. Nestmann & U. Goltz (2013): *On Distributability in Process Calculi*. In: Proc. of ESOP, LNCS 7792, pp. 310–329, doi:10.1007/978-3-642-37036-6_18.

[54] C.A. Petri (1962): *Kommunikation mit Automaten*. Ph.D. thesis, Institut für Instrumentelle Mathematik, Bonn.

[55] J.G. Riecke (1991): *Fully abstract translations between functional languages*. In: Proc. of POPL, ACM, pp. 245–254, doi:10.1145/99583.99617.

[56] D. Sangiorgi (1994): *An investigation into functions as processes*. In: Proc. of MFPS, LNCS 802, pp. 143–159, doi:10.1007/3-540-58027-1_7.

[57] D. Sangiorgi (2009): *On the Origins of Bisimulation and Coinduction*. ACM Transactions on Programming Languages and Systems (TOPLAS) 31(4), pp. 1–15, doi:10.1145/1516507.1516510.

[58] B. Victor & J. Parrow (1996): *Constraints as Processes*. In: Proc. of CONCUR, LNCS 1119, pp. 389–405, doi:10.1007/3-540-61604-7_66.

[59] M.G. Vigliotti, I. Phillips & C. Palamidessi (2007): *Tutorial on separation results in process calculi via leader election problems*. Theoretical Computer Science 388(1–3), pp. 267–289, doi:10.1016/j.tcs.2007.09.001.

[60] N. Yoshida (1996): *Graph Types for Monadic Mobile Processes*. In: Proc. of FST&TCS, LNCS 1180, pp. 371–386, doi:10.1007/3-540-62034-6_64.

Semantic Structures for Spatially-Distributed Multi-Agent Systems

Frank Valencia[*]
CNRS-LIX Ecole Polytechnique de Paris
Univ. Javeriana Cali
frank.valencia@lix.polytechnique.fr

Spatial constraint systems (scs) are semantic structures for reasoning about spatial and epistemic information in concurrent systems. They have been used to reason about beliefs, lies, and group epistemic behaviour inspired by social networks. They have also been used for proving new results about modal logics and giving semantics to process calculi. In this paper we will discuss the theory and main results about scs.

1 Introduction

Epistemic, mobile and spatial behavior are common place in today's distributed systems. The intrinsic *epistemic* nature of these systems arises from social behavior. Most people are familiar with digital systems where *agents* (users) share their *beliefs*, *opinions* and even intentional *lies* (hoaxes). Also, systems modeling decision behavior must account for those decisions' dependance on the results of interactions with others within some social context. The courses of action stemming from some agent decision result not only from the rational analysis of a particular situation but also from the agent beliefs or information that sprang from the interactions with other participants involved in that situation. Appropriate performance within these social contexts requires the agent to form beliefs about the beliefs of others. Spatial and mobile behavior is exhibited by apps and data moving across (possibly nested) spaces defined by, for example, friend circles and shared folders. We therefore believe that a solid understanding of the notion of *space* and *spatial mobility* as well as the flow of epistemic information is relevant in any model of today's distributed systems.

The notion of group is also fundamental in distributed systems. Since the early days of multi-user operating systems, information was categorized into that available to one user, some group of users, or everyone. Information was thus separated into "spaces" with boundaries defined by accessibility. In these systems we could say that, from the restrictive point of view of information "permissions", the notion of group was *reified* as another agent of the system.

In current distributed systems such as social networks, actors behave more as members of a certain *group* than as isolated individuals. Information, opinions, and beliefs of a particular actor are frequently the result of an evolving process of interchanges with other actors in a group. This suggests a reified notion of group as a single actor operating within the context of the collective information of its members. It also conveys two notions of information, one spatial and the other epistemic. In the former, information is localized in compartments associated with a user or group. In the latter, it refers to something known or believed by a single agent or collectively by a group.

Furthermore, in many real life multi-agent systems, the agents are unknown in advance. New agents can subscribe to the system in unpredictable ways. Thus, there is usually no a-priori bound on the number

[*]Frank Valencia has been partially supported by the ECOS-NORD project FACTS (C19M03).

of agents in the system. It is then often convenient to model the group of agents as an infinite set. In fact, in models from economics and epistemic logic [17, 16], groups of agents have been represented as infinite, even uncountable, sets. This raises interesting issues about the distributed information of such groups. In particular, that of *group compactness*: information that when obtained by an infinite group can also be obtained by one of its finite subgroups.

Spatial constraint systems (scs)[1] are semantic structures for the epistemic behaviour of multi-agent systems. These structures single out the notions we previously discussed: Namely, space, beliefs, and distributed information of potentially infinite groups. In this paper we will describe the theory of scs and highlight its main results from [8, 10, 11, 12, 9].

2 Overview

In this section we will give a brief description and motivate scs in the context of space, extrusion, and distributed information.

Declarative formalisms of concurrency theory such as process calculi for *concurrent constraint programming* (ccp) [24] were designed to give explicit access to the concept of partial information and, as such, have close ties with logic. This makes them ideal for the incorporation of epistemic and spatial concepts by expanding the logical connections to include *multi-agent modal logic* [19]. In fact, the sccp calculus [18] extends ccp with the ability to define local computational spaces where agents can store epistemic information and run processes.

Constraint systems (cs) are algebraic structures for the semantics of ccp [24, 2, 18, 5, 22, 20]. They specify the domain and elementary operations and partial information upon which programs (processes) of these calculi may act.

A cs can be formalized as a complete lattice $(Con, \sqsubseteq)$. The elements of Con represent partial information and we shall think of them as being *assertions*. They are traditionally referred to as *constraints* since they naturally express partial information (e.g., $x > 42$). The order $\sqsubseteq$ corresponds to entailment between constraints, $c \sqsubseteq d$, often written $d \sqsupseteq c$, means c can be derived from d, or that d represents as much information as c. The join $\sqcup$, the bottom *true*, and the top *false* of the lattice correspond to conjunction, the empty information, and the join of all (possibly inconsistent) information, respectively.

Constraint systems provide the domains and operations upon which the semantic foundations of ccp calculi are built. As such, ccp operations and their logical counterparts typically have a corresponding elementary construct or operation on the elements of the constraint system. In particular, parallel composition and conjunction correspond to the *join* operation, and existential quantification and local variables correspond to a cylindrification operation on the set of constraints [24].

Space. Similarly, the notion of computational space and the epistemic notion of belief in sccp [18] correspond to a family of join-preserving maps $\mathfrak{s}_i : Con \to Con$ called *space functions*. A cs equipped with space functions is called a *spatial constraint system* (scs). From a *computational point of view* $\mathfrak{s}_i(c)$ can be interpreted as an assertion specifying that c resides within the space of agent i. From an *epistemic point of view*, $\mathfrak{s}_i(c)$ specifies that i considers c to be true. An alternative epistemic view is that i interprets c as $\mathfrak{s}_i(c)$. All these interpretations convey the idea of c being local or subjective to agent i.

In the spatial ccp process calculus *sccp* [18], scs are used to specify the spatial distribution of information in configurations $\langle P, c\rangle$ where P is a process and c is a constraint, called *the store*, representing the current partial information. E.g., a reduction $\langle\ P, \mathfrak{s}_1(a) \sqcup \mathfrak{s}_2(b)\ \rangle \longrightarrow \langle\ Q, \mathfrak{s}_1(a) \sqcup \mathfrak{s}_2(b \sqcup c)\ \rangle$ means

[1]For simplicity we use *scs* for both *spatial constraint system* and its plural form.

that P, with a in the space of agent 1 and b in the space of agent 2, can evolve to Q while adding c to the space of agent 2.

Extrusion. An extrusion function for the space $\mathfrak{s}_i$ is a map $\mathfrak{e}_i : Con \to Con$ that satisfies $\mathfrak{s}_i(\mathfrak{e}_i(c)) = c$. This means that we think of extrusion as the *right inverse* of space. Intuitively, within a space context $\mathfrak{s}_i(\cdot)$, the assertion $\mathfrak{e}_i(c)$ specifies that c must be posted outside of agent i's space. The computational interpretation of $\mathfrak{e}_i$ is that of a process being able to extrude any c from the space $\mathfrak{s}_i$. The extruded information c may not necessarily be part of the information residing in the space of agent i. For example, using properties of space and extrusion functions we shall see that $\mathfrak{s}_i(\ d \sqcup \mathfrak{e}_i(c)) = \mathfrak{s}_i(d) \sqcup c$ specifying that c is extruded (while d is still in the space of i). The extruded c could be inconsistent with d (i.e., $c \sqcup d = false$), it could be related to d (e.g., $c \sqsubseteq d$), or simply unrelated to d. From an epistemic perspective, we can use $\mathfrak{e}_i$ to express *utterances* by agent i and such utterances could be intentional lies (i.e., inconsistent with their beliefs), informed opinions (i.e., derived from the beliefs), or simply arbitrary statements (i.e., unrelated to their beliefs).

Distributed Information. Let us consider again the sccp reduction $\langle\ P, \mathfrak{s}_1(a) \sqcup \mathfrak{s}_2(b)\ \rangle \longrightarrow \langle\ Q, \mathfrak{s}_1(a) \sqcup \mathfrak{s}_2(b \sqcup c)\ \rangle$. Assume that d is some piece of information resulting from the combination (join) of the three constraints above, i.e., $d = a \sqcup b \sqcup c$, but strictly above the join of any two of them. We are then in the situation where neither agent has d in their spaces, but as a group they could potentially have d by combining their information. Intuitively, d is distributed in the spaces of the group $I = \{1,2\}$. Being able to predict the information that agents 1 and 2 may derive as group is a relevant issue in multi-agent concurrent systems, particularly if d represents unwanted or conflicting information (e.g., $d = false$).

In [9] we introduced the theory of group space functions $\Delta_I : Con \to Con$ to reason about information distributed among the members of a potentially infinite group I. We refer to Δ_I as the *distributed space* of group I. In our theory $c \sqsupseteq \Delta_I(e)$ holds exactly when we can derive from c that e is distributed among the agents in I. E.g., for d above, we should have $\mathfrak{s}_1(a) \sqcup \mathfrak{s}_2(b \sqcup c) \sqsupseteq \Delta_{\{1,2\}}(d)$ meaning that from the information $\mathfrak{s}_1(a) \sqcup \mathfrak{s}_2(b \sqcup c)$ we can derive that d is distributed among the group $I = \{1,2\}$. Furthermore, $\Delta_I(e) \sqsupseteq \Delta_J(e)$ holds whenever $I \subseteq J$ since if e is distributed among a group I, it should also be distributed in a group that includes the agents of I.

Distributed information of infinite groups can be used to reason about multi-agent computations with unboundedly many agents. For example, a *computation* in sccp is a possibly infinite reduction sequence γ of the form $\langle\ P_0, c_0\ \rangle \longrightarrow \langle\ P_1, c_1\ \rangle \longrightarrow \cdots$ with $c_0 \sqsubseteq c_1 \sqsubseteq \cdots$. The *result* of γ is $\bigsqcup_{n \geq 0} c_n$, the join of all the stores in the computation. In sccp all fair computations from a configuration have the same result [18]. Thus, the *observable behaviour* of P with initial store c, written $\mathscr{O}(P,c)$, is defined as the result of any fair computation starting from $\langle P, c\rangle$. Now consider a setting where in addition to their sccp capabilities in [18], processes can also create new agents. Hence, unboundedly many agents, say agents $1,2,\ldots$, may be created during an infinite computation. In this case, $\mathscr{O}(P,c) \sqsupseteq \Delta_{\mathbb{N}}(false)$, where $\mathbb{N}$ is the set of natural numbers, would imply that some (finite or infinite) set of agents in any fair computation from $\langle P, c\rangle$ may reach contradictory local information among them. Notice that from the above-mentioned properties of distributed spaces, the existence of a finite set of agents $H \subseteq \mathbb{N}$ such that $\mathscr{O}(P,c) \sqsupseteq \Delta_H(false)$ implies $\mathscr{O}(P,c) \sqsupseteq \Delta_{\mathbb{N}}(false)$. The converse of this implication will be called *group compactness* and we will discuss meaningful sufficient conditions for it to hold.

In the next sections we will describe the above spatial and epistemic notions in more detail.

3 Background

We presuppose basic knowledge of domain and order theory [3, 1, 7] and use the following notions. Let $\mathbf{C}$ be a poset $(Con, \sqsubseteq)$, and let $S \subseteq Con$. We use $\bigsqcup S$ to denote the least upper bound (or *supremum* or *join*) of the elements in S, and $\bigsqcap S$ is the greatest lower bound (glb) (*infimum* or *meet*) of the elements in S. An element $e \in S$ is the *greatest element* of S iff for every element $e' \in S$, $e' \sqsubseteq e$. If such e exists, we denote it by $max\, S$. As usual, if $S = \{c, d\}$, $c \sqcup d$ and $c \sqcap d$ represent $\bigsqcup S$ and $\bigsqcap S$, respectively. If $S = \emptyset$, we denote $\bigsqcup S = true$ and $\bigsqcap S = false$. We say that $\mathbf{C}$ is a *complete lattice* iff each subset of Con has a supremum in Con. The poset $\mathbf{C}$ is *distributive* iff for every $a, b, c \in Con$, $a \sqcup (b \sqcap c) = (a \sqcup b) \sqcap (a \sqcup c)$. A non-empty set $S \subseteq Con$ is *directed* iff for every pair of elements $x, y \in S$, there exists $z \in S$ such that $x \sqsubseteq z$ and $y \sqsubseteq z$, or iff every *finite* subset of S has an upper bound in S. Also $c \in Con$ is *compact* iff for any directed subset D of Con, $c \sqsubseteq \bigsqcup D$ implies $c \sqsubseteq d$ for some $d \in D$. A *self-map* on Con is a function f from Con to Con. Let $(Con, \sqsubseteq)$ be a complete lattice. The self-map f on Con *preserves* the join of a set $S \subseteq Con$ iff $f(\bigsqcup S) = \bigsqcup\{f(c) \mid c \in S\}$. A self-map that preserves the join of finite sets is called *join-homomorphism*. A self-map f on Con is *monotonic* if $a \sqsubseteq b$ implies $f(a) \sqsubseteq f(b)$. We say that f *distributes* over joins (or that f *preserves* joins) iff it preserves the join of arbitrary sets. A self-map f on Con is *continuous* iff it preserves the join of any directed set.

Constraint systems [24] are semantic structures to specify partial information. They can be formalized as complete lattices [2].

Definition 3.1 (Constraint Systems [2])**.** *A* constraint system *(cs)* $\mathbf{C}$ *is a complete lattice* $(Con, \sqsubseteq)$. *The elements of Con are called* constraints. *The symbols* $\sqcup$, *true and false will be used to denote the least upper bound (lub) operation, the bottom, and the top element of* $\mathbf{C}$.

The elements of the lattice, the *constraints*, represent (partial) information. A constraint c can be viewed as an *assertion*. The lattice order $\sqsubseteq$ is meant to capture entailment of information: $c \sqsubseteq d$, alternatively written $d \sqsupseteq c$, means that the assertion d represents at least as much information as c. We think of $d \sqsupseteq c$ as saying that d *entails* c or that c can be *derived* from d. The operator $\sqcup$ represents join of information; $c \sqcup d$ can be seen as an assertion stating that both c and d hold. We can think of $\sqcup$ as representing conjunction of assertions. The top element represents the join of all, possibly inconsistent, information, hence it is referred to as *false*. The bottom element *true* represents *empty information*. We say that c is *consistent* if $c \neq false$, otherwise we say that c is *inconsistent*. Similarly, we say that c is consistent/inconsistent with d if $c \sqcup d$ is consistent/inconsistent.

Constraint Frames. One can define a general form of implication by adapting the corresponding notion from Heyting Algebras to cs. A *Heyting implication* $c \rightarrow d$ in our setting corresponds to the *weakest constraint* one needs to join c with to derive d.

Definition 3.2 (Constraint Frames [8])**.** *A constraint system* $(Con, \sqsubseteq)$ *is said to be a* constraint frame *iff its joins distribute over arbitrary meets. More precisely,* $c \sqcup \bigsqcap S = \bigsqcap\{c \sqcup e \mid e \in S\}$ *for every* $c \in Con$ *and* $S \subseteq Con$. *Define* $c \rightarrow d$ *as* $\bigsqcap\{e \in Con \mid c \sqcup e \sqsupseteq d\}$.

The following properties of Heyting implication correspond to standard logical properties (with $\rightarrow$, $\sqcup$, and $\sqsupseteq$ interpreted as implication, conjunction, and entailment).

Proposition 3.3 ([8])**.** *Let* $(Con, \sqsubseteq)$ *be a constraint frame. For every* $c, d, e \in Con$ *the following holds: (1)* $c \sqcup (c \rightarrow d) = c \sqcup d$, *(2)* $(c \rightarrow d) \sqsubseteq d$, *(3)* $c \rightarrow d = true$ *iff* $c \sqsupseteq d$.

4 Space and Beliefs

The authors of [18] extended the notion of cs to account for distributed and multi-agent scenarios with a finite number of agents, each having their own space for local information and their computations. The extended structures are called spatial cs (scs). Here we adapt scs to reason about possibly infinite groups of agents.

A *group* G is a set of agents. Each $i \in G$ has a *space* function $\mathfrak{s}_i : Con \to Con$ satisfying some structural conditions. Recall that constraints can be viewed as assertions. Thus given $c \in Con$, we can then think of the constraint $\mathfrak{s}_i(c)$ as an assertion stating that c is a piece of information residing *within a space of agent i*. Some alternative *epistemic* interpretations of $\mathfrak{s}_i(c)$ is that it is an assertion stating that agent i *believes* c, that c holds within the space of agent i, or that agent i *interprets* c as $\mathfrak{s}_i(c)$. All these interpretations convey the idea that c is local or subjective to agent i.

In [18] scs are used to specify the spatial distribution of information in configurations $\langle P, c\rangle$ where P is a process and c is a constraint. E.g., a reduction $\langle\, P, \mathfrak{s}_i(c) \sqcup \mathfrak{s}_j(d)\,\rangle \longrightarrow \langle\, Q, \mathfrak{s}_i(c) \sqcup \mathfrak{s}_j(d \sqcup e)\,\rangle$ means that P with c in the space of agent i and d in the space of agent j can evolve to Q while adding e to the space of agent j.

We now introduce the notion of space function.

Definition 4.1 (Space Functions)**.** *A* space function *over a cs* $(Con, \sqsubseteq)$ *is a* continuous *self-map* $f : Con \to Con$ *s.t. for every* $c, d \in Con$ *(S.1)* $f(true) = true$*, (S.2)* $f(c \sqcup d) = f(c) \sqcup f(d)$*. We shall use* $\mathscr{S}(\mathbf{C})$ *to denote the set of all space functions over* $\mathbf{C} = (Con, \sqsubseteq)$*.*

The assertion $f(c)$ can be viewed as saying that c is in the space represented by f. Property S.1 states that having an empty local space amounts to nothing. Property S.2 allows us to join and distribute the information in the space represented by f.

In [18] space functions were not required to be continuous. Nevertheless, continuity comes naturally in the intended phenomena we wish to capture: modelling information of possibly *infinite* groups. In fact, in [18] scs could only have finitely many agents.

In [9] we extended scs to allow arbitrary, possibly infinite, sets of agents. A *spatial cs* is a cs with a possibly infinite group of agents each having a space function.

Definition 4.2 (Spatial Constraint Systems)**.** *A* spatial cs (scs) *is a cs* $\mathbf{C} = (Con, \sqsubseteq)$ *equipped with a possibly infinite tuple* $\mathfrak{s} = (\mathfrak{s}_i)_{i \in G}$ *of space functions from* $\mathscr{S}(\mathbf{C})$*.*

We shall use $(Con, \sqsubseteq, (\mathfrak{s}_i)_{i \in G})$ *to denote an scs with a tuple* $(\mathfrak{s}_i)_{i \in G}$*. We refer to G and* $\mathfrak{s}$ *as the* group of agents *and* space tuple *of* $\mathbf{C}$ *and to each* $\mathfrak{s}_i$ *as the* space function *in* $\mathbf{C}$ *of agent i. Subsets of G are also referred to as groups of agents (or sub-groups of G).*

Let us illustrate a simple scs that will be used throughout the paper.

Example 4.3. The scs $(Con, \sqsubseteq, (\mathfrak{s}_i)_{i \in \{1,2\}})$ in Fig.1 is given by the complete lattice $\mathbf{M}_2$ and two agents. We have $Con = \{p \vee \neg p, p, \neg p, p \wedge \neg p\}$ and $c \sqsubseteq d$ iff c is a logical consequence of d. The top element *false* is $p \wedge \neg p$, the bottom element *true* is $p \vee \neg p$, and the constraints p and $\neg p$ are incomparable with each other. The set of agents is $\{1, 2\}$ with space functions $\mathfrak{s}_1$ and $\mathfrak{s}_2$: For agent 1, $\mathfrak{s}_1(p) = \neg p$, $\mathfrak{s}_1(\neg p) = p$, $\mathfrak{s}_1(false) = false$, $\mathfrak{s}_1(true) = true$, and for agent 2, $\mathfrak{s}_2(p) = false = \mathfrak{s}_2(false)$, $\mathfrak{s}_2(\neg p) = \neg p$, $\mathfrak{s}_2(true) = true$. The intuition is that the agent 2 sees no difference between p and *false* while agent 1 interprets $\neg p$ as p and vice versa.

More involved examples of scs include meaningful families of structures from logic and economics such as Kripke structures and Aumann structures (see [18]). We illustrate scs with infinite groups in the next section.

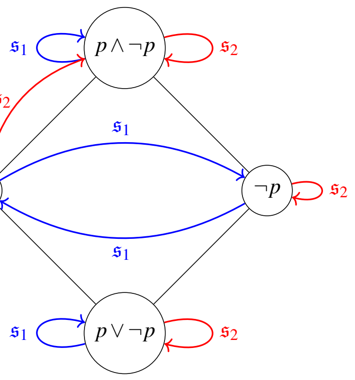


Figure 1: Cs given by lattice $\mathbf{M}_2$ ordered by implication and space functions $\mathfrak{s}_1$ and $\mathfrak{s}_2$.

5 Extrusion and Utterances

We can also equip each agent i with an *extrusion* function $\mathfrak{e}_i : Con \to Con$. Intuitively, within a space context $\mathfrak{s}_i(\cdot)$, the assertion $\mathfrak{e}_i(c)$ specifies that c must be posted outside of agent i's space. This is captured by requiring the *extrusion* axiom (E.1) $\mathfrak{s}_i(\mathfrak{e}_i(c)) = c$. In other words, we view *extrusion/utterance* as the right inverse of *space/belief* (and thus space/belief as the left inverse of extrusion/utterance).

Definition 5.1 (Extrusion)**.** *Given an scs $(Con, \sqsubseteq, (\mathfrak{s}_i)_{i \in G})$, we say that $\mathfrak{e}_i$ is an extrusion function for the space $\mathfrak{s}_i$ iff $\mathfrak{e}_i$ is a right inverse of $\mathfrak{s}_i$, i.e., iff $\mathfrak{s}_i(\mathfrak{e}_i(c)) = c$.* □

From the above definitions it follows that $\mathfrak{s}_i(c \sqcup \mathfrak{e}_i(d)) = \mathfrak{s}_i(c) \sqcup d$. From a spatial point of view, agent i *extrudes* d from its local space. From an epistemic view this can be seen as an agent i that believes c and *utters* d to the outside world. If d is inconsistent with c, i.e., $c \sqcup d = false$, we can see the utterance as an intentional *lie* by agent i: The agent i utters an assertion inconsistent with their own beliefs.

Example 5.2. Let $e = \mathfrak{s}_i(c \sqcup \mathfrak{e}_i(\mathfrak{s}_j(a))) \sqcup \mathfrak{s}_j(d)$. The constraint e specifies that agent i has c and wishes to transmit, via extrusion, a addressed to agent j. Agent j has d in their space. Indeed, with the help of E.1 and S.2, we can derive $e \sqsupseteq \mathfrak{s}_j(d \sqcup a)$ thus stating that e entails that a will be in the space of j.

The Extrusion Problem. A legitimate question is: Given space $\mathfrak{s}_i$ can we derive an extrusion function $\mathfrak{e}_i$ for it ? From set theory we know that there is an extrusion function (i.e., a right inverse) $\mathfrak{e}_i$ for $\mathfrak{s}_i$ iff $\mathfrak{s}_i$ is *surjective*. Recall that the *pre-image* of $y \in Y$ under $f : X \to Y$ is the set $f^{-1}(y) = \{x \in X \mid y = f(x)\}$. Thus the extrusion $\mathfrak{e}_i$ can be defined as a function, called *choice* function, that maps each element c to some element from the pre-image of c under $\mathfrak{s}_i$.

The existence of the above-mentioned choice function assumes the *Axiom of Choice*. The next proposition from [8] gives some constructive extrusion functions. It also identifies a distinctive property of space functions for which a right inverse exists.

Proposition 5.3 ([8])**.** *Let f be a space function over $(Con, \sqsubseteq)$. Then*

1. *If $f(false) \neq false$ then f does not have any right inverse.*
2. *If f is surjective then $g : c \mapsto \bigsqcup f(c)^{-1}$ is a right inverse of f that preserves arbitrary infima.*
3. *If f is surjective and preserves arbitrary infima then $h : c \mapsto \bigsqcap f(c)^{-1}$ is a right inverse of f that preserves arbitrary suprema.*

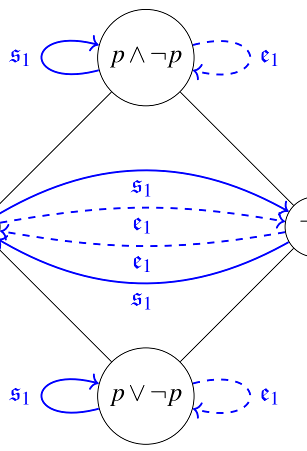


Figure 2: Cs given by lattice $\mathbf{M}_2$ ordered by implication and the space function $\mathfrak{s}_1$ with extrusion $\mathfrak{e}_1$.

The following example illustrates an application of Prop.5.3 to obtain an extrusion function for the space function $\mathfrak{s}_1$ from Ex.4.3. Notice that the space function $\mathfrak{s}_2$ from Ex.4.3 is not surjective thus it does not have an extrusion function.

Example 5.4. Fig.2 shows an extrusion function for the space function $\mathfrak{s}_1$ in Ex.4.3. This extrusion function can be obtained by applying Prop.5.3.2.

6 Groups and Distributed Knowledge

In [9] we introduced the notion of collective information of a group of agents. Roughly speaking, the *distributed (or collective) information* of a group I is the join of each piece of information that resides in the space of *some* $i \in I$. The distributed information of I w.r.t. c is the distributive information of I that can be derived from c. We wish to formalize whether a given e can be derived from the collective information of the group I w.r.t. c.

The following examples, which we will use throughout this section, illustrate the above intuition.

Example 6.1. Consider an scs $(Con, \sqsubseteq, (\mathfrak{s}_i)_{i \in G})$ where $G = \mathbb{N}$ and $(Con, \sqsubseteq)$ is a constraint frame. Let $c \stackrel{\texttt{def}}{=} \mathfrak{s}_1(a) \sqcup \mathfrak{s}_2(a \to b) \sqcup \mathfrak{s}_3(b \to e)$. The constraint c specifies the situation where $a, a \to b$ and $b \to e$ are in the spaces of agent 1, 2 and 3, respectively. Neither agent necessarily holds e in their space in c. Nevertheless, the information e can be derived from the collective information of the three agents w.r.t. c, since from Prop.3.3 we have $a \sqcup (a \to b) \sqcup (b \to e) \sqsupseteq e$. Let us now consider an example with infinitely many agents. Let $c' \stackrel{\texttt{def}}{=} \bigsqcup_{i \in \mathbb{N}} \mathfrak{s}_i(a_i)$ for some increasing chain $a_0 \sqsubseteq a_1 \sqsubseteq \dots$. Take e' s.t. $e' \sqsubseteq \bigsqcup_{i \in \mathbb{N}} a_i$. Notice that unless e' is compact (see Section 3), it may be the case that no agent $i \in \mathbb{N}$ holds e' in their space; e.g., if $e' \sqsupset a_i$ for any $i \in \mathbb{N}$. Yet, from our assumption, e' can be derived from the collective information w.r.t. c' of all the agents in $\mathbb{N}$, i.e., $\bigsqcup_{i \in \mathbb{N}} a_i$.

The above example may suggest that the distributed information can be obtained by joining individual local information derived from c. Individual information of an agent i can be characterized as the i-projection of c defined thus:

Definition 6.2 (Agent and Join Projections [9]). *Let* $\mathbf{C} = (Con, \sqsubseteq, (\mathfrak{s}_i)_{i \in G})$ *be an scs. Given* $i \in G$, *the* i*-agent projection of* $c \in Con$ *is defined as* $\pi_i(c) \stackrel{\texttt{def}}{=} \bigsqcup\{e \mid c \sqsupseteq \mathfrak{s}_i(e)\}$. *We say that* e *is* i*-agent derivable* *from c iff $\pi_i(c) \sqsupseteq e$. Given $I \subseteq G$ the I-join projection of a group I of c is defined as $\pi_I(c) \stackrel{\texttt{def}}{=} \bigsqcup\{\pi_i(c) \mid i \in I\}$. We say that e is I-join derivable from c iff $\pi_I(c) \sqsupseteq e$.*

The i-projection of an agent i of c naturally represents the join of all the information of agent i in c. It turns out that projections are extrusion functions: If $\mathfrak{s}_i$ admits extrusion then π_i is an extrusion function for the space $\mathfrak{s}_i$ (see Def.5.1). More precisely,

Proposition 6.3 (Projection as extrusion). *If $\mathfrak{s}_i$ is surjective then $\mathfrak{s}_i(\pi_i(c)) = c$ for every $c \in Con$.*

The I-join projection of group I joins individual i-projections of c for $i \in I$. This projection can be used as a sound mechanism for reasoning about distributed-information: If e is I-join derivable from c then it follows from the distributed-information of I w.r.t. c.

Example 6.4. Let c be as in Ex.6.1. We have $\pi_1(c) \sqsupseteq a$, $\pi_2(c) \sqsupseteq (a \to b)$, $\pi_3(c) \sqsupseteq (b \to e)$. Indeed e is I-join derivable from c since $\pi_{\{1,2,3\}}(c) = \pi_1(c) \sqcup \pi_2(c) \sqcup \pi_3(c) \sqsupseteq e$. Similarly we conclude that e' is I-join derivable from c' in Ex.6.1 since $\pi_{\mathbb{N}}(c') = \bigsqcup_{i \in \mathbb{N}} \pi_i(c) \sqsupseteq \bigsqcup_{i \in \mathbb{N}} a_i \sqsupseteq e'$.

Nevertheless, I-join projections do not provide a complete mechanism for reasoning about distributed information as illustrated below.

Example 6.5. Let $d \stackrel{\texttt{def}}{=} \mathfrak{s}_1(b) \sqcap \mathfrak{s}_2(b)$. Recall that we think of $\sqcup$ and $\sqcap$ as conjunction and disjunction of assertions: d specifies that b is present in the space of agent 1 or in the space of agent 2 though not exactly in which one. Thus from d we should be able to conclude that b belongs to the space of *some* agent in $\{1,2\}$. Nevertheless, in general b is not I-join derivable from d since from $\pi_{\{1,2\}}(d) = \pi_1(d) \sqcup \pi_2(d)$ we cannot, in general, derive b. To see this consider the scs in Fig.3a and take $b = \neg p$. We have $\pi_{\{1,2\}}(d) = \pi_1(d) \sqcup \pi_2(d) = true \sqcup true = true \not\sqsupseteq b$. One can generalize the example to infinitely many agents: Consider the scs in Ex.6.1. Let $d' \stackrel{\texttt{def}}{=} \bigsqcap_{i \in \mathbb{N}} \mathfrak{s}_i(b')$. We should be able to conclude from d' that b' is in the space of *some* agent in $\mathbb{N}$ but, in general, b' is not $\mathbb{N}$-join derivable from d'.

6.1 Distributed Spaces

In the previous section we illustrated that the I-join projection of c, $\pi_I(c)$, the join of individual projections, may not project all distributed information of a group I. To solve this problem we shall develop the notion of I-group projection of c, written as $\Pi_I(c)$. To do this we shall first define a space function Δ_I called the distributed space of group I. The function Δ_I can be thought of as a virtual space including all the information that can be in the space of a member of I. We shall then define an I-projection Π_I in terms of Δ_I much like π_i is defined in terms of $\mathfrak{s}_i$.

Recall that $\mathscr{S}(\mathbf{C})$ denotes the set of all space functions over a cs $\mathbf{C}$. For notational convenience, we shall use $(f_I)_{I \subseteq G}$ to denote the tuple $(f_I)_{I \in \mathscr{P}(G)}$ of elements of $\mathscr{S}(\mathbf{C})$.

Set of Space Functions. We begin by introducing a new partial order induced by $\mathbf{C}$. The set of space functions ordered point-wise.

Definition 6.6 (Space Functions Order). *Let $\mathbf{C} = (Con, \sqsubseteq, (\mathfrak{s}_i)_{i \in G})$ be an scs. Given $f, g \in \mathscr{S}(\mathbf{C})$, define $f \sqsubseteq_{\mathcal{S}} g$ iff $f(c) \sqsubseteq g(c)$ for every $c \in Con$. We shall use $\mathbf{C}_{\mathcal{S}}$ to denote the partial order $(\mathscr{S}(\mathbf{C}), \sqsubseteq_{\mathcal{S}})$; the set of all space functions ordered by $\sqsubseteq_{\mathcal{S}}$.*

A very important fact for the design of our structure is that the set of space functions $\mathscr{S}(\mathbf{C})$ can be made into a complete lattice.

Lemma 6.7 ([9]). *Let $\mathbf{C} = (Con, \sqsubseteq, (\mathfrak{s}_i)_{i \in G})$ be an scs. Then $\mathbf{C}_{\mathcal{S}}$ is a complete lattice.*

6.2 Distributed Spaces as Maximum Spaces

Let us consider the lattice of space functions $\mathbf{C_S} = (\mathscr{S}(\mathbf{C}), \sqsubseteq_{\mathtt{S}})$. Suppose that f and g are space functions in $\mathbf{C_S}$ with $f \sqsubseteq_{\mathtt{S}} g$. Intuitively, every piece of information c in the space represented by g is also in the space represented by f since $f(c) \sqsubseteq g(c)$ for every $c \in Con$. This can be interpreted as saying that the space represented by g is included in the space represented by f; in other words the bigger the space, the smaller the function that represents it in the lattice $\mathbf{C_S}$.

Following the above intuition, the order relation $\sqsubseteq_{\mathtt{S}}$ of $\mathbf{C_S}$ represents (reverse) space inclusion and the join and meet operations in $\mathbf{C_S}$ represent intersection and union of spaces. The biggest and the smallest spaces are represented by the bottom and the top elements of the lattice $\mathbf{C_S}$, here called $\lambda_\perp$ and $\lambda_\top$ and defined as follows.

Definition 6.8 (Top and Bottom Spaces). *For every $c \in Con$, define $\lambda_\perp(c) \stackrel{\mathtt{def}}{=} true$, $\lambda_\top(c) \stackrel{\mathtt{def}}{=} true$ if $c = true$ and $\lambda_\top(c) \stackrel{\mathtt{def}}{=} false$ if $c \neq true$.*

The distributed space Δ_I of a group I can be viewed as the function that represents the smallest space that includes all the local information of the agents in I. From the above intuition, Δ_I should be the *greatest space function* below the space functions of the agents in I. The existence of such a function follows from completeness of $(\mathscr{S}(\mathbf{C}), \sqsubseteq_{\mathtt{S}})$ (Lemma 6.7).

Definition 6.9 (Distributed Spaces [9]). *Let $\mathbf{C}$ be an scs $(Con, \sqsubseteq, (\mathfrak{s}_i)_{i\in G})$. The* distributed spaces *of $\mathbf{C}$ is given by $\Delta = (\Delta_I)_{I\subseteq G}$ where $\Delta_I \stackrel{\mathtt{def}}{=} max\{f \in \mathscr{S}(\mathbf{C}) \mid f \sqsubseteq_{\mathtt{S}} \mathfrak{s}_i$ for every $i \in I\}$. We shall say that e is distributed among $I \subseteq G$ w.r.t. c iff $c \sqsupseteq \Delta_I(e)$. We shall refer to each Δ_I as the* (distributed) space *of the group I.*

It follows from Lemma 6.7 that $\Delta_I = \bigsqcap\{\mathfrak{s}_i \mid i \in I\}$ (where $\bigsqcap$ is the meet in the complete lattice $(\mathscr{S}(\mathbf{C}), \sqsubseteq_{\mathtt{S}})$). Fig.3b illustrates an scs and its distributed space $\Delta_{\{1,2\}}$.

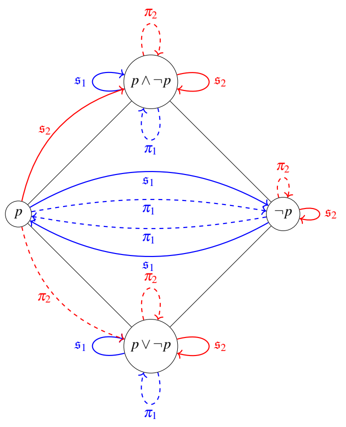
Compositionality. Distributed spaces have pleasant compositional properties. They capture the intuition that the *distributed information* of a group I can be obtained from the the distributive information of its subgroups.

Theorem 6.10 ([9]). *Let $(\Delta_I)_{I\subseteq G}$ be the distributed spaces of an scs $(Con, \sqsubseteq, (\mathfrak{s}_i)_{i\in G})$. Suppose that $K, J \subseteq I \subseteq G$. (1) $\Delta_I = \lambda_\top$ if $I = \emptyset$, (2) $\Delta_I = \mathfrak{s}_i$ if $I = \{i\}$, (3) $\Delta_J(a) \sqcup \Delta_K(b) \sqsupseteq \Delta_I(a \sqcup b)$, and (4) $\Delta_J(a) \sqcup \Delta_K(a \rightarrow c) \sqsupseteq \Delta_I(c)$ if $(Con, \sqsubseteq)$ is a constraint frame.*

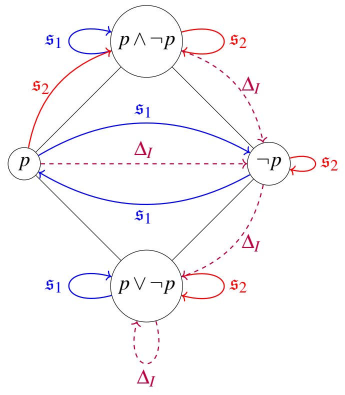
Recall that $\lambda_\top$ corresponds to the empty space (see Def.6.8). The first property realizes the intuition that the empty subgroup $\emptyset$ *does not* have any information whatsoever distributed w.r.t. a consistent c: for if $c \sqsupseteq \Delta_\emptyset(e)$ and $c \neq false$ then $e = true$. Intuitively, the second property says that the function Δ_I for the group of one agent must be the agent's space function. The third property states that a group can join the information of its subgroups. The last property uses constraint implication, hence the constraint frame condition, to express that by joining the information a and $a \rightarrow c$ of their subgroups, the group I can obtain c.

Let us illustrate how to derive information of a group from smaller ones using Thm.6.10.

Example 6.11. Let $c = \mathfrak{s}_1(a) \sqcup \mathfrak{s}_2(a \rightarrow b) \sqcup \mathfrak{s}_3(b \rightarrow e)$ as in Ex.6.1. We want to prove that e is distributed among $I = \{1,2,3\}$ w.r.t. c, i.e., $c \sqsupseteq \Delta_{\{1,2,3\}}(e)$. Using Properties 2 and 4 in Thm.6.10 we obtain $c \sqsupseteq \mathfrak{s}_1(a) \sqcup \mathfrak{s}_2(a \rightarrow b) = \Delta_{\{1\}}(a) \sqcup \Delta_{\{2\}}(a \rightarrow b) \sqsupseteq \Delta_{\{1,2\}}(b)$, and then $c \sqsupseteq \Delta_{\{1,2\}}(b) \sqcup \mathfrak{s}_3(b \rightarrow e) = \Delta_{\{1,2\}}(b) \sqcup \Delta_{\{3\}}(b \rightarrow e) \sqsupseteq \Delta_{\{1,2,3\}}(e)$ as wanted.



(a) Projections π_1 and π_2 given $\mathfrak{s}_1$ and $\mathfrak{s}_2$.



(b) Δ_I with $I = \{1,2\}$ given $\mathfrak{s}_1$ and $\mathfrak{s}_2$.

Figure 3: Projections (a) and Distributed Space function (b) over lattice $\mathbf{M}_2$.

Remark 6.1 (Continuity). The example with infinitely many agents in Ex.6.1 illustrates well why we require our spaces to be continuous in the presence of possibly infinite groups. Clearly $c' = \bigsqcup_{i\in\mathbb{N}} \mathfrak{s}_i(a_i) \sqsupseteq \bigsqcup_{i\in\mathbb{N}} \Delta_{\mathbb{N}}(a_i)$. By continuity, $\bigsqcup_{i\in\mathbb{N}} \Delta_{\mathbb{N}}(a_i) = \Delta_{\mathbb{N}}(\bigsqcup_{i\in\mathbb{N}} a_i)$ which indeed captures the idea that each a_i is in the distributed space $\Delta_{\mathbb{N}}$.

We conclude this subsection with an important family of scs from mathematical economics: Aumann structures. We illustrate that the notion of distributed knowledge in these structures is an instance of a distributed space.

Example 6.12. *Aumann Constraint Systems.* Aumann structures [16] are an *event-based* approach to modelling knowledge. An Aumann structure is a tuple $\mathscr{A} = (S, \mathscr{P}_1, \ldots, \mathscr{P}_n)$ where S is a set of states and each $\mathscr{P}_i$ is a partition on S for agent i. The partitions are called *information sets*. If two states t and u are in the same information set for agent i, it means that in state t agent i considers state u possible, and vice versa. An *event* in an Aumann structure is any subset of S. Event e holds at state t if $t \in e$. The set $\mathscr{P}_i(s)$ denotes the information set of $\mathscr{P}_i$ containing s. The event of *agent i knowing e* is defined as $\mathsf{K}_i(e) = \{s \in S \mid \mathscr{P}_i(s) \subseteq e\}$, and the *distributed knowledge of an event e among the agents in a group I* is defined as $\mathsf{D}_I(e) = \{s \in S \mid \bigcap_{i\in I} \mathscr{P}_i(s) \subseteq e\}$.

An Aumann structure can be seen as a spatial constraint system $\mathbf{C}(\mathscr{A})$ with events as constraints, i.e., $Con = \{e \mid e \text{ is an event in } \mathscr{A}\}$, and for every $e_1, e_2 \in Con$, $e_1 \sqsubseteq e_2$ iff $e_2 \subseteq e_1$. The operators join ($\sqcup$) and meet ($\sqcap$) are intersection ($\cap$) and union ($\cup$) of events, respectively; $true = S$ and $false = \emptyset$. The space functions are the knowledge operators, i.e., $\mathfrak{s}_i(c) = \mathsf{K}_i(c)$. From these definitions and since meets are unions one can easily verify that $\Delta_I(c) = \mathsf{D}_I(c)$ which shows the correspondence between distributed information and distributed knowledge.

6.3 Group Projections

As promised in Section 6.1 we now give a definition of *Group Projection*. The function $\Pi_I(c)$ extracts exactly all information that the group I may have distributed w.r.t. c.

Definition 6.13 (Group Projection [9]). *Let $(\Delta_I)_{I\subseteq G}$ be the distributed spaces of an scs $\mathbf{C} = (Con, \sqsubseteq, (\mathfrak{s}_i)_{i\in G})$. Given the set $I \subseteq G$, the I-group projection of $c \in Con$ is defined as $\Pi_I(c) \overset{\texttt{def}}{=} \bigsqcup\{e \mid c \sqsupseteq \Delta_I(e)\}$. We say that e is I-group derivable from c iff $\Pi_I(c) \sqsupseteq e$.*

Much like space functions and agent projections, group projections and distributed spaces also form a pleasant correspondence: a Galois connection [3].

Proposition 6.14 ([9]). *Let $(\Delta_I)_{I\subseteq G}$ be the distributed spaces of $\mathbf{C} = (Con, \sqsubseteq, (\mathfrak{s}_i)_{i\in G})$. For every $c, e \in Con$, (1) $c \sqsupseteq \Delta_I(e)$ iff $\Pi_I(c) \sqsupseteq e$, (2) $\Pi_I(c) \sqsupseteq \Pi_J(c)$ if $J \subseteq I$, and (3) $\Pi_I(c) \sqsupseteq \pi_I(c)$.*

The first property in Prop.6.14, a Galois connection, states that we can conclude from c that e is in the distributed space of I exactly when e is I-group derivable from c. The second says that the bigger the group, the bigger the projection. The last property says that whatever is I-join derivable is I-group derivable, although the opposite is not true as shown in Ex.6.5.

6.4 Group Compactness

Suppose that an *infinite* group of agents I can derive e from c (i.e., $c \sqsupseteq \Delta_I(e)$). A legitimate question is whether there exists a *finite* sub-group J of agents from I that can also derive e from c. The following theorem provides a positive answer to this question provided that e is a compact element and I-join derivable from c.

Theorem 6.15 (Group Compactness [9]). *Let $(\Delta_I)_{I\subseteq G}$ be the distributed spaces of an scs $\mathbf{C} = (Con, \sqsubseteq, (\mathfrak{s}_i)_{i\in G})$. Suppose that $c \sqsupseteq \Delta_I(e)$. If e is compact and I-join derivable from c then there exists a finite set $J \subseteq I$ such that $c \sqsupseteq \Delta_J(e)$.*

We conclude this section with the following example of group compactness.

Example 6.16. Consider the example with infinitely many agents in Ex.6.1. We have $c' = \bigsqcup_{i\in\mathbb{N}} \mathfrak{s}_i(a_i)$ for some increasing chain $a_0 \sqsubseteq a_1 \sqsubseteq \ldots$ and e' s.t. $e' \sqsubseteq \bigsqcup_{i\in\mathbb{N}} a_i$. Notice that $c' \sqsupseteq \Delta_{\mathbb{N}}(e')$ and $\pi_{\mathbb{N}}(c') \sqsupseteq e'$. Hence e' is $\mathbb{N}$-join derivable from c'. If e' is compact, by Thm.6.15 there must be a finite subset $J \subseteq \mathbb{N}$ such that $c' \sqsupseteq \Delta_J(e')$.

7 Computing Distributed Information

Let us consider a *finite* scs $\mathbf{C} = (Con, \sqsubseteq, (\mathfrak{s}_i)_{i\in G})$ with distributed spaces $(\Delta_I)_{I\subseteq G}$. By finite scs we mean that Con and G are finite sets. Let us consider the problem of computing Δ_I: Given a set $\{\mathfrak{s}_i\}_{i\in I}$ of space functions, we wish to find the greatest space function f such that $f \sqsubseteq \mathfrak{s}_i$ for all $i \in I$ (see Def.6.9).

Because of the finiteness assumption, the above problem can be rephrased in simpler terms: *Given a finite lattice L and a finite set S of join-homomorphisms on L, find the greatest join-homomorphism below all the elements of S.* Even in small lattices with four elements and two space functions, finding such greatest function may not be immediate, e.g., for $S = \{\mathfrak{s}_1, \mathfrak{s}_2\}$ and the lattice in Fig.1 the answer is given Fig.3b.

A *brute force* approach would be to compute $\Delta_I(c)$ by generating the set $\{f(c) \mid f \in \mathscr{S}(\mathbf{C})$ and $f \sqsubseteq \mathfrak{s}_i$ for all $i \in I\}$ and taking its join. This approach works since $(\bigsqcup S)(c) = \bigsqcup\{f(c) | f \in S\}$. However, the number of such functions in $\mathscr{S}(\mathbf{C})$ can be at least factorial in the size of Con. For distributive lattices, the size of $\mathscr{S}(\mathbf{C})$ can be non-polynomial in the size of Con.

Proposition 7.1 ([9]). *For every $n \geq 2$, there exists a lattice $\mathbf{C} = (Con, \sqsubseteq)$ such that $|\mathscr{S}(\mathbf{C})| \geq (n-2)!$ and $n = |Con|$. For every $n \geq 1$, there exists a distributed lattice $\mathbf{C} = (Con, \sqsubseteq)$ such that $|\mathscr{S}(\mathbf{C})| \geq n^{\log_2 n}$ and $n = |Con|$.*

Nevertheless, we can exploit order theoretical results and compositional properties of distributive spaces to compute Δ_I in polynomial time in the size of Con.

Theorem 7.2 ([9]). *Suppose that $(Con, \sqsubseteq)$ is a distributed lattice. Let J and K be two sets such that $I = J \cup K$. Then the following equalities hold:*

$$1.\ \Delta_I(c) = \bigsqcap\{\Delta_J(a) \sqcup \Delta_K(b) \mid a,b \in Con \text{ and } a \sqcup b \sqsupseteq c\}. \quad (1)$$

$$2.\ \Delta_I(c) = \bigsqcap\{\Delta_J(a) \sqcup \Delta_K(a \rightarrow c) \mid a \in Con\}. \quad (2)$$

$$3.\ \Delta_I(c) = \bigsqcap\{\Delta_J(a) \sqcup \Delta_K(a \rightarrow c) \mid a \in Con \text{ and } a \sqsubseteq c\}. \quad (3)$$

The above theorem characterizes the information of a group from that of its subgroups. It bears witness to the inherent compositional nature of our notion of distributed space, and realizes the intuition that by joining the information a and $a \rightarrow c$ of their subgroups, the group I can obtain c. This compositional nature is exploited by the algorithms below.

Given a finite scs $\mathbf{C} = (Con, \sqsubseteq, (\mathfrak{s}_i)_{i \in G})$, the recursive function DELTAPART3(I,c) in Algorithm 1 computes $\Delta_I(c)$ for any given c in Con. Its correctness, assuming that $(Con, \sqsubseteq)$ is a distributed lattice, follows from Thm.7.2(3). Termination follows from the finiteness of $\mathbf{C}$ and the fact the sets J and K in the recursive calls form a partition of I. Notice that we select a partition (in halves) rather than any two sets K,J satisfying the condition $I = J \cup K$ to avoid significant recalculation.

Algorithm 1 Function DELTAPART3(I,c) computes $\Delta_I(c)$

```
1: function DELTAPART3($I,c$)                                          ▷ Computes $\Delta_I(c)$
2:     if $I = \{i\}$ then
3:         return $\mathfrak{s}_i(c)$
4:     else
5:         $\{J,K\} \leftarrow$ PARTITION($I$)          ▷ returns a partition $\{J,K\}$ of $I$ s.t., $|J| = \lfloor |I|/2 \rfloor$
6:         return $\bigsqcap\{$DELTAPART3($J,a$) $\sqcup$ DELTAPART3($K, a \rightarrow c$) $\mid a \in Con$ and $a \sqsubseteq c\}$.
```

Algorithms. Notice DELTAPART3(I,c) computes $\Delta_I(c)$ using Thm.7.2(3). By modifying Line 6 with the corresponding meet operations, we obtain two variants of DELTAPART3 that use, instead of Thm.7.2(3), the Properties Thm.7.2(1) and Thm.7.2(2). We call them DELTAPART1 and DELTAPART2.

Worst-case time complexity. We assume that binary distributive lattice operations $\sqcap$, $\sqcup$, and $\rightarrow$ are computed in $O(1)$ time. We also assume a fixed group I of size $m = |I|$ and express the time complexity for computing Δ_I in terms of $n = |Con|$, the size of the set of constraints. The above-mentioned algorithms compute the value $\Delta_I(c)$. The worst-case time complexity for computing the function Δ_I is in $O(mn^{1+2\log_2 m})$ using DELTAPART1, and $O(mn^{1+\log_2 m})$ using DELTAPART2 and DELTAPART3 [9].

8 Conclusions and Related Work

We have highlighted some results about scs as semantic structures for spatially-distributed systems exhibiting epistemic behaviour. Our work in scs have been inspired by the seminal work on epistemic logic for knowledge and group knowledge in [15, 6, 16]. Meaningful families of structures from logic and economics such as Kripke structures and Aumann structures have been shown to be instances of scs [18]. Furthermore scs have been used to give semantics to modal logics and process calculi [18, 8, 12].

In [18] we introduced a spatial and epistemic process calculus, called sccp, for reasoning about spatial information and knowledge distributed among the agents of a system. In this work scs were introduced as the domain-theoretical structures to represent spatial and epistemic information. These structures are also used in the denotational and operational semantics of sccp processes. In [18] we also provided operational and denotational techniques for reasoning about the potentially infinite behaviour of spatial and epistemic processes.

In [8, 12] we developed the theory of spatial constraint systems (scs) with extrusion to specify information and processes moving from a space to another. In [11, 10] scs with extrusion are used to give a novel algebraic characterization of the notion of normality in modal logic and to derive right inverse/reverse operators for modal languages. These results were applied to derive new expressiveness results for bisimilarity and well-established modal languages such as Hennessy-Milner logic, and linear-time temporal logic.

In [8, 10, 11, 12] scs are used to reason about beliefs, lies and other epistemic utterances but also restricted to a finite number of agents and individual, rather than group, behaviour of agents.

In [9] we developed semantic foundations and provided algorithms for reasoning about the distributed information of possibly infinite groups in multi-agents systems. We plan to develop similar techniques for reasoning about other group phenomena in multi-agent systems from social sciences and computer music such as group polarization [4] and group improvisation [23].

We have recently learnt that the fundamental operations of dilation and erosion from digital images and mathematical morphology [25] are space and projection functions, respectively. Dilations are applied to figures. Intuitively, figures that are very lightly drawn get thick when dilated. We are currently studying potential applications of distributed spaces in mathematical morphology: E.g., for computing the greatest dilation under a given set of dilations. Similarly, we are also studying scs interpretations of other fundamental operations from mathematical morphology such as opening and closing.

We conclude with some applications of scs in the development of ccp tools and languages. In [14, 13] we described D-SPACES, an implementation of scs that provides property-checking methods as well as an implementation of a specific type of constraint systems (boolean algebras). In [21] we used rewriting logic for specifying and analyzing ccp processes combining spatial and real-time behavior. These processes can run processes in different computational spaces while subject to real-time requirements. The real-time rewriting logic semantics is fully executable in Maude with the help of rewriting modulo SMT: partial information (i.e., constraints) in the specification is represented by quantifier-free formulas on the shared variables of the system that are under the control of SMT decision procedures. The approach is used to symbolically analyze existential real-time reachability properties of process calculi in the presence of spatial hierarchies for sharing information and knowledge. We also developed `dspacenet`, a multi-agent spatial and reactive ccp language for programming academic forums[2]. The fundamental structure of `dspacenet` is that of *space*: A space may contain spatial and reactive ccp programs or other spaces. The fundamental operation of `dspacenet` is that of *program posting*: In each time unit, agents (users) can post spatial reactive ccp programs in the spaces they are allowed to do so. Currently `dspacenet` is used at Univ. Javeriana Cali for teaching spatial reactive declarative programming.

References

[1] Samson Abramsky & Achim Jung (1994): *Domain Theory*. In Samson Abramsky, editor: *Handbook of Logic in Computer Science*, 3, Oxford University Press, pp. 1–168.

[2] Frank S. Boer, Alessandra Di Pierro & Catuscia Palamidessi (1995): *Nondeterminism and infinite computations in constraint programming*. *Theoretical Computer Science*, pp. 37–78, DOI: 10.1016/0304-3975(95)00047-Z.

[2] You can try `dspacenet` at `http://www.dspacenet.com`.

[3] Brian A Davey & Hilary A Priestley (2002): *Introduction to lattices and order*, 2nd edition. Cambridge university press, DOI: `10.1017/CBO9780511809088`.

[4] Joan-Mara Esteban & Debraj Ray (1994): *On the Measurement of Polarization*. Econometrica 62(4), pp. 819–851, DOI: `10.2307/2951734`.

[5] François Fages, Paul Ruet & Sylvain Soliman (2001): *Linear Concurrent Constraint Programming: Operational and Phase Semantics*. Information and Computation, pp. 14–41, DOI: `10.1006/inco.2000.3002`.

[6] Ronald Fagin, Joseph Y Halpern, Yoram Moses & Moshe Y Vardi (1995): *Reasoning about knowledge*, 4th edition. MIT press Cambridge.

[7] Gerhard Gierz, Karl Heinrich Hofmann, Klaus Keimel, Jimmie D. Lawson, Michael Mislove & Dana S. Scott (2003): *Continuous lattices and domains*. Cambridge University Press, DOI: `10.1017/CBO9780511542725`.

[8] Michell Guzmán, Stefan Haar, Salim Perchy, Camilo Rueda & Frank Valencia (2016): *Belief, Knowledge, Lies and Other Utterances in an Algebra for Space and Extrusion*. Journal of Logical and Algebraic Methods in Programming, DOI: `10.1016/j.jlamp.2016.09.001`. Available at `https://hal.inria.fr/hal-01257113`.

[9] Michell Guzmán, Sophia Knight, Santiago Quintero, Sergio Ramírez, Camilo Rueda & Frank Valencia (2019): *Reasoning about Distributed Knowledge of Groups with Infinitely Many Agents*. Research Report, LIX, Ecole polytechnique. Available at `https://hal.archives-ouvertes.fr/hal-02172415`.

[10] Michell Guzman, Salim Perchy, Camilo Rueda & Frank Valencia (2016): *Deriving Inverse Operators for Modal Logic*. In: Theoretical Aspects of Computing – ICTAC 2016, Lecture Notes in Computer Science 9965, Springer, pp. 214–232, DOI: `10.1007/978-3-319-46750-4_13`. Available at `https://hal.inria.fr/hal-01328188`.

[11] Michell Guzmán, Salim Perchy, Camilo Rueda & Frank Valencia (2018): *Characterizing Right Inverses for Spatial Constraint Systems with Applications to Modal Logic*. Theoretical Computer Science 744(56–77), DOI: `10.1016/j.tcs.2018.05.022`. Available at `https://hal.inria.fr/hal-01675010`.

[12] Stefan Haar, Salim Perchy, Camilo Rueda & Frank Valencia (2015): *An Algebraic View of Space/Belief and Extrusion/Utterance for Concurrency/Epistemic Logic*. In: 17th International Symposium on Principles and Practice of Declarative Programming (PPDP 2015), ACM SIGPLAN, pp. 161–172, DOI: `10.1145/2790449.2790520`. Available at `https://hal.inria.fr/hal-01256984`.

[13] Stefan Haar, Salim Perchy & Frank Valencia (2017): *Declarative Framework for Semantical Interpretations of Structured Information - An Applicative Approach*. Int. J. Semantic Computing 11(4), pp. 451–472, DOI: `10.1142/S1793351X17400189`.

[14] Stefan Haar, Salim Perchy & Frank D. Valencia (2017): *D-SPACES: Implementing Declarative Semantics for Spatially Structured Information*. In: ICSC, IEEE Computer Society, pp. 227–233, DOI: `10.1109/ICSC.2017.34`.

[15] Joseph Y Halpern & Yoram Moses (1990): *Knowledge and common knowledge in a distributed environment*. Journal of the ACM (JACM) 37(3), pp. 549–587, DOI: `10.1145/79147.79161`.

[16] Joseph Y Halpern & Richard A Shore (2004): *Reasoning about common knowledge with infinitely many agents*. Information and Computation 191(1), pp. 1–40, DOI: `10.1016/j.ic.2004.01.003`.

[17] Werner Hildenbrand (1970): *On economies with many agents*. Journal of Economic Theory 2(2), pp. 161 – 188, DOI: `10.1016/0022-0531(70)90003-7`.

[18] Sophia Knight, Catuscia Palamidessi, Prakash Panangaden & Frank D. Valencia (2012): *Spatial and Epistemic Modalities in Constraint-Based Process Calculi*. In: CONCUR 2012 - 23rd International Conference on Concurrency Theory, Lecture Notes in Computer Science 7454, Springer, pp. 317–332, DOI: `10.1007/978-3-642-32940-1`. Available at `https://hal.archives-ouvertes.fr/hal-00761116`.

[19] Saul A Kripke (1963): *Semantical analysis of modal logic I normal modal propositional calculi*. Mathematical Logic Quarterly, pp. 67–96, DOI: `10.1002/malq.19630090502`.

[20] Mogens Nielsen, Catuscia Palamidessi & Frank D. Valencia (2002): *Temporal concurrent constraint programming: Denotation, logic and applications*. Nordic Journal of Computing 9(1), pp. 145–188.

[21] Sergio Ramírez, Miguel Romero, Camilo Rocha & Frank Valencia (2018): *Real-Time Rewriting Logic Semantics for Spatial Concurrent Constraint Programming*. In: Rewriting Logic and Its Applications, Lecture Notes in Computer Science 11152, Springer, pp. 226–244, DOI: 10.1007/978-3-319-99840-4_13.

[22] Jean-Hugues Réty (1998): *Distributed Concurrent Constraint Programming*. Fundamenta Informaticae, pp. 323–346.

[23] Camilo Rueda & Frank Valencia (2004): *On validity in modelization of musical problems by CCP*. Soft Computing 8(9), pp. 641–648, DOI: 10.1007/s00500-004-0390-7.

[24] Vijay A. Saraswat, Martin Rinard & Prakash Panangaden (1991): *The Semantic Foundations of Concurrent Constraint Programming*. In: Proceedings of the 18th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '91, ACM, pp. 333–352, DOI: 10.1145/99583.99627.

[25] Jean Serra (1983): *Image Analysis and Mathematical Morphology*. Academic Press, Inc., Orlando, FL, USA.

Distribution of Behaviour into Parallel Communicating Subsystems

Omar al Duhaiby   Jan Friso Groote
Eindhoven University of Technology
Eindhoven, The Netherlands
o.z.alzuhaibi@tue.nl   j.f.groote@tue.nl

The process of decomposing a complex system into simpler subsystems has been of interest to computer scientists over many decades, for instance, for the field of distributed computing. In this paper, motivated by the desire to distribute the process of active automata learning onto multiple subsystems, we study the equivalence between a system and the total behaviour of its decomposition which comprises subsystems with communication between them. We show synchronously- and asynchronously-communicating decompositions that maintain branching bisimilarity, and we prove that there is no decomposition operator that maintains divergence-preserving branching bisimilarity over all LTSs.

1 Introduction

The process of decomposing a complex system into simpler subsystems is the cornerstone of behavioural analysis regardless of where it is applied, to the atom or to the human psyche. Studying the relationship between a complex system and the total behaviour of its decomposition is the subject matter of this paper. However, instead of atoms or human brains, in the field of formal methods, we simply dissect automata. This paper studies how the behaviour of a Labelled Transition System (LTS) can be distributed into a parallel (de)composition of communicating subsystems while maintaining behavioural equivalence.

Motivation This work was motivated by a case study in the industry [4] based on which we pursued the possibility of deducing the internal components of a system based on the model inferred by the active model learning technique [1]. If it were possible at all, then the learned system must be equivalent to the parallel decomposition deduced.

Our goal is to examine the possibility of such deduction and we do that by devising a decompositioning scheme with certain assumptions, then examine its equivalence with the original system.

Related Work Previous work done on the topic of decomposition focuses on uniqueness properties and on producing simpler components. For example the primary decomposition theorem by Krohn and Rhodes states that any automaton can be decomposed into a cascaded product of simpler automata such that the automaton is homomorphic to its decomposition [10]. And in 1998, Milner and Moller introduced a semantics of parallel decompositions comprising non-communicating subsystems [12], and they proved that any finite system of behaviour can be decomposed into a unique set of prime parallel non-communicating subsystems. While Milner and Moller's theorem was in strong bisimulation set in the Calculus of Communicating Systems (CCS), Luttik [11] later introduced a more general framework of commutative monoids that can be used to establish unique decomposition in weak and branching bisimulation semantics.

In contrast to those works, we consider decomposing a system based on partitioning its alphabet into disjoint sets and we require that the subsystems communicate. In that sense, this work is more similar to Brinksma et al. [3] and Hultström [9] who made a decomposition based on a given partition of actions. Another similarity is the need for synchronisation between subsystems as defined in Section 4.1.

Contribution We define two decompositions of parallel communicating subsystems, one synchronous and the other asynchronous, and we prove that both decompositions maintain branching bisimilarity [6] with the source automaton. We also prove that there is no way of decomposing an automaton (under certain conditions) such that it is divergent-preserving branching-bisimilar [5] to the resulting decomposition. We consider branching bisimilarity, but the results of this paper easily carry over to other equivalences such as weak bisimilarity.

Outline The outline of this paper is as follows. Section 2 introduces the preliminaries. Section 3 defines and discussed the general decomposition operator on which we base our arguments. Section 4 defines two decompositions of communicating subsystems, one for synchronous communication and the other for asynchronous communication, and proves that each maintains a branching bisimulation relation with the source automaton. Section 5 contains the proof that there is no way of decomposing an automaton, through our general decomposition operator, such that it maintains divergence preserving branching bisimulation with its decomposition. Finally, Section 6 discusses the results and interprets them in light of our initial motivation.

Acknowledgement We wish to thank Rick Erkens, Bas Luttik, Thomas Neele, Joshua Moerman, Pieter Cuijpers, Bharat Garhewal, Hans van Wezep, Arjan Mooij and the referees of the EXPRESS/SOS 2019 workshop for their thorough feedback, for sharing their knowledge, and for their motivation and support.

2 Preliminaries

In this section, we present the preliminaries of labelled transition systems, the synchronous product and bisimulation relations, aided by [7]. We start with the definition of a labelled transition system (LTS).

Definition 2.1 (LTS)**.** We define our LTS as a four-tuple $(S, \Sigma, \rightarrow, s_0)$ where:

- S is a non-empty finite set of states.
- Σ is the alphabet, also referred to as the action set.
- $\rightarrow \subseteq S \times \Sigma \times S$ is a transition relation.
- s_0 is the initial state.

We use the notation $x \xrightarrow{a} y$ to express a transition with action a from state x to state y. This and variations of it are formally defined as follows.

Definition 2.2 (Transition Relation)**.** Let $(S, \Sigma, \rightarrow, s_0)$ be an LTS with $s, s' \in S$ and $a \in \Sigma \cup \{\tau\}$, where τ is the internal/unobservable action. We use the following notations:

$s \xrightarrow{a} s'$ iff $\langle s,a,s'\rangle \in \rightarrow$.
$s \xrightarrow{a}$ iff there is an s' such that $s \xrightarrow{a} s'$.
$s \not\xrightarrow{a}$ iff there is no s' such that $s \xrightarrow{a} s'$.
$s \xrightarrow{a}^{*} s_n$ iff there are $s_1, s_2, \ldots, s_n \in S$ such that $s \xrightarrow{a} s_1 \xrightarrow{a} s_2 \xrightarrow{a} \cdots \xrightarrow{a} s_n$.
$s \xrightarrow{a}^{\omega}$ iff there are $s_1, s_2, \ldots \in S$ such that $s \xrightarrow{a} s_1$ and for all $i \in \mathbb{N}$, $s_i \xrightarrow{a} s_{i+1}$.

Next, we define complementary actions, i.e., actions on which communicating systems synchronise. Then we define the synchronous product of two automata, and show what role complementary actions play in computing it.

Definition 2.3 (Co-actions)**.** For an arbitrary action a that is not τ, the action $\overline{a}$ (read as *a bar*) is called its co-action. Also, $\overline{(\overline{a})} = a$. We say that actions a and $\overline{a}$ are *complementary* to each other and we call them a pair of *complementary actions*.

We lift this operator to sets of actions such that $\overline{\Sigma} = \{\overline{a} \mid a \in \Sigma\}$.

Definition 2.4 (Synchronous Product)**.** The synchronous product of two LTSs $(S_1, \Sigma_1, \rightarrow_1, q_0) \times (S_2, \Sigma_2, \rightarrow_2, r_0)$ is the tuple $(S_1 \times S_2, \Sigma_x, \rightarrow_x, (q_0, r_0))$ where $\Sigma_x = (\Sigma_1 \cup \Sigma_2) \setminus \{a, \overline{a} \mid a \in \Sigma_1 \wedge \overline{a} \in \Sigma_2\}$.
The transition relation $\rightarrow_x \subseteq (S_1 \times S_2) \times \Sigma_x \times (S_1 \times S_2)$ is defined as follows:

$$\begin{cases} (s,t) \xrightarrow{a} (s',t) & \text{iff } a \in \Sigma_1 \wedge \overline{a} \notin \Sigma_2 \wedge s \xrightarrow{a}_1 s', \\ (s,t) \xrightarrow{a} (s,t') & \text{iff } a \in \Sigma_2 \wedge \overline{a} \notin \Sigma_1 \wedge t \xrightarrow{a}_2 t', \text{ and} \\ (s,t) \xrightarrow{\tau} (s',t') & \text{iff } a \in \Sigma_1 \wedge \overline{a} \in \Sigma_2 \wedge s \xrightarrow{a}_1 s' \wedge t \xrightarrow{\overline{a}}_2 t', \end{cases}$$

where τ is the unobservable action.

Next, we define two notions of behavioural equivalence.

Definition 2.5 (Branching bisimulation)**.** Given an LTS $(S, \Sigma, \rightarrow, s_0)$ and a relation $\mathscr{R} \subseteq S \times S$. We call $\mathscr{R}$ a branching bisimulation relation iff for all states $s,t \in S$ such that $\langle s,t\rangle \in \mathscr{R}$, it holds that:

1. if $s \xrightarrow{a} s'$, then:
   - $a = \tau$ and $\langle s',t\rangle \in \mathscr{R}$; or
   - $t \xrightarrow{\tau}^{*} t' \xrightarrow{a} t''$, $\langle s,t'\rangle \in \mathscr{R}$ and $\langle s',t''\rangle \in \mathscr{R}$.
2. Symmetrically, if $t \xrightarrow{a} t'$, then:
   - $a = \tau$ and $\langle s,t'\rangle \in \mathscr{R}$; or
   - $s \xrightarrow{\tau}^{*} s' \xrightarrow{a} s''$, $\langle s',t\rangle \in \mathscr{R}$ and $\langle s'',t'\rangle \in \mathscr{R}$.

Two states s and t are branching *bisimilar*, denoted $s \leftrightarroweq_b t$ iff there is a branching bisimulation relation $\mathscr{R}$ such that $\langle s,t\rangle \in \mathscr{R}$. Two LTSs P and Q are branching bisimilar, denoted $P \leftrightarroweq_b Q$, iff their initial states are.

A state s with $s \xrightarrow{\tau}^{\omega}$ is called *divergent*. Hence, a state with a τ loop is also called divergent. Branching bisimulation does not preserve divergence, i.e., a divergent state can be branching bisimilar to a non-divergent one. Therefore, a stronger equivalence relation, namely divergence-preserving branching bisimulation, is defined below.

Definition 2.6 (Divergence-preserving branching bisimulation)**.** Given an LTS $(S, \Sigma, \rightarrow, s_0)$ and a relation $\mathscr{R} \subseteq S \times S$. We call $\mathscr{R}$ a divergence-preserving branching bisimulation relation iff it is a branching bisimulation relation and for all states $s,t \in S$ with $\langle s,t\rangle \in \mathscr{R}$, there is an infinite sequence $s \xrightarrow{\tau} s_1 \xrightarrow{\tau} s_2 \xrightarrow{\tau}$ with $\langle s_i,t\rangle \in \mathscr{R}$ for all $i > 0$ iff there is an infinite sequence $t \xrightarrow{\tau} t_1 \xrightarrow{\tau} t_2 \xrightarrow{\tau}$ and $\langle s,t_i\rangle \in \mathscr{R}$ for all $i > 0$.

Two states s and t are divergence-preserving branching bisimilar, denoted $s \leftrightarroweq_{db} t$ iff there is a divergence-preserving branching bisimulation relation $\mathscr{R}$ such that $\langle s,t\rangle \in \mathscr{R}$. Two LTSs P and Q are divergence-preserving branching bisimilar, denoted $P \leftrightarroweq_{db} Q$, iff their initial states are.

3 The Decomposition Operation

In this section, we establish a decompositioning scheme that is based on action partitioning in order to refer to it as *the* general decomposition operation on which our theorems apply. Our general decomposition operation is a function transforming a single LTS, given two disjoint actions sets, into two LTSs. It is formally defined as follows.

Definition 3.1 (General Decomposition Operation)**.** Given an LTS M with alphabet Σ and given two alphabets Σ_1, Σ_2 such that $\Sigma = \Sigma_1 \cup \Sigma_2$ and $\Sigma_1 \cap \Sigma_2 = \emptyset$, we call G a *general decomposition operator* iff $G(M, \Sigma_1, \Sigma_2) = (M_1, M_2)$ such that M_1 has alphabet Σ_{M_1} with $\Sigma_1 \subseteq \Sigma_{M_1}$ and $\Sigma_{M_1} \cap \Sigma_2 = \emptyset$, and likewise, M_2 has alphabet Σ_{M_2} with $\Sigma_2 \subseteq \Sigma_{M_2}$ and $\Sigma_{M_2} \cap \Sigma_1 = \emptyset$.

We refer to a method of decomposing automata as a *decomposition operation* whereas the result of such transformation is called a *decomposition*. A decomposition comprises two or more automata. This transformation is depicted in Figure 2. Throughout the paper, we compare LTSs to the synchronous product of the decomposition, and if a a certain bisimulation relation holds between these two, then we say that the operation *maintains* that relation.

The idea behind this transformation is, given a partition of actions of a system, to generate two subsystems, each using exclusively one of the two parts.

Recursive decomposition. Note that Definition 3.1 can easily be applied recursively allowing to decompose behaviour into multiple components. As the alphabets over which an automaton can be decomposed can be empty, the number of components over which behaviour can be split can even be arbitrarily large.

4 Branching Bisimilar Decompositions

In this section, we define two decomposition operations that are designed to maintain branching bisimilarity, and we actually prove that they do. The first one ($decomp_s$) decomposes into synchronously communicating subsystems while the second ($decomp_a$) decomposes into asynchronously communicating ones. In both of these operations, we build two automata that pass control between one another using c messages. Only if an automaton seizes control can it perform one of its actions, otherwise it has to wait for the other to hand control over. Formal definitions and more detail follow.

4.1 Decomposing into Synchronous Subsystems

We define the decomposition of synchronous subsystems, summarised in Figure 1 in two patterns; the top dictates the decomposition of every state in the source LTS while the bottom dictates the decomposition of every transition. An omitted third pattern is symmetric to the second such that the transition's label simply belongs to the second subsystem rather than the first.

Definition 4.1 (Synchronous Decomposition Operation)**.** Given an LTS $M = (S, \Sigma, \rightarrow, q)$ and two alphabets Σ_1, Σ_2 such that $\Sigma = \Sigma_1 \cup \Sigma_2$ and $\Sigma_1 \cap \Sigma_2 = \emptyset$, then we can decompose M over Σ_1 and Σ_2 by applying the following operation:

$decomp_s(M, \Sigma_1, \Sigma_2) = (M_1, M_2)$ where:

1. $M_1 = (S_C \cup S_{T_1}, \Sigma_1 \cup \Sigma_{S_1}, \rightarrow_1, (q, 1))$
   $M_2 = (S_C \cup S_{T_2}, \Sigma_2 \cup \Sigma_{S_2}, \rightarrow_2, (q, 1))$.

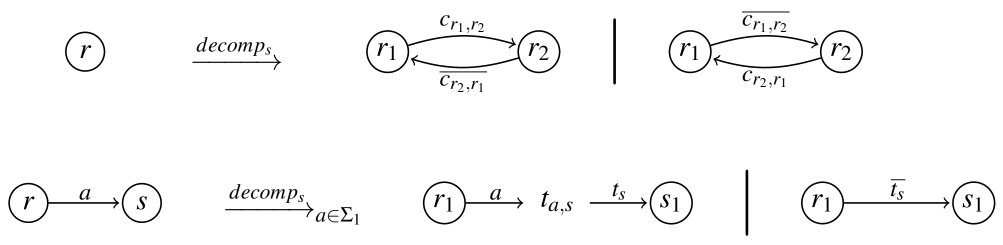


Figure 1: The two patterns that delineate the operator *decomp*$_s$ (Definition 4.1).

2. For every state s in S, we introduce two states $(s,1),(s,2) \in S_C$:

$$S_{C_1} = \{(s,1) \mid s \in S\} \qquad S_{C_2} = \{(s,2) \mid s \in S\} \qquad S_C = S_{C_1} \cup S_{C_2} \tag{1}$$

Notation. Tuple-states of the form (s,i) such as $(s,1)$ and $(s,2)$ are shortened to s_i. Therefore, it is to be held throughout the paper that s_i is derived from s rather than it being a completely unrelated symbol to s.

3. The set of c-actions is defined as follows:

$$\Sigma_C = \{c_{s_1,s_2}, \overline{c_{s_2,s_1}} \mid s_1 \in S_{C_1}, s_2 \in S_{C_2}\} \tag{2}$$

4. The sets of t actions and t states are defined as follows:

$$\begin{array}{ll} \Sigma_{T_1} = \{t_{s_1} \mid s_1 \in S_C\} & \Sigma_{T_2} = \{t_{s_2} \mid s_2 \in S_C\} \\ S_{T_1} = \{t_{a,s_1} \mid a \in \Sigma_1, s_1 \in S_{C_1}\} & S_{T_2} = \{t_{a,s_2} \mid a \in \Sigma_2, s_2 \in S_{C_2}\} \end{array} \tag{3}$$

5. The complete sets of actions of M_1 and M_2 are respectively defined as:

$$\Sigma_{S_1} = \Sigma_{T_1} \cup \Sigma_C \cup \overline{\Sigma_{T_2}} \qquad \Sigma_{S_2} = \Sigma_{T_2} \cup \overline{\Sigma_C} \cup \overline{\Sigma_{T_1}} \tag{4}$$

6. The transition relations $\rightarrow_i \subseteq (S_C \cup S_{T_i}) \times (\Sigma_i \cup \Sigma_{S_i}) \times (S_C \cup S_{T_i})$ are defined as follows. For $i,j \in \{1,2\}$ and $i \neq j$, $\rightarrow_i$ is the minimal relation satisfying the following:

   (a) For all $s \in S$ and for all $c_{s_i,s_j} \in \Sigma_C$:

$$s_i \xrightarrow{c_{s_i,s_j}}_i s_j \qquad\qquad s_i \xrightarrow{\overline{c_{s_i,s_j}}}_j s_j \tag{5}$$

   (b) For all $s,s' \in S$, and all $a \in \Sigma_i$, if $s \xrightarrow{a} s'$, then:

$$\begin{array}{r} s_i \xrightarrow{a}_i t_{a,s'_i} \xrightarrow{t_{s'_i}}_i s'_i \\ s_i \xrightarrow{\overline{t_{s'_i}}}_j s'_i \end{array} \tag{6}$$

Two classes of actions are introduced, c-actions and t-actions. The c-actions come in pairs, and they resemble passing a control token between M_1 and M_2. For instance, looking at Figure 2, when, at some state $r \in S$ for which a pair of states $r_1, r_2 \in S_C$ exists in both M_1 and M_2, and control is to be passed from M_1 to M_2, then a pair of complementary c actions synchronises, namely, actions c_{r_1,r_2} and $\overline{c_{r_1,r_2}}$, to produce a synchronous transition in both machines from r_1 to r_2. Likewise, actions c_{r_2,r_1} and $\overline{c_{r_2,r_1}}$ synchronise to pass control in the opposite direction from M_2 to M_1.

The t-actions are introduced to synchronise transitions occurring in one machine with the other. In addition, they require the introduction of t-states. Observe Figure 2 where an a_1 transition occurs in M_1. The aim is the transition $r_1 \xrightarrow{a_1} s_1$, but in order to synchronise this with M_2, we introduce a middle state $t_{a_1,s_1} \in S_{T_1}$ from which the only possible transition is $t_{a_1,s_1} \xrightarrow{t_{s_1}} s_1$ which synchronises with the transition $r_1 \xrightarrow{\overline{t_{s_1}}} s_1$ in M_2.

The operation ($decomp_s$) can be summarised by two patterns shown in Figure 1; the top pattern applies to each state and the bottom one applies to each transition.

Is the Decomposition a Simplification? The decomposition is obviously larger than the original system. That is due to the nature of an alphabet-partitioning-based decomposition where subsystems must hand control over between one another. Thus, every subsystem must have, for each state in the original, multiple ones expressing where control lies. In special cases, a smaller component may suffice; for example, in Figure 2, because state r enables no b actions, a state r_2 is not needed and all transitions going to r_2 can instead go to r_1. However, we believe that our definition the way it is is more understandable because of its generality and symmetry. The element of simplification lies not in reducing the size of a system but rather in partitioning its alphabet.

Computing the Synchronous Product. For a decomposition (M_1, M_2) by Definition 4.1, the synchronous product $M_x = M_1 \times M_2$ is the LTS $(S_x, \Sigma_1 \cup \Sigma_2, \rightarrow_x, (q_1, q_1))$, where:

$$\begin{aligned} S_x = S_1 \times S_2 &= (S_C \cup S_{T_1}) \times (S_C \cup S_{T_2}) \\ &= (S_C \times S_C) \cup (S_{T_1} \times S_{T_2}) \\ &\quad \cup (S_{T_1} \times S_C) \cup (S_C \times S_{T_2}) \end{aligned} \tag{7}$$

with $\Sigma_{S_1}, \Sigma_{S_2}, S_{T_1}, S_{T_2}$ being sets introduced by $decomp_s$. The transition relation $\rightarrow_x$ is defined as follows for $i, j \in \{1,2\}$ and $i \neq j$:

1. if $s \xrightarrow{a} s'$ and $a \in \Sigma_i$ then by (6) there is a state $t_{a,s'_i} \in S_{T_i}$ and a pair of complementary actions $t_{s'_i}, \overline{t_{s'_i}} \in \Sigma_{S_i}$ such that:

$$(s_i, s_i) \xrightarrow{a}_x s_a \xrightarrow{\tau}_x (s'_i, s'_i), \tag{8}$$

where $s_a = \begin{cases} (t_{a,s'_i}, s_i) & \text{if } i = 1, \\ (s_i, t_{a,s'_i}) & \text{if } i = 2. \end{cases}$

2. For all $s \in S$, there exist $c_{s_i,s_j}, c_{s_j,s_i} \in \Sigma_C$ such that, by (5), $s_i \xrightarrow{c_{s_i,s_j}}_i s_j$ and $s_i \xrightarrow{\overline{c_{s_i,s_j}}}_j s_j$, and thus:

$$(s_i, s_i) \xrightarrow{\tau}_x (s_j, s_j) \tag{9}$$

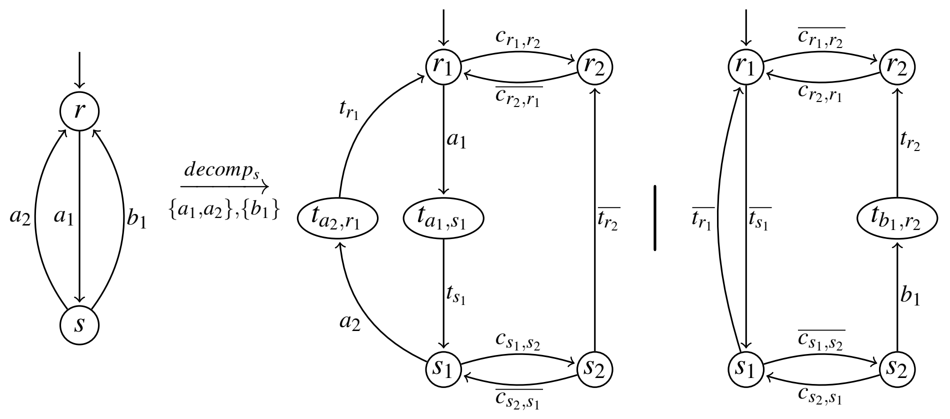


Figure 2: Example of synchronous decomposition operation of Definition 4.1.

4.2 Proof that the synchronous decomposition operation maintains branching bisimulation

In this subsection, we show an application of *decomp*$_s$ (Definition 4.1) to a sample LTS, we demonstrate that *decomp*$_s$ maintains branching bisimilarity, and then we prove that branching bisimilarity is maintained through any and all applications of *decomp*$_s$.

Figure 2 shows the LTS at the left side and its decomposition at the right side. The two patterns shown in Figure 1 can be applied directly to this LTS. The top pattern applies twice, once per state, and the bottom pattern applies three times, once per transition.

Next, we compute the synchronous product and form one LTS shown at the right of Figure 3. The nodes are divided into two equivalence classes, top and bottom. The states in the top class are branching bisimilar to state r whereas the states in the bottom one are branching bisimilar to state s.

The following proves the branching bisimilarity and thus proves that there is a way of decomposing an LTS such that branching bisimilarity is maintained.

Theorem 4.2. Given an LTS $M = (S,\Sigma,\rightarrow,s_0)$ and two alphabets Σ_1,Σ_2 such that $\Sigma = \Sigma_1 \cup \Sigma_2$ and $\Sigma_1 \cap \Sigma_2 = \emptyset$, and given an LTS $M_x = M_1 \times M_2$ where $(M_1,M_2) = decomp_s(M)$ by Definition 4.1, then $M \leftrightarrow_b M_x$.

Proof. Let $M_1 = (S_C \cup S_{T_1},\Sigma_1 \cup \Sigma_{S_1},\rightarrow_1,q_1)$ and $M_2 = (S_C \cup S_{T_2},\Sigma_2 \cup \Sigma_{S_2},\rightarrow_2,q_2)$.

Define a relation $\mathscr{R} \subseteq S \times ((S_C \cup S_{T_1}) \times (S_C \cup S_{T_2}))$ with $\mathscr{R} = \{\langle s,(s_n,s_n)\rangle, \langle r',(t_{a,r'_n},r_n)\rangle, \langle r',(r_n,t_{a,r'_n})\rangle \mid s,r,r' \in S, n \in \{1,2\}, a \in \Sigma, r \xrightarrow{a} r'\}$. We prove that $\mathscr{R}$ is a branching bisimulation relation through the following cases:

1. Consider a pair $\langle s,s_x\rangle = \langle s,(s_n,s_n)\rangle$ where $n \in \{1,2\}$.
   (a) Assume $s \xrightarrow{a} s'$. Then we have two cases:
      i. $a \in \Sigma_1$. Then, by (8), $s_x \xrightarrow{a}_x (t_{a,s'_n},s_n)$. We see that $\langle s',(t_{a,s'_n},s_n)\rangle \in \mathscr{R}$.
      ii. $a \in \Sigma_2$. Then, by (8), $s_x \xrightarrow{a}_x (s_n,t_{a,s'_n})$. We see that $\langle s',(s_n,t_{a,s'_n})\rangle \in \mathscr{R}$.
   (b) Assume $s_x \xrightarrow{a}_x s'_x$. Then we have the following three cases:
      i. $a \in \Sigma_1 \wedge \overline{a} \notin \Sigma_2$, then this transition is only possible, by definition, through the transition $s \xrightarrow{a} s'$ for some s' such that $s'_x \overset{(8)}{=} (t_{a,s'},s_1)$. We see that $\langle s',s'_x\rangle \in \mathscr{R}$.

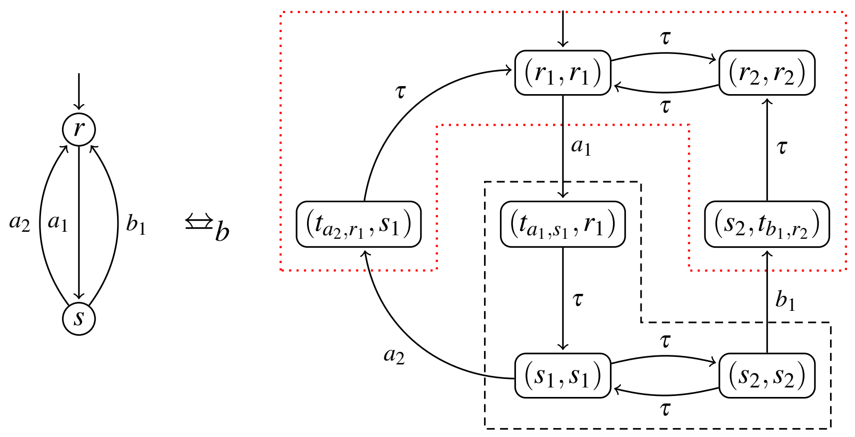


Figure 3: Showing branching bisimulation on the example of Figure 2.

ii. $a \in \Sigma_2 \wedge \overline{a} \notin \Sigma_1$. This is a symmetric case where $s \xrightarrow{a} s'$ and $s'_x \overset{(8)}{=} (s_2, t_{a,s'})$. We see that $\langle s', s'_x \rangle \in \mathscr{R}$.

iii. $a \in \Sigma_1 \wedge \overline{a} \in \Sigma_2$, then the only transition possible is the τ transition of (9). Then $s'_x = (s_m, s_m)$ where $m \in \{1,2\}$ and $m \neq n$. We see that $\langle s, s'_x \rangle \in \mathscr{R}$.

2. Consider a pair $\langle r, r_x \rangle = \langle s', (t_{a,s'_n}, s_n) \rangle$ where $n \in \{1,2\}$, $a \in \Sigma$ and $s \xrightarrow{a} s'$.

(a) Assume $r \xrightarrow{a} r'$. Then we show that $r_x \xrightarrow{\tau}_x r'_x$ and $r'_x \xrightarrow{a}_x r''_x$ and $\langle r, r'_x \rangle \in \mathscr{R}$ and $\langle r', r''_x \rangle \in \mathscr{R}$. We do this for $a \in \Sigma_1$. The case for $a \in \Sigma_2$ is symmetric.

i. $r'_x \overset{(8)}{=} (s'_2, s'_2)$. We see that $\langle r, r'_x \rangle \in \mathscr{R}$.

ii. Since $r = s'$ and $r \xrightarrow{a} r'$, then by (8), there exists a state r''_x such that $r'_x \xrightarrow{a}_x r''_x$, and $r''_x = (t_{a,s''_2}, s'_2)$, where $s'' = r'$. We see that $\langle r', r''_x \rangle \in \mathscr{R}$.

(b) Assume $r_x \xrightarrow{a}_x r'_x$. By the definition of $\rightarrow_x$, it is only possible that a is a τ action and that $n = 1$. Thus, $r'_x \overset{(8)}{=} (s'_2, s'_2)$. We see that $\langle r', r''_x \rangle \in \mathscr{R}$.

3. Consider a pair $\langle r, r_x \rangle = \langle s', (s_n, t_{a,s'_n}) \rangle$ where $n \in \{1,2\}$, $a \in \Sigma$ and $s \xrightarrow{a} s'$. This case is symmetric to Case 2.

□

Corollary 4.3. It follows from Theorem 4.2 that there is a universal way of decomposing an LTS M using a general synchronous decomposition operator (Definition 3.1) such that M is branching-bisimilar to the synchronous product of its decomposition.

4.3 Decomposing into Asynchronous Subsystems

We consider asynchronous communication between subsystems simply because many practical systems use asynchronous communication and the aim is to extend our proof towards that. So, we define a new decomposition operation ($decomp_a$) such that the communication between subsystems is asynchronous. We assign each subsystem a queue that stores received messages until they are consumed. An action of sending such a message does synchronise, however, with the queue of the opposite side receiving it. The operation $decomp_a$ is summarised in Figure 4.

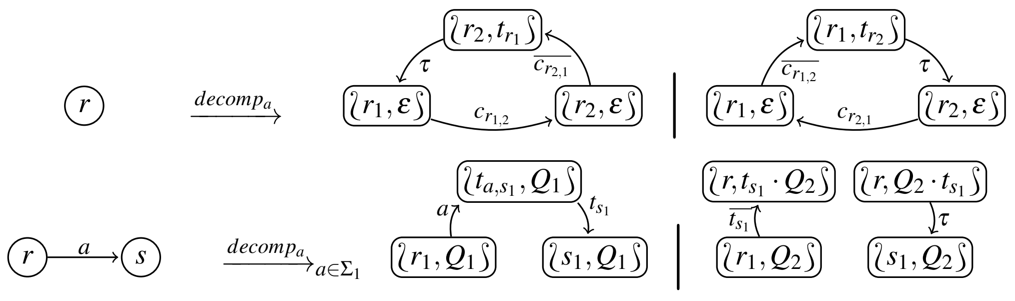


Figure 4: The two patterns that delineate the $decomp_a$ operator (Definition 4.5).

Definition 4.4 (LTS with Queue)**.** A queue is an ordered-list of actions. An LTS with a queue is a transition system of the shape $(S \times Q, \Sigma, \rightarrow, s_0)$. A state in $S \times Q$ holds the contents of the queue Q and is written as $\wr s, Q \wr$.

Elements in a queue are concatenated using the $\cdot$ operator. Appending an element m to the back of a queue Q produces the queue $m \cdot Q$, while $Q \cdot m$ represents the queue with m in the front. The symbol ε represents the empty queue.

Definition 4.5 (Decomposing into asynchronous subsystems)**.** Given an LTS $M = (S, \Sigma, \rightarrow, r_0)$ and two alphabets Σ_1, Σ_2 such that $\Sigma = \Sigma_1 \cup \Sigma_2$ and $\Sigma_1 \cap \Sigma_2 = \emptyset$, then we can decompose M over Σ_1 and Σ_2 by applying the following operation:

$decomp_a(M, \Sigma_1, \Sigma_2) = (M_1, M_2)$ where, for $i, j \in \{1, 2\}$ and $i \neq j$, M_i is an LTS with a queue (Definition 4.4) defined as follows:

1. $M_i = ((S_C \cup S_{T_i}, Q_i), \Sigma_i \cup \Sigma_{S_i}, \rightarrow_i, r_1)$
2. For every state in S, we introduce a pair of states $s_1, s_2 \in S_C$, a pair of c-actions, and a pair of t-actions:

$$\begin{aligned} S_{C_i} &= \{s_i \mid s \in S\} & S_C &= S_{C_1} \cup S_{C_2} \\ \Sigma_{C_{i,j}} &= \{c_{s_{i,j}} \mid s \in S\} & \Sigma_{T_i} &= \{t_{s_i} \mid s \in S\} \end{aligned} \tag{10}$$

3. Sets of t-states are defined as follows:

$$S_{T_i} = \{t_{a,s_i} \mid a \in \Sigma_i, s_i \in S_{C_i}\} \tag{11}$$

4. Sets of synchronous actions are defined as follows:

$$\Sigma_{S_i} = \Sigma_{T_i} \cup \overline{\Sigma_{T_j}} \cup \Sigma_{C_{i,j}} \cup \overline{\Sigma_{C_{j,i}}} \tag{12}$$

5. The transition relation $\rightarrow_i \subseteq (S_C \cup S_{T_i}) \times Q_i \times (\Sigma_i \cup \Sigma_{S_i}) \times (S_C \cup S_{T_i}) \times Q_i$ is the minimal relation satisfying the following:
   (a) For all $s \in S$:

$$\wr s_i, Q_i \wr \xrightarrow{c_{s_{i,j}}}_i \wr s_j, Q_i \wr \qquad \wr s_i, Q_i \wr \xrightarrow{\overline{c_{s_{i,j}}}}_j \wr s_i, t_{s_j} \cdot Q_i \wr \tag{13}$$

(b) For all $s, s' \in S$, and all $a \in \Sigma_i$, if $s \xrightarrow{a} s'$, then:

$$\begin{aligned} \wr s_i, Q_i \wr \xrightarrow{a}_i \wr t_{a,s'_i}, Q_i \wr \xrightarrow{t_{s'_i}}_i \wr s'_i, Q_i \wr \\ \wr s_i, Q_i \wr \xrightarrow{\overline{t_{s'_i}}}_j \wr s_i, t_{s'_i} \cdot Q_i \wr \end{aligned} \tag{14}$$

(c) Consuming an element from the front of a queue is an internal transition of the form:

$$\wr s, Q \cdot t_{s'} \wr \xrightarrow{\tau} \wr s', Q \wr \tag{15}$$

We see in (13) that the two automata synchronise on action $c_{s_{i,j}}$. The effect is a message sent from M_i and received in the queue of M_j. The same occurs in (14). Moreover, this makes sending messages only possible when both machines are in sync, i.e., on the same state s_i. This model is inspired by asynchronous communication in practice and we found it necessary that a sent message is received before any other actions are performed by either side.

The question may arise “why do we still see synchronisation? Is this still considered asynchronous communication?”; and the answer is that it is asynchronous in the sense that the sending party does not know whether and when the receiving party is willing to receive the communication. This is different from synchronous communication where the sender knows that the receiver is willing to participate.

For this construction, a queue of size 1 is enough as it never contains more than one message. If the size of the queue was more than 1, then the transition in (14) can apply recursively until the queue is full. However, any transition beyond the first one does not translate into a transition in the product because action $c_{s_{i,j}}$ needs to synchronise with its co-action. In other words, in the product automaton, the queue cannot contain more than one message. Therefore, the upperbound of size 1 of the queue is not a requirement, but rather follows from the construction.

4.4 Proof that the Asynchronous Decomposition Operation Maintains Branching Bisimulation

In this subsection, similar to Section 4.2, we prove that the asynchronous decomposition operation ($decomp_a$) also maintains branching bisimilarity. Figure 5 shows the result of applying $decomp_a$ to the same example behaviour as in Figure 2. In Figure 6, we compute the synchronous product of the decomposition of Figure 5 and then divide the nodes of the product into two equivalence classes, top and bottom. The states in the top class are branching bisimilar to state r whereas the states in the bottom on are branching bisimilar to state s.

Next, we prove that any LTS decomposed using Definition 4.5 maintains branching bisimulation with its decomposition, thus by proving that there is at least one universal method of decomposing LTSs into asynchronous ones while maintaining branching bisimulation.

Theorem 4.6. Given an LTS $M = (S, \Sigma, \rightarrow, s_0)$ and two alphabets Σ_1, Σ_2 such that $\Sigma = \Sigma_1 \cup \Sigma_2$ and $\Sigma_1 \cap \Sigma_2 = \emptyset$, and given an LTS $M_x = M_1 \times M_2$ where $(M_1, M_2) = decomp_a(M)$ by Definition 4.5, then $M \leftrightarrows_b M_x$.

Proof. Let $M_1 = ((S_C \cup S_{T_1}, Q_1), \Sigma_1 \cup \Sigma_{S_1}, \rightarrow_1, r_1)$ and $M_2 = ((S_C \cup S_{T_2}, Q_2), \Sigma_2 \cup \Sigma_{S_2}, \rightarrow_2, r_1)$.

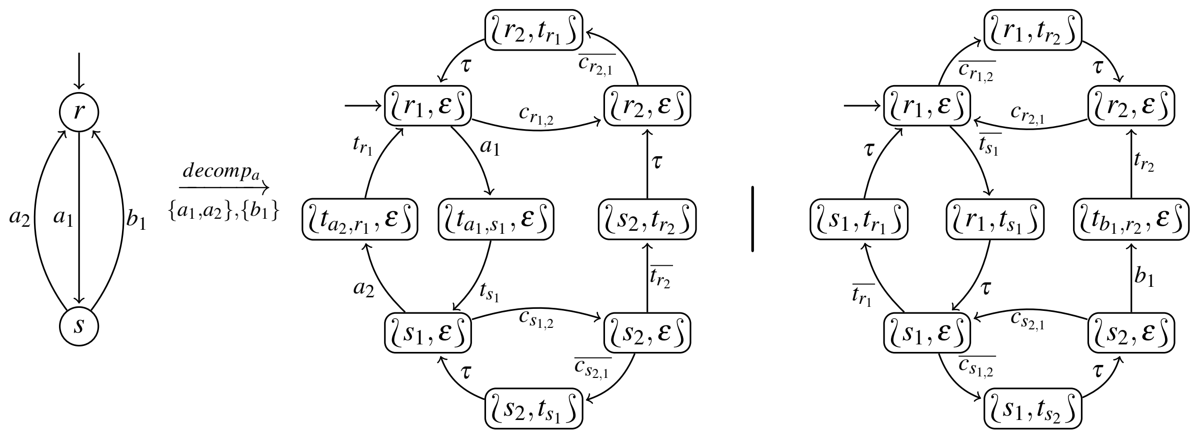

r
s
a_2
a_1
b_1
$\xrightarrow[\{a_1,a_2\},\{b_1\}]{decomp_a}$
$\lbag r_2, t_{r_1} \rbag$
$\lbag r_1, \varepsilon \rbag$
$\lbag r_2, \varepsilon \rbag$
$\lbag t_{a_2,r_1}, \varepsilon \rbag$
$\lbag t_{a_1,s_1}, \varepsilon \rbag$
$\lbag s_2, t_{r_2} \rbag$
$\lbag s_1, \varepsilon \rbag$
$\lbag s_2, \varepsilon \rbag$
$\lbag s_2, t_{s_1} \rbag$
τ
$\overline{c_{r_{2,1}}}$
$c_{r_{1,2}}$
t_{r_1}
$\overline{t_{r_2}}$
t_{s_1}
$c_{s_{1,2}}$
$\overline{c_{s_{2,1}}}$
$\lbag r_1, t_{r_2} \rbag$
$\lbag s_1, t_{r_1} \rbag$
$\lbag r_1, t_{s_1} \rbag$
$\lbag t_{b_1,r_2}, \varepsilon \rbag$
$\lbag s_1, t_{s_2} \rbag$
$\overline{c_{r_{1,2}}}$
$c_{r_{2,1}}$
$\overline{t_{s_1}}$
t_{r_2}
$\overline{t_{r_1}}$
b_1
$c_{s_{2,1}}$
$\overline{c_{s_{1,2}}}$

Figure 5: Example of asynchronous decomposition operation (Definition 4.5).

Define a relation $\mathscr{R} \subseteq S \times ((S_C \cup S_{T_1}, Q_1) \times (S_C \cup S_{T_2}, Q_2))$ with $\mathscr{R} =$

$$\begin{aligned}
&\{\langle s, (\lbag s_i, \varepsilon \rbag, \lbag s_i, \varepsilon \rbag)\rangle,\\
&\langle s, (\lbag s_i, t_{s_j} \rbag, \lbag s_j, \varepsilon \rbag)\rangle, \langle s, (\lbag s_j, \varepsilon \rbag, \lbag s_i, t_{s_j} \rbag)\rangle,\\
&\langle s, (t_{a,s_i}, \lbag r_i, \varepsilon \rbag)\rangle, \langle s, (\lbag r_i, \varepsilon \rbag, t_{a,s_i})\rangle,\\
&\langle s, (\lbag s_i, \varepsilon \rbag, \lbag r_i, t_{s_i} \rbag)\rangle, \langle s, (\lbag r_i, t_{s_i} \rbag, \lbag s_i, \varepsilon \rbag)\rangle,\\
&\langle u, (t_{b,s'_i}, \lbag r_i, t_{s_i} \rbag)\rangle, \langle u, (\lbag r_i, t_{s_i} \rbag, t_{b,s'_i})\rangle \mid\\
&r, s, u \in S \text{ and } i, j \in \{1,2\} \text{ where } i \neq j \text{ and } a, b \in \Sigma_i \text{ and } r \xrightarrow{a} s \xrightarrow{b} u\}.
\end{aligned}$$

We prove that $\mathscr{R}$ is a branching bisimulation relation through the following cases:

1. Consider a pair $\langle s, s_x \rangle = \langle s, (\lbag s_i, \varepsilon \rbag, \lbag s_i, \varepsilon \rbag)\rangle$ where $i \in \{1,2\}$.
   (a) Assume $s \xrightarrow{a} s'$. Then if $a \in \Sigma_1$, then $s_x \xrightarrow{a}_1 s'_x$ where $s'_x \overset{(14)}{=} (t_{a,s'_i}, \lbag s_i, \varepsilon \rbag)$ with $i = 1$. Else if $a \in \Sigma_2$ then $s_x \xrightarrow{a}_2 s''_x$ where $s''_x \overset{(14)}{=} (\lbag s_i, \varepsilon \rbag, t_{a,s'_i})$ with $i = 2$. We see that both pairs $\langle s, s'_x \rangle$ and $\langle s, s''_x \rangle$ are in $\mathscr{R}$.
   (b) Assume $s_x \xrightarrow{a}_x s'_x$. Then we have the following three cases:
      i. $a \in \Sigma_1 \wedge \overline{a} \notin \Sigma_2$, then this transition is only possible, by definition, through the transition $s \xrightarrow{a} s'$ for some s' such that $s'_x \overset{(14)}{=} (t_{a,s'}, s_1)$. We see that the pair $\langle s', s'_x \rangle \in \mathscr{R}$ and is covered in case 4.
      ii. $a \in \Sigma_2 \wedge \overline{a} \notin \Sigma_1$. This is a symmetric case where $s \xrightarrow{a} s'$ and $s'_x \overset{(14)}{=} (s_2, t_{a,s'})$. We see that the pair $\langle s', s'_x \rangle \in \mathscr{R}$ and is covered in case 5.
      iii. $a \in \Sigma_1 \wedge \overline{a} \in \Sigma_2$, then the only transition possible is the τ transition of (13). Then either $s'_x = (\lbag s_j, \varepsilon \rbag, \lbag s_i, t_{s_j} \rbag)$ or $s'_x = (\lbag s_i, t_{s_j} \rbag, \lbag s_j, \varepsilon \rbag)$ where $j \in \{1,2\}$ and $j \neq i$. We see that in both possible values of s'_x, the pair $\langle s, s'_x \rangle \in \mathscr{R}$ and is covered in cases 2 and 3.
2. Consider a pair $\langle s, s_x \rangle = \langle s, (\lbag s_i, t_{s_j} \rbag, \lbag s_j, \varepsilon \rbag)\rangle$.
   (a) Assume $s \xrightarrow{a} s'$. Then $s_x \xrightarrow{\tau}_x (\lbag s_j, \varepsilon \rbag, \lbag s_j, \varepsilon \rbag)$, and we covered the pair $\langle s, (\lbag s_j, \varepsilon \rbag, \lbag s_j, \varepsilon \rbag)\rangle$ in case 1.
   (b) Assume $s_x \xrightarrow{a}_x s'_x$. The only possible transition in $\rightarrow_x$ is if a is a τ action consuming the queue message t_{s_j} then $s'_x = (\lbag s_j, \varepsilon \rbag, \lbag s_j, \varepsilon \rbag)$ and we covered the pair $\langle s, (\lbag s_j, \varepsilon \rbag, \lbag s_j, \varepsilon \rbag)\rangle$ in case 1.

3. Consider a pair $\langle s, s_x\rangle = \langle s, (\wr s_j, \varepsilon \wr, \wr s_i, t_{s_j} \wr)\rangle$. Symmetric to case 2.
4. Consider a pair $\langle s, s_x\rangle = \langle s, (t_{a,s_i}, \wr r_i, \varepsilon \wr)\rangle$ such that $r \xrightarrow{a} s$.
   (a) Assume $s \xrightarrow{b} s'$. Then $s_x \xrightarrow{\tau}_x (\wr s_i, \varepsilon \wr, \wr r_i, t_{s_i} \wr)$.
   (b) Assume $s_x \xrightarrow{b}_x s'_x$. The only possible transition in $\rightarrow_x$ is if b is a τ action resulting from the synchronisation of the two transitions $t_{a,s_i} \xrightarrow{t_{s_i}}_i \wr s_i, \varepsilon \wr$ and $\wr r_i, \varepsilon \wr \xrightarrow{\overline{t_{s_i}}}_j \wr r_i, t_{s_i} \wr$. Then, in the product, $s_x \xrightarrow{\tau}_x (\wr s_i, \varepsilon \wr, \wr r_i, t_{s_i} \wr)$; and the pair $\langle s, (\wr s_i, \varepsilon \wr, \wr r_i, t_{s_i} \wr)\rangle \in \mathscr{R}$ and is covered in case 6.
5. Consider a pair $\langle s, s_x\rangle = \langle s, (\wr r_i, \varepsilon \wr, t_{a,s_i})\rangle$. Symmetric to case Case 4.
6. Consider a pair $\langle s, s_x\rangle = \langle s, (\wr s_i, \varepsilon \wr, \wr r_i, t_{s_i} \wr)\rangle$ such that $r \xrightarrow{a} s$.
   (a) Assume $s \xrightarrow{b} s'$. Then because of the queue-consuming transition $\wr r_i, t_{s_i} \wr \xrightarrow{\tau}_j \wr s_i, \varepsilon \wr$, then $s_x \xrightarrow{\tau}_x (\wr s_i, \varepsilon \wr, \wr s_i, \varepsilon \wr)$.
   The pair $\langle s, (\wr s_i, \varepsilon \wr, \wr s_i, \varepsilon \wr)\rangle$ is covered in case 1.
   (b) Assume $s_x \xrightarrow{b}_x s'_x$, then there are two possible values for b:
      i. Action b is a queue-consuming τ, then $s_x \xrightarrow{\tau}_x (\wr s_i, \varepsilon \wr, \wr s_i, \varepsilon \wr)$; and the pair $\langle s, (\wr s_i, \varepsilon \wr, \wr s_i, \varepsilon \wr)\rangle$ is covered in case 1.
      ii. $b \in \Sigma_i$, then $s_x \xrightarrow{b}_x (t_{b,s'_i}, \wr r_i, t_{s_i} \wr)$ such that $s \xrightarrow{b} s'$; and the pair $\langle s', (t_{b,s'_i}, \wr r_i, t_{s_i} \wr)\rangle \in \mathscr{R}$.
7. Consider a pair $\langle s, s_x\rangle = \langle s, \wr r_i, t_{s_i} \wr, \wr s_i, \varepsilon \wr)\rangle$. Symmetric to case 6.
8. Consider a pair $\langle s, s_x\rangle = \langle s, (t_{a,s_i}, \wr \wr p_i, \varepsilon \wr, t_{r_i} \wr)\rangle$ such that $p \xrightarrow{b} r \xrightarrow{a} s$. Then $s_x \xrightarrow{\tau} (t_{a,s_i}, \wr r_i, \varepsilon \wr)$; and the pair $\langle s, (t_{a,s_i}, \wr r_i, \varepsilon \wr)\rangle$ is covered in case 4.
9. Consider a pair $\langle s, s_x\rangle = \langle s, (\wr p_i, t_{r_i} \wr, t_{a,s_i})\rangle$ such that $p \xrightarrow{b} r \xrightarrow{a} s$. This is symmetric to case 8.

□

Corollary 4.7. It follows from Theorem 4.6 that there is a universal way of decomposing an LTS M using a general asynchronous decomposition operator (Definition 3.1) such that M is branching-bisimilar to the synchronous product of its decomposition.

5 Proof that no decomposition operation maintains $\leftrightarrow_{db}$

In this section, we prove that there is no way of decomposing an LTS such that it is divergence-preserving branching-bisimilar to the synchronous product of its decomposition.

We define confluence based on [8].

Definition 5.1 (Confluence)**.** An LTS $(S, \Sigma_1 \cup \Sigma_2, \rightarrow, s_0)$ is called confluent over Σ_1 and Σ_2 iff for all states $s, s_a, s_b \in S$ and for all $a \in \Sigma_1$ and $b \in \Sigma_2$, if $s \xrightarrow{a} s_a$ and $s \xrightarrow{b} s_b$, then there is a state s_c such that $s_b \xrightarrow{a} s_c$ and $s_a \xrightarrow{b} s_c$.

Lemma 5.2. Any LTS M that is the synchronous product (Definition 2.4) of two LTSs M_1 and M_2 whose action sets are Σ_1 and Σ_2 respectively, is confluent over two sets $\Sigma_1 \setminus \overline{\Sigma_2}$ and $\Sigma_2 \setminus \overline{\Sigma_1}$.

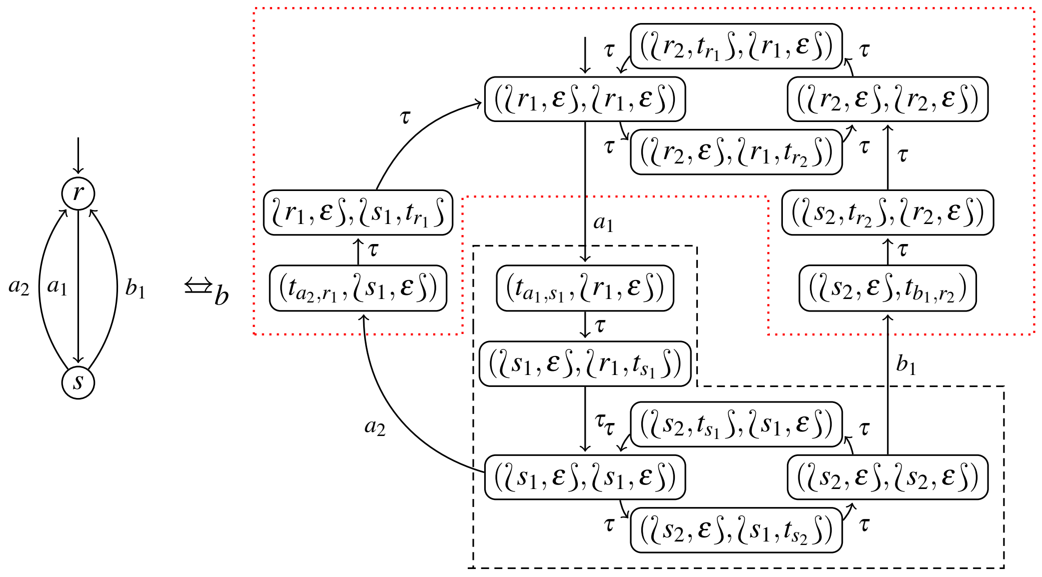


Figure 6: Showing branching bisimulation following Figure 5.

Proof. Consider the synchronous product $(S_1 \times S_2, \Sigma_x, \rightarrow_x, (q_0, r_0))$ from Definition 2.4 and some actions $a \in \Sigma_1 \setminus \overline{\Sigma_2}$ and $b \in \Sigma_2 \setminus \overline{\Sigma_1}$. Then $a \neq \overline{b}$. Consider some states $s, s' \in S_1$, $t, t' \in S_2$, and $s_a \in S_1 \times S_2$. We know that if $(s,t) \xrightarrow{a} s_a$ then that is due to a transition $s \xrightarrow{a} s'$ and that makes $s_a = (s', t)$, and that given a transition $t \xrightarrow{b} t'$, then a transition $(s',t) \xrightarrow{b} (s',t')$ is possible. Similarly, if $(s,t) \xrightarrow{b} (s,t')$ then $(s,t') \xrightarrow{a} (s',t')$. Therefore, the defined synchronous product is confluent. □

Figure 7 (centre) shows a simple LTS P. Concretely, it is defined as $(\{p,r,s\}, \Sigma_1 \cup \Sigma_2, \rightarrow, p)$ with alphabets $\Sigma_1 = \{a\}$ and $\Sigma_2 = \{b\}$ and transitions $p \xrightarrow{a} r$ and $p \xrightarrow{b} s$. In the following lemma and theorem, we prove that no way of decomposing P maintains divergence-preserving branching bisimulation.

Lemma 5.3. Given the LTS P (Figure 7, centre) with action set $\Sigma_1 \cup \Sigma_2$, let P_1 and P_2 be two LTSs with action sets Σ_{P_1} and Σ_{P_2} respectively, and with $\Sigma_1 \subseteq \Sigma_{P_1}$ and $\Sigma_2 \subseteq \Sigma_{P_2}$ and $\Sigma_1 \cap \Sigma_2 = \emptyset = \Sigma_1 \cap \Sigma_{P_2} = \Sigma_{P_1} \cap \Sigma_2$. Let P_x be the synchronous product $P_1 \times P_2$ by Definition 2.4. Then $P \not\leftrightarrow_{db} P_x$.

Proof. We prove this lemma by contradiction. Assume that $P \leftrightarrow_{db} P_x$, and let p_x be the initial state of P_x, then $p \leftrightarrow_{db} p_x$. As p is not divergent and cannot do a τ-transition, it holds that only finite sequences of τ's are possible from p_x. This can be seen as follows. If $p_x \xrightarrow{\tau} p_1 \xrightarrow{\tau} p_2 \xrightarrow{\tau} \cdots$, then $p \leftrightarrow_b p_i$ for all $i > 0$. Hence, p_x is divergent. But this is not possible because p is not divergent. So, p_x takes a finite number of τ steps to reach some state p'_x where $p'_x \not\xrightarrow{\tau}$.

Since it must be that $p \leftrightarrow_{db} p'_x$, and since $p \xrightarrow{a} r$ and $p \xrightarrow{b} s$ where $a \in \Sigma_1$ and $b \in \Sigma_2$, then there are two states r_x and s_x such that $p'_x \xrightarrow{a} r_x$ and $p'_x \xrightarrow{b} s_x$, and $r \leftrightarrow_{db} r_x$ and $s \leftrightarrow_{db} s_x$.

Now because $a \in \Sigma_1 \setminus \overline{\Sigma_2}$ and $b \in \Sigma_2 \setminus \overline{\Sigma_1}$, then P_x is confluent over these two sets, then there must exist a state p''_x such that $r_x \xrightarrow{b} p''_x$. However, $r \not\xrightarrow{b}$. Therefore, $r \not\leftrightarrow_{db} r_x$. Contradiction. Therefore $P \not\leftrightarrow_{db} P_x$. □

The proof is illustrated in Figure 7 showing that divergence-preserving branching bisimulation ($\leftrightarrow_{db}$) does not hold when decomposing the LTS P due to the confluence property of decompositions. On the other hand (literally the other hand of the same figure), branching bisimulation holds when decomposing

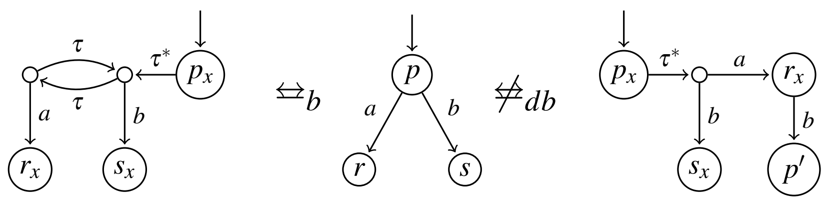


Figure 7: Illustration for Lemma 5.3.

LTS P. The reason it holds under $\leftrightarrow_b$, but not under $\leftrightarrow_{db}$ is that the former admits infinite τ cycles, i.e. divergence, which, as demonstrated here in right side of the figure, avoids the premise of confluence altogether. [2] provides a similar insight into how divergence maintains branching bisimilarity but breaks divergence-preserving branching bisimilarity.

Theorem 5.4. There is no decomposition operation that maintains divergence-preserving branching bisimulation ($\leftrightarrow_{db}$) for all LTSs.

Proof. We prove this theorem by contradiction. Assume that there is a decomposition operation that maintains $\leftrightarrow_{db}$ for all LTSs. Then it must do so for any arbitrary LTS P. But since Lemma 5.3 proves that no LTS maintains $\leftrightarrow_{db}$ for one such LTS P, i.e the one in Figure 7, then there is no decomposition operation that maintains $\leftrightarrow_{db}$ for all LTSs. □

6 Interpretation

One way to understand this fundamental result is that if the subsystems of the decomposition must communicate, then there is no escape from introducing divergence in order to maintain equivalence over any and all decompositions of LTSs.

With respect to automata learning, this result implies that, unless one values divergency, it is not possible to make any assumption about the distribution of components based on the information that is learned. If one can observe divergencies while learning behaviour, it might be possible to say something about the internal structure of a system though this will be highly non trivial to accomplish.

And with respect to software, our result says that it is always possible to distribute a piece of software over different components if one allows divergent behaviour. Otherwise, such a distribution is not possible and based on our proof. One can see that this already applies to very simple behaviours.

Furthermore, divergence, in an industrial context, is undesired due to the requirement of fairness, i.e., one subsystem seizing unfair control over the total behaviour of the system through infinite looping. This means that if some decomposition is found to maintain fairness, then that is guaranteed not to be the case universally over all contexts and all LTSs.

References

[1] Dana Angluin (1987): *Learning Regular Sets from Queries and Counterexamples*. Inf. Comput. 75(2), pp. 87–106, doi:10.1016/0890-5401(87)90052-6.

[2] Jos C. M. Baeten, Bas Luttik & Paul van Tilburg (2013): *Reactive Turing machines*. Inf. Comput. 231, pp. 143–166, doi:10.1016/j.ic.2013.08.010.

[3] Hendrik Brinksma & Romanus Langerak (1995): *Functionality Decomposition by Compositional Correctness Preserving Transformation*. South African Computer Journal 13, pp. 2–13.

[4] Omar al Duhaiby, Arjan J. Mooij, Hans van Wezep & Jan Friso Groote (2018): *Pitfalls in Applying Model Learning to Industrial Legacy Software*. In: Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice - 8th International Symposium, ISoLA 2018, Limassol, Cyprus, November 5-9, 2018, Proceedings, Part IV, pp. 121–138, doi:10.1007/978-3-030-03427-6_13.

[5] Rob J. van Glabbeek, Bas Luttik & Nikola Trcka (2009): *Branching Bisimilarity with Explicit Divergence*. Fundam. Inform. 93(4), pp. 371–392, doi:10.3233/FI-2009-109.

[6] Rob J. van Glabbeek & W. P. Weijland (1996): *Branching Time and Abstraction in Bisimulation Semantics*. J. ACM 43(3), pp. 555–600, doi:10.1145/233551.233556.

[7] Jan Friso Groote & Mohammad Reza Mousavi (2014): *Modeling and Analysis of Communicating Systems*. MIT Press, doi:10.7551/mitpress/9946.001.0001.

[8] Jan Friso Groote & M. P. A. Sellink (1996): *Confluence for Process Verification*. Theor. Comput. Sci. 170(1-2), pp. 47–81, doi:10.1016/S0304-3975(96)80702-X.

[9] Maria Hultström (1994): *Structural decomposition*. In: Protocol Specification, Testing and Verification XIV, Proceedings of the Fourteenth IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification, Vancouver, BC, Canada, 1994, pp. 201–216.

[10] Kenneth Krohn, Richard Mateosian & John Rhodes (1967): *Methods of the Algebraic Theory of Machines. I: Decomposition Theorem for Generalized Machines; Properties Preserved under Series and Parallel Compositions of Machines*. J. Comput. Syst. Sci. 1(1), pp. 55–85, doi:10.1016/S0022-0000(67)80007-2.

[11] Bas Luttik (2016): *Unique parallel decomposition in branching and weak bisimulation semantics*. Theor. Comput. Sci. 612, pp. 29–44, doi:10.1016/j.tcs.2015.10.013.

[12] Robin Milner & Faron Moller (1993): *Unique Decomposition of Processes*. Theor. Comput. Sci. 107(2), pp. 357–363, doi:10.1016/0304-3975(93)90176-T.

On the Meaning of Transition System Specifications

Rob van Glabbeek
Data61, CSIRO, Sydney, Australia
School of Computer Science and Engineering, University of New South Wales, Sydney, Australia
rvg@cs.stanford.edu

Transition System Specifications provide programming and specification languages with a semantics. They provide the meaning of a closed term as a *process graph*: a state in a labelled transition system. At the same time they provide the meaning of an n-ary operator, or more generally an open term with n free variables, as an n-ary operation on process graphs. The classical way of doing this, the *closed-term semantics*, reduces the meaning of an open term to the meaning of its closed instantiations. It makes the meaning of an operator dependent on the context in which it is employed. Here I propose an alternative *process graph semantics* of TSSs that does not suffer from this drawback.

Semantic equivalences on process graphs can be lifted to open terms conform either the closed-term or the process graph semantics. For pure TSSs the latter is more discriminating.

I consider five sanity requirements on the semantics of programming and specification languages equipped with a recursion construct: *compositionality*, applied to n-ary operators, recursion and variables, *invariance under α-conversion*, and the *recursive definition principle*, saying that the meaning of a recursive call should be a solution of the corresponding recursion equations. I establish that the satisfaction of four of these requirements under the closed-term semantics of a TSS implies their satisfaction under the process graph semantics.

1 Introduction

Transition System Specifications (TSSs) [16] are a formalisation of *Structural Operational Semantics* [22] providing programming and specification languages with an interpretation. They provide the meaning of a closed term as a *process graph*: a state in a labelled transition system. At the same time they provide the meaning of an n-ary operator of the language, or more generally an open term with n free variables, as an n-ary operation on process graphs. The classical way of doing this proceeds by reducing the meaning of an open term to the meaning of its closed instantiations. I call this the *closed-term semantics* of TSSs. A serious shortcoming of this approach is that it makes the meaning of an operator dependent on the context in which it is employed.

Example 1 Consider a TSS featuring unary operators f, id and $a.\_$ for each action a drawn from an alphabet A, and a constant 0. The set of admitted transition labels is $Act := A \uplus \{\tau\}$. The transition rules are

$$a.x \xrightarrow{a} x \ \text{(for all } a \in A) \qquad \frac{x \xrightarrow{a} x'}{f(x) \xrightarrow{a} f(x')} \ \text{(for all } a \in A) \qquad \frac{x \xrightarrow{\alpha} x'}{id(x) \xrightarrow{\alpha} id(x')} \ \text{(for all } \alpha \in Act)$$

When considered in their own right, the operators f and id are rather different: the latter can mimic τ-transitions of its argument, and the former can not. Yet, in the context of the given TSS, one has $f(p) \leftrightarroweq id(p)$, no matter which term p is substituted for the argument of these operators. Here $\leftrightarroweq$ denotes strong bisimulation equivalence, as defined in [21, 12]. This is because no process in the given TSS ever generates a transition with the label τ. The identification of f and id up to $\leftrightarroweq$ can be considered unfortunate, one reason being that is ceases to hold as soon as the language is enriched with a fresh operator $\tau.\_$ with the transition rule $\tau.x \xrightarrow{\tau} x$. As I will show later, this invalidates an intuitively plausible theorem on the relative expressiveness of specification languages.

Here I propose an alternative *process graph semantics* of TSSs that does not suffer from this drawback.

In [10] I proposed five requirements on the semantics of programming and specification languages equipped with a recursion construct: *compositionality*, applied to *n*-ary operators, recursion and variables, *invariance under α-conversion*, and the *recursive definition principle*, saying that the meaning of a recursive call should be a solution of the corresponding recursion equations.

In many prior works on structural operational semantics, (some of) these requirements have been shown to hold for various TSSs when employing the closed-term semantics. It would be time consuming to redo all that work for the process graph semantics proposed here. To prevent this I show that the satisfaction of four of these requirements under the closed-term semantics of a TSS implies their satisfaction under the process graph semantics. The remaining requirement holds almost always.

Overview of the paper Section 2 presents the syntax of the programming and specification languages I consider here. For simplicity I restrict myself to languages with single-sorted signature, optionally featuring a recursion construct. This is a rich enough setting to include process algebras like CCS [21], CSP [5], ACP [3], MEIJE [1, 24] and SCCS [20].

The traditional "*closed-term*" interpretation of the process calculi CCS, MEIJE and SCCS effectively collapses syntax and semantics by interpreting the entire language as one big labelled transition system (LTS) in which the closed terms of the language constitute the set of states. This LTS is generated by a TSS, as formally defined in Section 5. Semantic equivalences on LTSs thereby directly relate closed terms. Two open terms are judged equivalent iff each of their closed substitutions are.

In Section 3 I present an interpretation of programming and specification languages that is more common in universal algebra and mathematical logic [19], and is also used in the traditional semantics of ACP and CSP. It separates syntax and semantics through a semantic mapping that associates with each closed term a value, and with each open term an operation on values. This matches with what often is called *denotational semantics*, except that I do not require the meaning of recursion constructs to be provided by means of fixed point techniques. In Section 7 I specialise this general approach to an operational one by taking the values to be *process graphs*: states in labelled transition systems. Likewise, Section 6 casts the closed-term interpretation as a special case of the approach from Section 3, by taking the values to be the closed terms.

Section 3 also formulates the five sanity requirements mentioned above, most in a couple of equivalent forms. Section 6 shows how these requirements simplify to better recognisable forms under the closed-term interpretation of programming and specification languages. These requirements are parametrised by the choice of a semantic equivalence $\sim$ on values, relating values that one does not need to distinguish. The traditional treatments of universal algebra and mathematical logic, and the process algebra CSP, do not involve such a semantic equivalence; this corresponds to letting $\sim$ be the identity relation. In Section 4 I observe that any choice of $\sim$ can be reduced to the identity relation, namely by taking as values $\sim$-equivalence classes of values. This reduction preserves the five sanity requirements.

After these preparations, Section 8 defines the promised process graph interpretation of TSSs. Some TSSs do not have a process graph interpretation, but I show that the large class of *pure* TSSs do.

Semantic equivalences on process graphs can be lifted to open terms conform either the closed-term or the process graph interpretation. Section 9 shows, under some mild conditions, that for pure TSSs the latter is more discriminating. Section 10 illustrates on a practical process algebra that whether a semantic equivalence is a congruence may depend on which of the two interpretations is chosen.

Section 11 proves the promised result that when four of the five sanity requirements have been established for the closed-term interpretation of a TSS, they also hold for its process graph interpretation. It also shows that the remaining requirement almost always holds

Section 12 argues that something is gained by moving from the closed-term interpretation of TSSs to the process-graph interpretation. Based on Example 1 it formulates an intuitively plausible theorem relating relative expressiveness of specification languages and conservative extensions, and shows how this theorem fails under the closed-term interpretation, but holds under the process graph interpretation.

Section 13 addresses related work, and Section 14 evaluates the five sanity requirements for languages specified by TSSs of a specific form.

2 Syntax

In this paper *Var* is an infinite set of *variables*, ranged over by X, Y, x, y, x_i etc.

Definition 1 (*Terms*). A *function declaration* is a pair (f,n) of a *function symbol* $f \notin \mathit{Var}$ and an *arity* $n \in \mathbb{N}$.[1] A function declaration $(c,0)$ is also called a *constant declaration*. A *signature* is a set of function declarations. The set $\mathbb{T}^r(\Sigma)$ of *terms with recursion* over a signature Σ is defined inductively by:

- $\mathit{Var} \subseteq \mathbb{T}^r(\Sigma)$,
- if $(f,n) \in \Sigma$ and $t_1,\dots,t_n \in \mathbb{T}^r(\Sigma)$ then $f(t_1,\dots,t_n) \in \mathbb{T}^r(\Sigma)$,
- If $V_S \subseteq \mathit{Var}$, $S : V_S \rightarrow \mathbb{T}^r(\Sigma)$ and $X \in V_S$, then $\langle X|S\rangle \in \mathbb{T}^r(\Sigma)$.

A term $c()$ is abbreviated as c. A function S as appears in the last clause is called a *recursive specification*. It is often displayed as $\{X = S_X \mid X \in V_S\}$. Each term S_Y for $Y \in V_S$ counts as a subterm of $\langle X|S\rangle$. An occurrence of a variable y in a term t is *free* if it does not occur in a subterm of the form $\langle X|S\rangle$ with $y \in V_S$. For $t \in \mathbb{T}^r(\Sigma)$ a term, $\mathit{var}(t)$ denotes the set of variables occurring free in t. A term is *closed* if it contains no free occurrences of variables. For $W \subseteq \mathit{Var}$, let $\mathbb{T}^r(\Sigma, W)$ denote the set of terms t with $\mathit{var}(t) \subseteq W$, and let $\mathrm{T}^r(\Sigma) = \mathbb{T}^r(\Sigma, \emptyset)$ be the set of closed terms over Σ. The sets $\mathbb{T}(\Sigma)$, $\mathbb{T}(\Sigma, W)$ and $\mathrm{T}(\Sigma)$ of open and closed terms over Σ without recursion are defined likewise, but without the last clause.

Definition 2 (*Substitution*). A Σ-*substitution* σ is a partial function from *Var* to $\mathbb{T}^r(\Sigma)$. It is *closed* if it is a total function from *Var* to $\mathrm{T}^r(\Sigma)$. If σ is a substitution and t a term, then $t[\sigma]$ denotes the term obtained from t by replacing, for x in the domain of σ, every free occurrence of x in t by $\sigma(x)$, while renaming bound variables if necessary to prevent name-clashes. In that case $t[\sigma]$ is called a *substitution instance* of t. A substitution instance $t[\sigma]$ where σ is given by $\sigma(x_i) = u_i$ for $i \in I$ is denoted as $t[u_i/x_i]_{i \in I}$, and for S a recursive specification $\langle t|S\rangle$ abbreviates $t[\langle Y|S\rangle/Y]_{Y \in V_S}$.

Sometimes the syntax of a language is given as a signature together with an annotation that places some restrictions on the use of recursion [10]. This annotation may for instance require the sets V_S to be finite, the functions S computable, or the sets of equations S to be *guarded*: a syntactic criterion that ensures that they have unique solutions under a given interpretation. It may also rule out recursion altogether.

3 Semantics

A *language* can be given by an annotated signature, specifying its syntax, and an *interpretation*, assigning to every term t its meaning $[\![t]\!]$. The meaning of a closed term is a *value* chosen from a class of values $\mathbb{D}$,

[1] This work generalises seamlessly to operators with infinitely many arguments. Such operators occur, for instance, in [4, Appendix A.2]. Hence one may take n to be any ordinal. It also generalises to operators, like the *summation* or *choice* of CCS [21], that take any *set* of arguments.

which is called a *domain*. The meaning of an open term is a *Var-ary operation* on $\mathbb{D}$: a function of type $\mathbb{D}^{Var} \to \mathbb{D}$, where $\mathbb{D}^{Var}$ is the class of functions from *Var* to $\mathbb{D}$. It associates a value $[\![t]\!](\rho) \in \mathbb{D}$ to t that depends on the choice of a *valuation* $\rho : Var \to \mathbb{D}$. The valuation assigns a value from $\mathbb{D}$ to each variable.

A *partial valuation* is a function $\xi : W \to \mathbb{D}$ for $W \subseteq Var$ that assigns a value only to certain variables. A *W-ary operation*, for $W \subseteq Var$, is a function $F : \mathbb{D}^W \to \mathbb{D}$. It associates a value to every W-tuple of values, i.e. to every partial valuation of the variables with domain W. If $F : \mathbb{D}^{Var} \to \mathbb{D}$ and $\zeta \in \mathbb{D}^{Var \backslash W}$ then $F(\zeta) : \mathbb{D}^W \to \mathbb{D}$ is given by $F(\zeta)(\xi) = F(\xi \cup \zeta)$ for any $\xi \in \mathbb{D}^W$. For ρ a valuation and W a set of variables, $\rho \backslash W$ is the partial valuation with domain $Var \backslash W$ such that $(\rho \backslash W)(x) = \rho(x)$ for $x \in Var \backslash W$.

3.1 Sanity requirements on interpretations

Usually interpretations are required to satisfy some sanity requirements. The work [10] proposed five such requirements: *compositionality*, applied to variables, *n*-ary operators and recursion, *invariance under α-conversion*, and the *recursive definition principle* (RDP).

In this paper I work with domains of interpretation $\mathbb{D}$ that are equipped with a semantic equivalence relation $\sim \subseteq \mathbb{D} \times \mathbb{D}$. It indicates that values $v, w \in \mathbb{D}$ with $v \sim w$ need not be distinguished on our chosen level of abstraction. The equivalence $\sim$ extends to functions $F, G : \mathbb{D}^W \to \mathbb{D}$ by $F \sim G$ iff $F(\xi) \sim G(\xi)$ for all $\xi \in \mathbb{D}^W$. It extends to partial valuations $\rho, \nu : W \to \mathbb{D}$ or functions ρ, ν of type $(\mathbb{D}^W \to \mathbb{D})^W$ by $\rho \sim \nu$ iff $\rho(X) \sim \nu(X)$ for all $X \in W$. Such an equivalence relation relaxes the requirements invariance under α-conversion and RDP, and modifies compositionality; I speak of *compositionality up to* $\sim$. The default case in which no semantic equivalence is in force corresponds to taking $\sim$ to be the identity relation.

Compositionality up to $\sim$ demands that the meaning of a variable is given by the chosen valuation, i.e.,

$$[\![x]\!](\rho) \sim \rho(x) \tag{1}$$

for each $x \in Var$ and valuation $\rho : Var \to \mathbb{D}$, and that the meaning of a term is completely determined by the meaning of its direct subterms. This means that for operators $(f,n) \in \Sigma$ and valuations $\rho, \nu : Var \to \mathbb{D}$

$$[\![t_i]\!](\rho) \sim [\![u_i]\!](\nu) \text{ (for all } i = 1,...,n) \quad \Rightarrow \quad [\![f(t_1,...,t_n)]\!](\rho) \sim [\![f(u_1,...,u_n)]\!](\nu) \tag{2}$$

and for recursive specifications S and S' with $X \in V_S = V_{S'}$ and valuations $\rho, \nu : Var \to \mathbb{D}$

$$[\![S_Y]\!](\rho \backslash V_S) \sim [\![S'_Y]\!](\nu \backslash V_S) \text{ (for all } Y \in V_S) \quad \Rightarrow \quad [\![\langle X|S\rangle]\!](\rho) \sim [\![\langle X|S'\rangle]\!](\nu). \tag{3}$$

Note that the precondition of (3) evaluates the variables in S_Y and S'_Y that are free in $\langle X|S\rangle$ and $\langle X|S'\rangle$ according to ρ and ν, respectively, and the variables from $V_S = V_{S'}$ under any common valuation ζ.

Invariance under α-conversion demands that the meaning of a term is independent of the names of its bound variables, i.e. for any injective substitution $\gamma : V_S \to Var$ such that the range of γ contains no variables occurring free in $\langle S_Y|S\rangle$ for some $Y \in V_S$

$$[\![\langle \gamma(X)|S[\gamma]\rangle]\!] \sim [\![\langle X|S\rangle]\!]. \tag{4}$$

Finally, the meaning of a term $\langle X|S\rangle$ should be the X-component of a solution of S. To be precise,

$$[\![\langle X|S\rangle]\!] \sim [\![\langle S_X|S\rangle]\!]. \tag{5}$$

This property is called the *recursive definition principle* [6].

A straightforward structural induction on t using (1), (2) and (3) shows that $[\![t]\!](\rho)$ depends only on the restriction of ρ to those variables that occur free in t. If there are no such variables, $[\![t]\!](\rho)$ does not depends on ρ at all, and consequently can be abbreviated to $[\![t]\!]$.

3.2 Alternative forms of the sanity requirements

Note that (2) holds iff for every $(f,n) \in \Sigma$ there is a function $f^{\mathbb{D}} : \mathbb{D}^n \to \mathbb{D}$ such that

$$\forall \rho \in \mathbb{D}^{Var} : [\![f(t_1,\dots,t_n)]\!](\rho) \sim f^{\mathbb{D}}([\![t_1]\!](\rho),\dots,[\![t_n]\!](\rho)).$$

Requirement (3) can be characterised in the same vein:

Proposition 1 Property (3) holds iff for every set $W \subseteq Var$ there is a function $\mu_W^{\mathbb{D}} : (\mathbb{D}^W \to \mathbb{D})^W \to \mathbb{D}^W$ such that for every recursive specification $S : W \to \mathbb{T}^r(\Sigma)$ with $X \in W$, and every $\rho : Var \to \mathbb{D}$,

$$[\![\langle X|S\rangle]\!](\rho) \sim \mu_W^{\mathbb{D}}([\![S]\!](\rho\backslash W))(X).$$

Proof: Since the meaning of term $S_Y \in \mathbb{T}^r(\Sigma)$ is of type $\mathbb{D}^{Var} \to \mathbb{D}$, the meaning of a recursive specification $S : W \to \mathbb{T}^r(\Sigma)$ is of type $[\![S]\!] : W \to (\mathbb{D}^{Var} \to \mathbb{D})$. Applying to that a partial valuation $\rho\backslash W : Var\backslash W \to \mathbb{D}$ yields a function $[\![S]\!](\rho\backslash W)$ of type $(\mathbb{D}^W \to \mathbb{D})^W$. Now for each $\chi \in (\mathbb{D}^W \to \mathbb{D})^W$ choose, if possible, a pair $S^\chi : W \to \mathbb{T}^r(\Sigma)$ and $\rho^\chi : Var \to \mathbb{D}$ such that $[\![S^\chi]\!](\rho^\chi\backslash W) \sim \chi$, and define $\mu_W^{\mathbb{D}}(\chi)(X)$, for $X \in W$, to be $[\![\langle X|S^\chi\rangle]\!](\rho^\chi)$. For a χ for which no such pair can be found the definition of $\mu_W^{\mathbb{D}}(\chi)(X)$ is arbitrary. Now pick $S : W \to \mathbb{T}^r(\Sigma)$ and $\rho : Var \to \mathbb{D}$. Let $\chi := [\![S]\!](\rho\backslash W)$. Then $[\![S]\!](\rho\backslash W) = \chi \sim [\![S^\chi]\!](\rho^\chi\backslash W)$, so by (3) one has

$$[\![\langle X|S\rangle]\!](\rho) \sim [\![\langle X|S^\chi\rangle]\!](\rho^\chi) = \mu_W^{\mathbb{D}}(\chi)(X) = \mu_W^{\mathbb{D}}([\![S]\!](\rho\backslash W))(X).$$

The other direction, that the existence of such a $\mu_W^{\mathbb{D}}$ implies (3), is trivial. □

Write $t \overset{\alpha}{=} u$ if the terms $t,u \in \mathbb{T}^r(\Sigma)$ differ only in the names of their bound variables. Then (4) can be rewritten as

$$t \overset{\alpha}{=} u \quad \Rightarrow \quad [\![t]\!] \sim [\![u]\!]. \tag{6}$$

Proposition 2 In the presence of (2) and (3), (4) is equivalent to (6).

Proof: Clearly, (4) is a special case of (6). The other direction proceeds by structural induction on t.

In case $t = X \in Var$ then $u = X$ and thus $[\![t]\!] \sim [\![u]\!]$.

Let $t = f(t_1,...,t_n)$. Then $u = f(u_1,...,u_n)$ and $t_i \overset{\alpha}{=} u_i$ for each $i = 1,...,n$. By induction $[\![t_i]\!] \sim [\![u_i]\!]$ for each i. This means that $[\![t_i]\!](\rho) \sim [\![u_i]\!](\rho)$ for each $\rho : Var \to \mathbb{D}$. Hence, by (2), $[\![t]\!] \sim [\![u]\!]$.

Let $t = \langle X|S\rangle$. Then $u = \langle \gamma(X)|S'[\gamma]\rangle$ for a recursive specification $S' : V_S \to \mathbb{T}^r(\Sigma)$ with $S_Y \overset{\alpha}{=} S'_Y$ for all $Y \in V_S$, and an injective substitution $\gamma : V_S \to Var$ such that the range of γ contains no variables occurring free in $\langle S'_Y|S\rangle$ for some $Y \in V_S$. By induction $[\![S_Y]\!] \sim [\![S'_Y]\!]$ for each Y. This means that $[\![S_Y]\!](\rho) \sim [\![S'_Y]\!](\rho)$ for each $\rho : Var \to \mathbb{D}$, so in particular $[\![S_Y]\!](\rho\backslash V_S) \sim [\![S'_Y]\!](\rho\backslash V_S)$ for each such ρ. Hence, by (3), $[\![\langle X|S\rangle]\!] \sim [\![\langle X|S'\rangle]\!]$. Thus, by (4), $[\![t]\!] \sim [\![\langle X|S'\rangle]\!] \sim [\![\langle \gamma(X)|S'[\gamma]\rangle]\!] = [\![u]\!]$. □

3.3 Applying semantic interpretations to substitutions

The semantic mapping $[\![\quad]\!] : \mathbb{T}^r(\Sigma) \to ((Var \to \mathbb{D}) \to \mathbb{D})$ extends to substitutions $\sigma : Var \rightharpoonup \mathbb{T}^r(\Sigma)$ by $[\![\sigma]\!](\rho)(X) := [\![\sigma(X)]\!](\rho)$ for all $X \in Var$ and $\rho : Var \to \mathbb{D}$—here σ is extended to a total function by $\sigma(Y) := Y$ for all $Y \notin dom(\sigma)$. Thus $[\![\sigma]\!]$ is of type $(Var \to \mathbb{D}) \to (Var \to \mathbb{D})$, i.e. a map from valuations to valuations. The following results applies to languages satisfying sanity requirements (1)–(4).

Proposition 3 Let $t \in \mathbb{T}^r(\Sigma)$ be a term, $\sigma : Var \rightharpoonup \mathbb{T}^r(\Sigma)$ a substitution, and $\rho : Var \to \mathbb{D}$ a valuation. Then

$$[\![t[\sigma]]\!](\rho) \sim [\![t]\!]([\![\sigma]\!](\rho)). \tag{7}$$

Proof: By the definition of substitution, there is an $u \in \mathbb{T}^r(\Sigma)$ with $t \overset{\alpha}{=} u$, such that $t[\sigma] = u[\sigma]$, and when performing the substitution σ on u there is no need to rename any bound variables occurring in u. It now suffices to obtain $[\![u[\sigma]]\!](\rho) \sim [\![u]\!]([\![\sigma]\!](\rho))$, because then $[\![t[\sigma]]\!](\rho) = [\![u[\sigma]]\!](\rho) \sim [\![u]\!]([\![\sigma]\!](\rho)) \overset{(6)}{\sim} [\![t]\!]([\![\sigma]\!](\rho))$. For this reason it suffices to establish (7) for terms t and substitutions σ with the property

(*) that whenever a variable Z occurs free within a subterm $\langle X|S\rangle$ of t with $Y \in V_S$ then Y does not occur free in $\sigma(Z)$. I proceed with structural induction on t, while quantifying over all ρ.

Let $t = X \in Var$. Then $[\![X]\!]([\![\sigma]\!](\rho)) \overset{(1)}{\sim} ([\![\sigma]\!](\rho))(X) \overset{def}{=} [\![\sigma(X)]\!](\rho) = [\![X[\sigma]]\!](\rho)$.

Let $t = f(t_1,...,t_n)$. By induction I may assume that $[\![t_i[\sigma]]\!](\rho) \sim [\![t_i]\!]([\![\sigma]\!](\rho))$ for $i = 1,...,n$. Hence

$$[\![f(t_1,\dots,t_n)[\sigma]]\!](\rho) = [\![f(t_1[\sigma],\dots,t_n[\sigma])]\!](\rho) \overset{(2)}{\sim} [\![f(t_1,\dots,t_n)]\!]([\![\sigma]\!](\rho)).$$

Let $t = \langle X|S\rangle$. Then $t[\sigma] = \langle X|S[\sigma\backslash V_S]\rangle$. Given that the pair t,σ satisfies property (*), so do the pairs $S_Y, \sigma\backslash V_S$ for all $Y \in V_S$. Moreover, no $Y \in V_S$ occurs free in $\sigma(Z)$ for $Z \notin V_S$ occurring free in S. (#)
By induction I assume $[\![S_Y[\sigma\backslash V_S]]\!](\rho) \sim [\![S_Y]\!]\big([\![\sigma\backslash V_S]\!](\rho)\big)$ for all $Y \in V_S$ and all $\rho : Var \to \mathbb{D}$. Any such ρ can be written as $(\rho\backslash V_S) \cup \xi$ for some $\xi : V_S \to \mathbb{D}$. Now (#) yields that for all $Z \in Var$ occurring free in S

$$\big([\![\sigma\backslash V_S]\!]((\rho\backslash V_S)\cup\xi)\big)(Z) = \big((([\![\sigma]\!](\rho))\backslash V_S)\cup\xi\big)(Z).$$

Hence $[\![S_Y[\sigma\backslash V_S]]\!](\rho\backslash V_S)(\xi) \sim [\![S_Y]\!]\big([\![\sigma\backslash V_S]\!]((\rho\backslash V_S)\cup\xi)\big) = [\![S_Y]\!]\big((([\![\sigma]\!](\rho))\backslash V_S\big)(\xi)$ for all $Y \in V_S$, $\rho : Var \to \mathbb{D}$ and $\xi : V_S \to \mathbb{D}$. So $[\![S_Y[\sigma\backslash V_S]]\!](\rho\backslash V_S) \sim [\![S_Y]\!]\big(([\![\sigma]\!](\rho))\backslash V_S\big)$ for all $Y \in V_S$ and $\rho : Var \to \mathbb{D}$.
One obtains $\quad [\![\langle X|S\rangle[\sigma]]\!](\rho) = [\![\langle X|S[\sigma\backslash V_S]\rangle]\!](\rho) \overset{(3)}{\sim} [\![\langle X|S\rangle]\!]([\![\sigma]\!](\rho)).$ □

4 Quotient Domains

An equivalence relation $\sim$ on $\mathbb{D}$ is a *congruence*[2] for $\mathscr{L}$ if

$$\rho \sim \nu \;\Rightarrow\; [\![t]\!](\rho) \sim [\![t]\!](\nu) \tag{8}$$

for any term t and any valuations $\rho, \nu : Var \to \mathbb{D}$.

Proposition 4 If a language $\mathscr{L}$ is compositional up to an equivalence $\sim$ then $\sim$ is a congruence for $\mathscr{L}$.

Proof: A straightforward structural induction on t. □

Given a domain $\mathbb{D}$ for interpreting languages and an equivalence relation $\sim$, the *quotient domain* $\mathbb{D}/_\sim$ consists of the $\sim$-equivalence classes of elements of $\mathbb{D}$. For $v \in \mathbb{D}$ let $[v]_\sim \in \mathbb{D}/_\sim$ denote the equivalence class containing $v \in \mathbb{D}$. Likewise, for a valuation $\rho : Var \to \mathbb{D}$ in $\mathbb{D}$, the valuation $[\rho]_\sim : Var \to \mathbb{D}/_\sim$ in $\mathbb{D}/_\sim$ is given by $[\rho]_\sim(x) := [\rho(x)]_\sim$; it also represents the $\sim$-equivalence class of valuations in $\mathbb{D}$ of which ρ is a member. Each valuation in $\mathbb{D}/_\sim$ is of the form $[\rho]_\sim$.

An interpretation $[\![\_]\!] : \mathbb{T}^r(\Sigma) \to (\mathbb{D}^{Var} \to \mathbb{D})$ that satisfies (8) is turned into the *quotient interpretation* $[\![\_]\!]_\sim : \mathbb{T}^r(\Sigma) \to ((\mathbb{D}/_\sim)^{Var} \to \mathbb{D}/_\sim)$ by defining $[\![t]\!]_\sim([\rho]_\sim) := [[\![t]\!](\rho)]_\sim$. By (8), this is independent of the choice of a representative valuation ρ within the equivalence class $[\rho]_\sim$.

Let $[\![\_]\!]$ be an interpretation and $\sim$ an equivalence such that (8) holds. Then $[\![\_]\!]$ satisfies the sanity requirements (1)–(5) of Section 3 up to $\sim$ iff $[\![\_]\!]_\sim$ satisfies these requirements up to $=$.

[2]This property is called *lean congruence* in [13]. There $\sim$ is called a *full congruence* for $\mathscr{L}$ iff $\mathscr{L}$ is compositional up to $\sim$. In the absence of recursion this is equivalent to (8), but in general it is a stronger requirement—i.e., the reverse of Proposition 4 does not hold [13].

5 Transition System Specifications

Definition 3 (*Transition system specification;* GROOTE & VAANDRAGER [16]). Let Σ be an annotated signature and A a set (of *actions*). A *(positive)* (Σ,A)*-literal* is an expression $t \xrightarrow{a} t'$ with $t,t' \in \mathbb{T}^r(\Sigma)$ and $a \in A$. A *transition rule* over (Σ,A) is an expression of the form $\frac{H}{\lambda}$ with H a set of (Σ,A)-literals (the *premises* of the rule) and λ a (Σ,A)-literal (the *conclusion*). A rule $\frac{H}{\lambda}$ with $H = \emptyset$ is also written λ. A *transition system specification (TSS)* is a triple (Σ,A,R) with R a set of transition rules over (Σ,A).

The following definition (from [9]) tells when a literal is provable from a TSS. It generalises the standard definition (see e.g. [16]) by (also) allowing the derivation of transition rules. The derivation of a literal $t \xrightarrow{a} t'$ corresponds to the derivation of the rule $\frac{H}{t \xrightarrow{a} t'}$ with $H = \emptyset$. The case $H \neq \emptyset$ corresponds to the derivation of $t \xrightarrow{a} t'$ under the assumptions H.

Definition 4 (*Proof*). Let $P = (\Sigma,A,R)$ be a TSS. A *proof* of a transition rule $\frac{H}{\lambda}$ from P is a well-founded, upwardly branching tree of which the nodes are labelled by (Σ,A)-literals, such that:

- the root is labelled by λ, and
- if κ is the label of a node q and K is the set of labels of the nodes directly above q, then
  - either $K = \emptyset$ and $\kappa \in H$,
  - or $\frac{K}{\kappa}$ is a substitution instance of a rule from R.

If a proof of $\frac{H}{\lambda}$ from P exists, then $\frac{H}{\lambda}$ is *provable* from P, denoted $P \vdash \frac{H}{\lambda}$.

A *labelled transition system* (LTS) is a triple $(S,A,\rightarrow)$ with S a set of *states* or *processes*, A a set of *actions*, and $\rightarrow \subseteq S \times A \times S$ the *transition relation*, or set of *transitions*. A TSS $P = (\Sigma,A,R)$ *specifies* the LTS $(\mathrm{T}^r(\Sigma),A,\rightarrow)$ whose states are the closed terms over Σ and whose transitions are the closed literals provable from P.

For the sake of simplicity, the above treatment of TSSs deals with positive premises only. However, all results of this paper apply equally well, and with unaltered proofs, to TSS with negative premises $t \stackrel{a}{\nrightarrow}$, following the treatment below. The rest of the section may be skipped in first reading.

5.1 TSSs with negative premises

A *negative* (Σ,A)*-literal* is an expression $t \stackrel{a}{\nrightarrow}$. A transition rule may have positive and negative literals as premises, but must have a positive conclusion. Literals $t \xrightarrow{a} u$ and $t \stackrel{a}{\nrightarrow}$ are said to *deny* each other.

Definition 5 [11] Let $P = (\Sigma,A,R)$ be a TSS. A *well-supported proof* from P of a closed literal λ is a well-founded tree with the nodes labelled by closed literals, such that the root is labelled by λ, and if κ is the label of a node and K is the set of labels of the children of this node, then:

1. either κ is positive and $\frac{K}{\kappa}$ is a closed substitution instance of a rule in R;
2. or κ is negative and for each set N of closed negative literals with $\frac{N}{\nu}$ provable from P and ν a closed positive literal denying κ, a literal in K denies one in N.

$P \vdash_{ws} \lambda$ denotes that a well-supported proof from P of λ exists. A standard TSS P is *complete* if for each p and a, either $P \vdash_{ws} p \stackrel{a}{\nrightarrow}$ or there exists a closed term q such that $P \vdash_{ws} p \xrightarrow{a} q$.

In [11] it is shown that no TSS admit well-supported proofs of literals that deny each other. Only a complete TSSs specifies an LTS; its transitions are the closed positive literals with a well-supported proof.

6 The Closed-term Semantics of Transition System Specifications

The default semantics of a language given as a TSS (Σ,A,R) is to take the domain $\mathbb{D}$ in which the expressions are interpreted to be $\mathrm{T}^r(\Sigma)$, the set of closed terms over Σ. The meaning of a closed expression $p \in \mathrm{T}^r(\Sigma)$ is simply itself: $[\![p]\!] := p$. The meaning $[\![t]\!] \in \mathbb{D}^{Var} \rightarrow \mathbb{D}$ of an open expression $t \in \mathbb{T}^r(\Sigma)$ is given by $[\![t]\!](\rho) := t[\rho]$. Here one uses the fact that a valuation $\rho : Var \rightarrow \mathbb{D}$ is also a closed substitution $\rho : Var \rightarrow \mathrm{T}^r(\Sigma)$. Given a semantic equivalence relation $\sim \subseteq \mathrm{T}^r(\Sigma) \times \mathrm{T}^r(\Sigma)$, the closed-term semantics of a TSS always satisfies Requirement (1), whereas (2)–(5) simplify to

$$p_i \sim q_i \text{ (for all } i = 1,\dots,n) \;\Rightarrow\; f(p_1,\dots,p_n) \sim f(q_1,\dots,q_n) \quad \text{and} \tag{2$'$}$$

$$S_Y[\sigma] \sim S'_Y[\sigma] \text{ (for all } Y \in W \text{ and } \sigma : W \rightarrow \mathrm{T}^r(\Sigma)) \;\Rightarrow\; \langle X|S\rangle \sim \langle X|S'\rangle \tag{3$'$}$$

$$\langle \gamma(X)|S[\gamma]\rangle \sim \langle X|S\rangle \tag{4$'$}$$

$$\langle X|S\rangle \sim \langle S_X|S\rangle \tag{5$'$}$$

for all functions $(f,n) \in \Sigma$, closed terms $p_i,q_i \in \mathrm{T}^r(\Sigma)$, recursive specifications $S,S' : W \rightarrow \mathbb{T}^r(\Sigma,W)$ with $X \in W \subseteq Var$, and $\gamma : W \rightarrow Var$ injective.

7 Process Graphs

When the expressions in a language are meant to represent processes, they are called *process expressions*, and the language a *process description language*. Suitable domains for interpreting process description languages are the class of *process graphs* [3] and its quotients. In such *graph domains* a process is represented by either a process graph, or an equivalence class of process graphs. Process graphs are also known as *state-transition diagrams* or *automata*. They are LTSs equipped with an initial state. A process graph can also be seen as a state in an LTS.

Definition 6 A *process graph*, labelled over a set A of actions, is a triple $G = (S,A,\rightarrow,I)$ with

- S a set of *nodes* or *states*,
- $\rightarrow \subseteq S \times A \times S$ a set of *edges* or *transitions*,
- and $I \in S$ the *root* or *initial* state.

Let $\mathbb{G}(A)$ be the domain of process graphs labelled over A.

One writes $r \xrightarrow{a} s$ for $(r,a,s) \in \rightarrow$. Virtually all so-called *interleaving models* for the representation of processes are isomorphic to graph models. For instance, the *failure sets* that represent expressions in the process description language CSP [5] can easily be encoded as equivalence classes of graphs, under a suitable equivalence. In [3] the language ACP is equipped with a process graph semantics, and the semantics of CCS, SCCS and MEIJE given in [21, 20, 1, 24] are operational ones, which, as I will show below, induce process graph semantics. In the languages $\mathscr{L}$ studied in this paper, the domain $\mathbb{D}$ in which $\mathscr{L}$-expressions are interpreted will be $\mathbb{G}(A)$ for some set of actions A, or a subclass of $\mathbb{G}(A)$.

Usually the parts of a graph that cannot be reached from the initial state by following a finite path of transitions are considered meaningless for the description of processes. This means that one is only interested in process graphs as a model of system behaviour up to some equivalence, and this equivalence identifies at least graphs with the same reachable parts.

Definition 7 The *reachable part* of a process graph $(S,A,\rightarrow,I)$ is the process graph $(S',A,\rightarrow',I)$ where

- $S' \subseteq S$ is the smallest set such that (1) $I \in S'$ and (2) if $r \in S'$ and $r \xrightarrow{a} s$ then $s \in S'$,
- and $\rightarrow'$ is the restriction of $\rightarrow$ to $S' \times A \times S'$.

8 A Process Graph Semantics of Transition System Specifications

This section proposes a process graph semantics of TSSs. For each TSS $P = (\Sigma, A, R)$ it defines an interpretation $[\![\_]\!]_P : \mathbb{T}^r(\Sigma) \rightarrow (\mathbb{G}(A)^{\mathit{Var}} \rightarrow \mathbb{G}(A))$, thus taking $\mathbb{D}$ to be $\mathbb{G}(A)$.

Definition 8 (*Interpreting the closed expressions in a TSS as process graphs*). Let $P = (\Sigma, A, R)$ be a TSS and $p \in \mathrm{T}^r(\Sigma)$. Then $[\![p]\!]_P^{\emptyset} \in \mathbb{G}(A)$ is the reachable part of the process graph $(\mathrm{T}^r(\Sigma), A, \rightarrow, p)$ with $\rightarrow$ the set of transitions provable from P.

To define an interpretation $[\![\_]\!]_P : \mathbb{T}^r(\Sigma) \rightarrow (\mathbb{G}(A)^{\mathit{Var}} \rightarrow \mathbb{G}(A))$ of the open Σ-terms in $\mathbb{G}(A)$ I would like to simply add to the signature Σ a constant G for each process graph $G \in \mathbb{G}(A)$. However, $\mathbb{G}(A)$ is a proper class, whereas a signature needs to be a set. For this reason I work with appropriate subsets $\mathbb{G}^*$ of $\mathbb{G}(A)$ instead of with $\mathbb{G}(A)$ itself. I will discuss the selection of $\mathbb{G}^*$ later, but one requirement will be

$$\text{if } (S, A, \rightarrow, r) \in \mathbb{G}^* \text{ and } r \xrightarrow{a} s \text{ then also } (S, A, \rightarrow, s) \in \mathbb{G}^* \qquad (\textit{transition closure}).$$

Define the transition relation $\rightarrow_{\mathbb{G}^*} \subseteq \mathbb{G}^* \times A \times \mathbb{G}^*$ by $G \xrightarrow{a}_{\mathbb{G}^*} G'$ iff (i) $G = (S, A, \rightarrow, r)$, (ii) there is a transition $(r, a, s) \in \rightarrow$, and (iii) $G' = (S, A, \rightarrow, s)$ is the same graph but with s as initial state.

Now consider a term $t \in \mathbb{T}^r(\Sigma)$ and a valuation $\rho : \mathit{Var} \rightarrow \mathbb{G}(A)$. In order to define $[\![t]\!]_P(\rho)$, make sure that $\mathbb{G}^*$ *supports* (t, ρ), meaning that it contains $\rho(x)$ for any variable $x \in \mathit{Var}$ occurring free in t. Let $P + \mathbb{G}^*$ be the TSS P to which all graphs $G \in \mathbb{G}^*$ have been added as constants, and all transitions in $\rightarrow_{\mathbb{G}^*}$ as transition rules without premises. As the valuation ρ now also is a substitution, $t[\rho]$ is a closed term in the TSS $P + \mathbb{G}^*$. Define $[\![t]\!]_P^{\mathbb{G}^*}(\rho)$ to be $[\![t[\rho]]\!]_{P+\mathbb{G}^*}^{\emptyset} \in \mathbb{G}(A)$: the interpretation according to Definition 8 of the closed term $t[\rho]$ from the TSS $P + \mathbb{G}^*$.

Definition 9 (*The simple process graph semantics of TSSs*). A TSS P *manifestly induces a process graph semantics* iff, for any term t and valuation ρ, the interpretation $[\![t]\!]_P^{\mathbb{G}^*}(\rho)$ is independent of the choice of $\mathbb{G}^*$, as long as $\mathbb{G}^*$ is transition-closed and supports (t, ρ). In that case $[\![t]\!]_P(\rho)$ is defined to be $[\![t]\!]_P^{\mathbb{G}^*}(\rho)$.

It is possible to enlarge the class of TSSs that induce a process graph semantics a little bit:

Definition 10 (*The process graph semantics of TSSs*). Call a choice of $\mathbb{G}^*$ *adequate* for (the interpretation of) $[\![t]\!]_P(\rho)$ if $\mathbb{G}^*$ is transition-closed and supports (t, ρ), and $[\![t]\!]_P^{\mathbb{G}'}(\rho) = [\![t]\!]_P^{\mathbb{G}^*}(\rho)$ for any transition-closed superset $\mathbb{G}'$ of $\mathbb{G}^*$. Now P *induces a process graph semantics* iff, for any term t and valuation ρ, an adequate choice $\mathbb{G}^*$ for $[\![t]\!]_P(\rho)$ exists. In that case $[\![t]\!]_P(\rho)$ is defined to be $[\![t]\!]_P^{\mathbb{G}^*}(\rho)$ for any adequate choice of $\mathbb{G}^*$.

Example 1 (continued) Reconsider the operators f and id from Example 1. To judge whether they are essentially different one compares the open terms $f(x)$ and $id(x)$. Their meanings are values that depend on the choice of a valuation ρ, mapping variables to values. In fact they depend on the value $\rho(x)$ only.

Under the closed term interpretation of the TSS P of Example 1, $\rho(x)$ is a closed term in the language; it cannot have an outgoing τ-transition. Thus $[\![f(x)]\!]_P(\rho) \leftrightarrow [\![x]\!]_P(\rho) \leftrightarrow [\![id(x)]\!]_P(\rho)$ for any such ρ, so $[\![f(x)]\!]_P \leftrightarrow [\![x]\!]_P \leftrightarrow [\![id(x)]\!]_P$, i.e., f and id are strongly bisimilar.

However, under the process graph interpretation, $\rho(x)$ is a process graph, and one may take $\rho(x)$ to be $\rightarrow\circ \xrightarrow{\tau} \circ \xrightarrow{c} \circ$, where the short arrow indicates the initial state. With this valuation, the process graph $[\![id(x)]\!]_P(\rho)$ is isomorphic to $\rho(x)$, whereas $[\![f(x)]\!]_P$ has no outgoing transitions. So $[\![f(x)]\!]_P(\rho) \not\leftrightarrow [\![x]\!]_P(\rho) \leftrightarrow [\![id(x)]\!]_P(\rho)$ and consequently $[\![f(x)]\!]_P \not\leftrightarrow [\![x]\!]_P \leftrightarrow [\![id(x)]\!]_P$, i.e., f and id are not bisimilar.

The smallest set of process graphs $\mathbb{G}^*$ that is adequate for the interpretation of $[\![f(x)]\!]_P(\rho)$ and $[\![id(x)]\!]_P(\rho)$ is $\left\{ \overset{\downarrow}{\circ} \xrightarrow{\tau} \circ \xrightarrow{c} \circ \;,\; \circ \xrightarrow{\tau} \overset{\downarrow}{\circ} \xrightarrow{c} \circ \;,\; \circ \xrightarrow{\tau} \circ \xrightarrow{c} \overset{\downarrow}{\circ} \right\}$.

Example 2 A TSS with a rule $c \xrightarrow{a} x$ (without premises) does not induce a process graph semantics. Namely, no matter which set $\mathbb{G}^*$ one takes, there is always a larger set $\mathbb{G}'$ and a process graph $G \in \mathbb{G}' \setminus \mathbb{G}^*$. Thus, the process graph $[\![c]\!]_P^{\mathbb{G}'}$ has the transition $c \xrightarrow{a} G$ but $[\![c]\!]_P^{\mathbb{G}^*}$ does not. Hence $\mathbb{G}^*$ is not adequate. Consequently, the TSS does not induce a process graph semantics.

Example 3 Let P be the TSS with constants c and 0 and as only rule

$$\frac{x \xrightarrow{a} y \quad z \xrightarrow{b} y}{c \xrightarrow{a} 0} .$$

Then $[\![c]\!]_P^{\mathbb{G}^*}(\rho)$ is independent of ρ, since c is a closed term. In any adequate choice of $\mathbb{G}^*$ there is a graph in which an a-transition and a b-transition end in a common state, and using such a choice one finds that $[\![c]\!]_P^{\mathbb{G}^*}$ has the transition $c \xrightarrow{a} 0$. However, when taking $\mathbb{G}^*$ to be a set of trees, $[\![c]\!]_P^{\mathbb{G}^*}$ has no transitions.

P induces a process graph semantics according to Definition 10, but not according to Definition 9.

If we stick with Definition 9, $[\![t]\!]_P = [\![t]\!]_P^{\emptyset}$ for closed terms t.

Definition 11 The *rule-bound variables* of a transition rule $\frac{H}{t \xrightarrow{a} t'}$ form the smallest set B such that

- $var(t) \subseteq B$, and
- if $u \xrightarrow{b} u'$ is a premise in H and $var(u) \subseteq B$ then $var(u') \subseteq B$.

A TSS is called *pure* if all variables occurring free in one of its rules are rule-bound in that rule.

This concept of a pure TSS generalises the one from [16], and coincides with it for TSSs in the tyft/tyxt format studied in [16]. The TSSs of all common process algebras are pure. So is the TSS of Example 1, but the ones of Examples 2 and 3 are not.

Proposition 5 Any pure TSS manifestly induces a process graph semantics.

Proof: For a given term $t \in \mathbb{T}^r(\Sigma)$ and valuation $\rho : Var \to \mathbb{G}(A)$, let $\mathbb{G}_0^*$ be the smallest set of process graphs that is transition-closed and supports (t,ρ). Then for any $\mathbb{G}^* \supseteq \mathbb{G}_0^*$ and any transition $t[\rho] \xrightarrow{a} u$ provable from $P + \mathbb{G}^*$, any term v occurring in a proof of this transition, including the target u, has the form $t'[\rho]$ with $t' \in \mathbb{T}^r(\Sigma)$, such that $\rho(x) \in \mathbb{G}_0^*$ for any $x \in var(t)$. This follows by a fairly straightforward induction on the size of proofs, with a nested induction on the derivation of rule-boundedness. As a consequence, the process graph $[\![t[\rho]]\!]_{P+\mathbb{G}^*}^{\emptyset}$ does not depend in any way on $\mathbb{G}^* \setminus \mathbb{G}_0^*$. □

Remark One may wonder whether the above treatment can be simplified by skipping, in Definition 8, "the reachable part of". The answer is negative, for in that case $[\![t]\!]_P^{\mathbb{G}^*}(\rho)$ would never be independent of the choice of $\mathbb{G}^*$, because all $G \in \mathbb{G}^*$ would occur as (unreachable) states in the process graph $[\![t]\!]_P^{\mathbb{G}^*}(\rho)$.

Summary In this section, terms in a TSS are interpreted in the domain of process graphs as follows. Let $P = (\Sigma, A, R)$ be a TSS and $t \in \mathbb{T}^r(\Sigma)$ a term. The meaning $[\![t]\!]_P : \mathbb{G}(A)^{Var} \to \mathbb{G}(A)$ of t is given by $[\![t]\!]_P(\rho) := [\![t[\rho]]\!]_{P+\mathbb{G}^*}^{\emptyset}$, with $\mathbb{G}^*$ adequate for $[\![t]\!]_P(\rho)$. Here $\mathbb{G}^*$ is transition-closed and supports (t,ρ); it is *adequate* if further increasing this set does not alter the definition of $[\![t]\!]_P(\rho)$. If no adequate $\mathbb{G}^*$ exists, $[\![t]\!]_P(\rho)$ remains undefined, and P does not induce a process graph semantics. If P is pure, any transition-closed set $\mathbb{G}^* \subseteq \mathbb{G}(A)$ supporting (t,ρ) is adequate.

9 Lifting Semantic Equivalences to Open Terms

The following definition shows how any equivalence relation $\sim$ defined on a domain $\mathbb{D}$ in which a language $\mathscr{L}$ is interpreted, lifts to the open terms of $\mathscr{L}$.

Definition 12 (*Lifting equivalences to open terms*). For a language $\mathscr{L}$, given as an annotated signature Σ and an interpretation $[\![\_]\!] : \mathbb{T}^r(\Sigma) \to (\mathbb{D}^{Var} \to \mathbb{D})$, and an equivalence relation $\sim$ on $\mathbb{D}$, write $t \sim_{\mathscr{L}} u$ for $t,u \in \mathbb{T}^r(\Sigma)$ iff $[\![t]\!](\rho) \sim [\![u]\!](\rho)$ for all valuations $\rho : Var \to \mathbb{D}$.

This definition can be applied to any language $\mathscr{L}$ given by a TSS $P = (\Sigma, A, R)$. In this case $\sim$ must be defined on $\mathbb{G}(A)$. Write $\sim_P^{\mathsf{pg}}$ for $\sim_{\mathscr{L}}$ as defined above when taking as interpretation the process graph semantics $[\![\_]\!] : \mathbb{T}^r(\Sigma) \to (\mathbb{G}(A)^{Var} \to \mathbb{G}(A))$ of $\mathscr{L}$.

An equivalence $\sim$ on $\mathbb{G}(A)$ also lifts to the closed terms $\mathrm{T}^r(\Sigma)$ of $\mathscr{L}$. Namely, let $(\mathrm{T}^r(\Sigma), A, \to)$ be the LTS specified by P as defined in Section 5. Then $(\mathrm{T}^r(\Sigma), A, \to, p) \in \mathbb{G}(A)$ is a process graph for any $p \in \mathrm{T}^r(\Sigma)$. Now write $p \sim q$, for $p,q \in \mathrm{T}^r(\Sigma)$, whenever $(\mathrm{T}^r(\Sigma), A, \to, p) \sim (\mathrm{T}^r(\Sigma), A, \to, p)$.

Using this, Definition 12 can also be instantiated by taking as interpretation the closed-term semantics $[\![\_]\!] : \mathbb{T}^r(\Sigma) \to (\mathrm{T}^r(\Sigma)^{Var} \to \mathrm{T}^r(\Sigma))$ of $\mathscr{L}$, as defined in Section 6. Write $\sim_P^{\mathsf{ci}}$ for $\sim_{\mathscr{L}}$ defined thusly. So

$$t \sim_P^{\mathsf{ci}} u \text{ iff } t[\sigma] \sim u[\sigma] \text{ for any closed substitution } \sigma,$$

i.e., two open terms are related by $\sim_P^{\mathsf{ci}}$ if all of their *closed instantiations* are related by $\sim$.

Having lifted semantic equivalences $\sim$ from process graphs to open terms in two ways, one wonders how the resulting equivalences compare. Instantiating $\sim$ with strong bisimilarity, $\leftrightarrow$, Example 1 shows that $f(x) \leftrightarrow_P^{\mathsf{ci}} id(x)$ yet $f(x) \not\leftrightarrow_P^{\mathsf{pg}} id(x)$. For the other direction consider the TSS with constants 0, c and d, alphabet $\{a,b\}$, and the rules

$$d \xrightarrow{a} 0 \qquad \frac{x \xrightarrow{b} y}{c \xrightarrow{a} 0}\,.$$

Under the closed-term interpretation, no b-transition from any term can be derived, so $0 \leftrightarrow^{\mathsf{ci}} c \not\leftrightarrow^{\mathsf{ci}} d$. Yet under the process graph interpretation, since there exists some graph that can do a b-transition, one has $c \xrightarrow{a} 0$, and obtains $0 \not\leftrightarrow^{\mathsf{pg}} c \leftrightarrow^{\mathsf{pg}} d$. So in general $\leftrightarrow^{\mathsf{ci}}$ and $\leftrightarrow^{\mathsf{pg}}$ are incomparable.

The above TSS is not pure; the variable x is not rule-bound. For pure TSSs no such example exists.

Theorem 1 Let $P = (\Sigma, A, R)$ be a pure TSS and $\approx$ an equivalence on $\mathbb{G}(A)$ that relates each process graph with its reachable part. Moreover, let $\sim \subseteq \approx$ be a possibly finer, or more discriminating, equivalence that satisfies requirements (1)–(4) of Section 3.1. Then $t \approx_P^{\mathsf{pg}} u$ implies $t \approx_P^{\mathsf{ci}} u$.

Proof: Suppose $t \approx_P^{\mathsf{pg}} u$, and let $\sigma : Var \to \mathrm{T}^r(\Sigma)$ be a closed substitution. It suffices to establish that $t[\sigma] \approx u[\sigma]$. Let $[\![\sigma]\!]_P : Var \to \mathbb{G}(A)$ be the valuation defined by $[\![\sigma]\!]_P(X) := [\![\sigma(X)]\!]_P$. This is the definition of $[\![\sigma]\!]_P$ from Section 3.3, specialised to closed substitutions. Here $[\![q]\!]_P$, for $q \in \mathrm{T}^r(\Sigma)$, is the process graph semantics of q as defined in Section 8, so $[\![q]\!]_P = [\![q]\!]^{\emptyset}_{P+\mathbb{G}^*}$ for an adequate choice of $\mathbb{G}^*$. Since P is pure and q closed, the empty set of process graphs is adequate by Proposition 5. By Definition 8, $[\![q]\!]^{\emptyset}_P$ is the reachable part of the process graph $(\mathrm{T}^r(\Sigma), A, \to, p)$, so $[\![q]\!]^{\emptyset}_P \approx (\mathrm{T}^r(\Sigma), A, \to, p)$. Since $t \approx_P^{\mathsf{pg}} u$, one has $[\![t]\!]_p([\![\sigma]\!]_P) \approx [\![u]\!]_p([\![\sigma]\!]_P)$ by Definition 12. Moreover, $[\![t]\!]_p([\![\sigma]\!]_P) \sim [\![t[\sigma]]\!]_P$ by Proposition 3. Hence

$$[\![t[\sigma]]\!]^{\emptyset}_P = [\![t[\sigma]]\!]_P \approx [\![t]\!]_p([\![\sigma]\!]_P) \approx [\![u]\!]_p([\![\sigma]\!]_P) \approx [\![u[\sigma]]\!]_P = [\![u[\sigma]]\!]^{\emptyset}_P$$

and thus $(\mathrm{T}^r(\Sigma), A, \to, t[\sigma]) \approx [\![t[\sigma]]\!]^{\emptyset}_{P+\mathbb{G}^*} \approx [\![u[\sigma]]\!]^{\emptyset}_P \approx (\mathrm{T}^r(\Sigma), A, \to, u[\sigma])$, i.e., $t[\sigma] \approx u[\sigma]$. □

So, under the conditions of Theorem 1, $\approx_P^{\mathsf{pg}}$ is a finer, or more discriminating, equivalence than $\approx_P^{\mathsf{ci}}$.

10 Congruence Properties

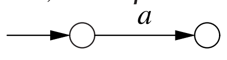
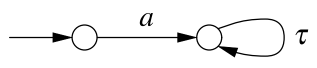
Whether a semantic equivalence $\sim$ is a congruence (cf. (8) in Section 4) may depend on whether the closed-term or the process graph semantics is chosen. The following example illustrates this for a practical process algebra.

Example 4 Consider the TSS with constants 1 and α for $\alpha \in Act = A \uplus \{\tau\}$ and binary operators $+$ and ;, denoting choice and sequencing in a process algebra, with the following transition rules:

$$\alpha \xrightarrow{\alpha} 1 \qquad \frac{x \xrightarrow{\alpha} x'}{x+y \xrightarrow{\alpha} x'} \qquad \frac{y \xrightarrow{\alpha} y'}{x+y \xrightarrow{\alpha} y'} \qquad \frac{x \xrightarrow{\alpha} x'}{x;y \xrightarrow{\alpha} x';y} \qquad \frac{x \stackrel{\alpha}{\not\longrightarrow} \text{ for all } \alpha \in Act \quad y \xrightarrow{\beta} y'}{x;y \xrightarrow{\beta} y'}$$

The process 1, like 0 in CCS, has no outgoing transitions, meaning that it performs no actions. The sequencing operator performs all actions its first argument can do, until its first argument can perform no further actions; then it continues with its second argument. I employ no recursion here.

As equivalence relation $\sim$ I take weak bisimulation equivalence, $\underline{\leftrightarrow}_w$, as defined in [21, 12]. For the term t from (8) take $x;b$. Let ρ and ν be valuations with

$\rho(x) =$  and $\nu(x) =$  .

Then $\rho(x) \,\underline{\leftrightarrow}_w\, \nu(x)$, so that I may assume $\rho \,\underline{\leftrightarrow}_w\, \nu$. Now the term t performs the sequential composition of the process filled in for x with the process doing a single b action. One has $[\![x;b]\!](\rho) \not\underline{\leftrightarrow}_w [\![x;b]\!](\nu)$ because only the first of these processes can ever perform the b. Thus, when using the process graph semantics of this TSS, $\underline{\leftrightarrow}_w$ fails to be a congruence for the language specified.

However, when taking the closed-term semantics, all processes that my be filled in for x are terms in the given language and thus must terminate after performing finitely many transitions. In this setting $\underline{\leftrightarrow}_w$ is actually a congruence. However, it stops being a congruence when recursion is added to the language.

11 Relating Sanity Requirements for the Two Semantics of TSSs

Given an equivalence relation $\sim$ on $\mathbb{G}(A)$, let $\sim_P^{\mathbf{cl}}$ be the equivalence relation on the set $\mathrm{T}^r(\Sigma)$ of closed terms of a TSS $P = (\Sigma, A, R)$ defined by $p \sim_P^{\mathbf{cl}} q$ iff $[\![\,p\,]\!]_P^{\emptyset} \sim [\![\,q\,]\!]_P^{\emptyset}$ (cf. Definition 8). In case $\sim$ relates each process graph with its reachable part, the equivalence $\sim_P^{\mathbf{cl}}$ coincides with $\sim_P^{\mathbf{ci}}$, as defined in Section 9.

Observation 1 If P is pure and $p,q \in \mathrm{T}^r(\sigma)$, then $p \sim_P^{\mathbf{cl}} q$ iff $p \sim_P^{\mathbf{pg}} q$.

Theorem 2 Let P be a TSS that induces a process graph semantics $[\![\_]\!]_P$ and let $\sim$ be an equivalence relation on $\mathbb{G}(A)$. Then $[\![\_]\!]_P$ satisfies the sanity requirements (2)–(5) of Section 3 up to $\sim$ if the closed-term semantics of $P+\mathbb{G}^*$ satisfies these requirements up to $\sim_{P+\mathbb{G}^*}^{\mathbf{cl}}$ for any choice of $\mathbb{G}^*$.

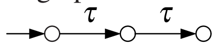
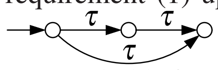
Proof: Let $\rho, \nu : Var \to \mathbb{G}(A)$, $(f,n) \in \Sigma$ and $t_i, u_i \in \mathbb{T}^r(\Sigma)$, such that $[\![t_i]\!]_P(\rho) \sim [\![u_i]\!]_P(\nu)$ for $i = 1,\dots,n$. Let the set $\mathbb{G}^* \subseteq \mathbb{G}(A)$ be adequate for the definition of $[\![t_i]\!]_P(\rho)$ and $[\![u_i]\!]_P(\nu)$ for $i = 1,\dots,n$ as well as for $[\![f(t_1,\dots,t_n)]\!]_P(\rho)$ and $[\![f(u_1,\dots,u_n)]\!]_P(\nu)$. Then $[\![t_i[\rho]]\!]_{P+\mathbb{G}^*}^{\emptyset} \sim [\![u_i[\nu]]\!]_{P+\mathbb{G}^*}^{\emptyset}$, i.e., $t_i[\rho] \sim_{P+\mathbb{G}^*}^{\mathbf{cl}} u_i[\nu]$, for $i = 1,\dots,n$. So $f(t_1[\rho],\dots,t_n[\rho]) \sim_{P+\mathbb{G}^*}^{\mathbf{cl}} f(u_1[\nu],\dots,u_n[\nu])$ by Requirement (2′) for the closed-term semantics of $P+\mathbb{G}^*$; that is, $[\![f(t_1[\rho],\dots,t_n[\rho])]\!]_{P+\mathbb{G}^*}^{\emptyset} \sim [\![f(u_1[\nu],\dots,u_n[\nu])]\!]_{P+\mathbb{G}^*}^{\emptyset}$, or $[\![f(t_1,\dots,t_n)]\!]_P(\rho) \sim [\![f(u_1,\dots,u_n)]\!]_P(\nu)$.

Let $\rho, \nu : Var \to \mathbb{G}(A)$ and $S, S' : W \to \mathbb{T}^r(\Sigma)$ with $X \in W \subseteq Var$, such that $[\![S_Y]\!]_P(\rho\backslash W) \sim [\![S'_Y]\!]_P(\nu\backslash W)$ for $Y \in W$. The latter means that $[\![S_Y]\!]_P(\rho\backslash W)(\xi) \sim [\![S'_Y]\!]_P(\nu\backslash W)(\xi)$ for all $Y \in W$ and $\xi : W \to \mathbb{G}(A)$. Let $\mathbb{G}^* \subseteq \mathbb{G}(A)$ be adequate for $[\![S_Y]\!]_P(\xi \cup \rho\backslash W)$ and $[\![S'_Y]\!]_P(\xi \cup \nu\backslash W)$ for all $Y \in W$ and $\xi : W \to \mathbb{G}(A)$, as well as for $[\![\langle X|S\rangle]\!]_P(\rho)$ and $[\![\langle X|S'\rangle]\!]_P(\nu)$. Then $S_Y[\rho\backslash W][\xi] \sim^{\mathbf{cl}}_{P+\mathbb{G}^*} S'_Y[\nu\backslash W][\xi]$ for all $Y \in W$ and $\xi : W \to \mathbb{G}(A)$. So $\langle X|S[\rho\backslash W]\rangle \sim^{\mathbf{cl}}_{P+\mathbb{G}^*} \langle X|S'[\nu\backslash W]\rangle$ by Requirement (3′) for the closed-term semantics of $P+\mathbb{G}^*$, and $[\![\langle X|S\rangle]\!]_P(\rho) = [\![\langle X|S\rangle[\rho]]\!]^{\emptyset}_{P+\mathbb{G}^*} = [\![\langle X|S[\rho\backslash W]\rangle]\!]^{\emptyset}_{P+\mathbb{G}^*} \sim [\![\langle X|S'[\nu\backslash W]\rangle]\!]^{\emptyset}_{P+\mathbb{G}^*} = [\![\langle X|S'\rangle]\!]_P(\nu)$.

Let $S : V_S \to \mathbb{T}^r(\Sigma)$ with $X \in V_S \subseteq \mathit{Var}$, and let $\gamma : V_S \to \mathit{Var}$ be an injective substitution such that the range of γ contains no variables occurring free in $\langle S_Y|S\rangle$ for some $Y \in V_S$. Take $\rho : \mathit{Var} \to \mathbb{G}(A)$. Let $\mathbb{G}^*$ be adequate for $[\![\langle X|S\rangle]\!]_P(\rho)$ and $[\![\langle \gamma(X)|S[\gamma]\rangle]\!]_P(\rho)$. By Requirement (4′) for the closed-term semantics of $P+\mathbb{G}^*$ one has $\langle\gamma(X)|S[\gamma]\rangle[\rho] \sim^{\mathbf{cl}}_{P+\mathbb{G}^*} \langle X|S\rangle[\rho]$, using that $\langle X|S\rangle[\rho] = \langle X|S[\rho\backslash V_S]\rangle$ and $\langle\gamma(X)|S[\gamma]\rangle[\rho] = \langle\gamma(X)|S[\gamma][\rho\backslash\gamma(V_S)]\rangle = \langle\gamma(X)|S[\rho\backslash V_S][\gamma]\rangle$. So $[\![\langle\gamma(X)|S[\gamma]\rangle]\!]_P(\rho) \sim [\![\langle X|S\rangle]\!]_P(\rho)$.

Let $S : V_S \to \mathbb{T}^r(\Sigma)$ with $X \in V_S \subseteq \mathit{Var}$. Take $\rho : \mathit{Var} \to \mathbb{G}(A)$. Let $\mathbb{G}^*$ be adequate for $[\![\langle X|S\rangle]\!]_P(\rho)$ and $[\![S_X|S]\!]_P(\rho)$. Then $\langle X|S[\rho\backslash V_S]\rangle \sim^{\mathbf{cl}}_{P+\mathbb{G}^*} \langle S_X[\rho\backslash V_S]|S[\rho\backslash V_S]\rangle$ by Requirement (5′) for the closed-term semantics of $P+\mathbb{G}^*$. Hence $[\![\langle X|S\rangle]\!]_P(\rho) = [\![\langle X|S\rangle[\rho]]\!]^{\emptyset}_{P+\mathbb{G}^*} = [\![\langle X|S[\rho\backslash V_S]\rangle]\!]^{\emptyset}_{P+\mathbb{G}^*} \sim [\![\langle S_X[\rho\backslash V_S]|S[\rho\backslash V_S]\rangle]\!]^{\emptyset}_{P+\mathbb{G}^*}$ $= [\![\langle S_X|S\rangle[\rho]]\!]^{\emptyset}_{P+\mathbb{G}^*} = [\![S_X|S]\!]_P(\rho)$. □

Theorem 2 does not extend to sanity requirement (1). In fact, this requirement always holds for the closed-term interpretation of a TSS, yet it does not always hold for the process graph interpretation:

Example 5 Let P be a TSS with the single rule $\frac{x \xrightarrow{\tau} y,\ y \xrightarrow{\tau} z}{x \xrightarrow{\tau} z}$. Then the process graph semantics of P fails to satisfy sanity requirement (1) up to $\underline{\leftrightarrow}$. Namely, if $\rho(x)$ is a graph , then $[\![x]\!]_P(\rho)$ is the graph , and the two graphs are not (strongly) bisimulation equivalent.

Nevertheless, requirement (1) holds for almost all process algebras found in the literature:

Proposition 6 Let P be a TSS that has no rule with a variable as the left-hand side of the conclusion. The process-graph interpretation of P always satisfies requirement (1) of Section 3 up to $\underline{\leftrightarrow}$.

Proof: For $x \in \mathit{Var}$ and $\rho : \mathit{Var} \to \mathbb{G}(A)$ let $\mathbb{G}^* \subseteq \mathbb{G}(A)$ be adequate for $[\![x]\!]_P(\rho)$ and let $\rho(x) \in \mathbb{G}^*$ be the graph $g = (S, \to, I)$. For each state $s \in S$ there is a graph $g_s := (S, \to, s)$ in $\mathbb{G}^*$. Now $[\![x]\!]_P(\rho)$ is the reachable part of the graph $(G, \to_G, g)$, where $G = \{g_s \mid s \in S\}$ and $\to_G = \{(g_s, a, g_t) \mid (s, a, t) \in \to\}$. The relation $\mathscr{R}$ given by $g_s \mathrel{\mathscr{R}} s$ for all states $s \in S$ reachable from I clearly is a bisimulation. Therefore $[\![x]\!]_P(\rho) \mathrel{\underline{\leftrightarrow}} \rho(x)$. □

12 Preservation of Relative Expressiveness under Conservative Extensions

Definition 13 If $P = (\Sigma_P, A, R_P)$ and $Q = (\Sigma_Q, A, R_Q)$ are TSSs with Σ_P and Σ_Q disjoint then $P + Q$ denotes the TSS $(\Sigma_P \cup \Sigma_Q, A, R_P \cup R_Q)$.

Let P_0 be the TSS of Example 1, but without the operator f. Let P_f be the part of the TSS of Example 1 that only contains the operator f, so that the entire TSS of Example 1 is $P_0 + P_f$. This TSS does not feature recursion.

A *translation* between two languages with signatures Σ and Σ' is a mapping $\mathscr{T} : \mathbb{T}^r(\Sigma) \to \mathbb{T}^r(\Sigma')$. Consider the translation $\mathscr{T}_{id}$ from the language specified by P_0 to the language specified by $P_0 + P_f$, given by $\mathscr{T}_{id}(t) = t$ for all $t \in \Sigma_{P_0}$. Also consider the translation $\mathscr{T}_{op}$ in the opposite direction, given by $\mathscr{T}_{op}(a.t) := a.\mathscr{T}_{op}(t)$, $\mathscr{T}_{op}(id(t)) := id(\mathscr{T}_{op}(t))$ and $\mathscr{T}_{op}(f(t)) := id(\mathscr{T}_{op}(t))$ for all $t \in \mathbb{T}(\Sigma_{P_o+P_f})$.

In e.g. [14] a concept of expressiveness of specification languages is studied such that language $\mathscr{L}'$ is at least as expressive as language $\mathscr{L}$ up to a semantic equivalence relation $\sim$ iff there exist a translation from $\mathscr{L}$ into $\mathscr{L}'$ that is valid up to $\sim$. It is not important here to state the precise definition of validity; it suffices to point out that $\mathscr{T}_{op}$ is valid up to $\underline{\leftrightarrow}$ iff one has $f(x) \mathrel{\underline{\leftrightarrow}} id(x)$. Thus, applying Definition 12, it is valid when employing the closed-term semantics, but not when employing the process graph semantics.

The transition $\mathscr{T}_{id}$ on the other hand is valid up to $\leftrightarrow$ regardless which of the two interpretations one picks. So under the closed-term interpretation the two languages are equally expressive, whereas under the process graph semantics the language given by $P_0 + P_f$ is more expressive than the one given by P_0.

For TSSs P and Q, write $P \preceq Q$ if the language specified by Q is at least as expressive as the one specified by P. An intuitively plausible theorem is that

$$P_1 \preceq P_2 \quad \text{implies} \quad P_1 + Q \preceq P_2 + Q, \tag{9}$$

at least under some mild conditions on the TSSs P_1, P_2 and Q, for instance that they are pure and fit the *tyft* format defined in [16].[3] This theorem fails when employing the closed-term semantics of TSSs: take P_1 to be $P_0 + P_f$, P_2 to be P_0, with $\mathscr{T}_{op}$ being the witness for $P_1 \preceq P_2$, and Q to be the TSS with as single operator $\tau.\_$ and as only transition rule $\tau.x \xrightarrow{\tau} x$. For the operator f in the TSS $P_0 + P_f + Q$ drops τ-transitions, and has no counterpart in the TSS $P_0 + Q$.

This problem is fixed when employing the process graph semantics. Once the omitted definition of validity is supplied [14], the proof of (9) is entirely straightforward.

13 Related Work

Dissatisfaction with the traditional closed-term interpretation of TSSs occurred earlier in [17, 18, 8, 23] and [2]. However, rather than adapting the interpretation of TSSs, as in the present paper, these papers abandon the notion of a TSS in favour of different frameworks of system specification that are arguably more suitable for giving meaning to open terms. Larsen and Liu [17] use *context systems*. The CCS transition rule

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

for instance takes in a context system the shape $x|y \xrightarrow[a,\bar{a}]{\tau} x'|y'$, or rather, suppressing the redundant variable names, $| \xrightarrow[a,\bar{a}]{\tau} |$. It says that the operator $|$ can perform a τ-transition, provided its first argument does an a-transition, and its second argument an $\bar{a}$. The context systems of [17] form the counterpart of TSSs in the De Simone format [24]. The model is generalised by Lynch & Vaandrager [18] to *action transducers*, by Gadducci & U. Montanari [8] to the *tile model*, and by Rensink [23] to *conditional transition systems*. The latter two proposals are further generalised to *symbolic transition systems* by Baldan, Bracciali & Bruni [2].

One method to relate these models with TSSs under the closed-term and process graph interpretations is through notions of strong bisimilarity on open terms. This is a central theme in [23]. The most natural notion of bisimulation on the above models is *bisimulation under formal hypothesis*, $\leftrightarrow^{\mathsf{fh}}$. That name stems from De Simone [24], who defined the same concept in terms of TSSs. On the context systems sketched above it requires the usual transfer property for bisimulations for doubly labelled transitions such as $\xrightarrow[a,\bar{a}]{\tau}$. On TSSs, a bisimulation under formal hypothesis essentially is a symmetric relation $\mathscr{R}$ on open terms such that

$$\text{if } t\,\mathscr{R}\,u \text{ and } P \vdash \frac{\{x_i \xrightarrow{a_i} y_i \mid i \in I\}}{t \xrightarrow{a} t'} \text{ then } P \vdash \frac{\{x_i \xrightarrow{a_i} y_i \mid i \in I\}}{u \xrightarrow{a} u'} \text{ for an } u' \in \mathbb{T}^r(\Sigma_Q) \text{ with } t'\,\mathscr{R}\,u'.$$

Rensink [23] shows that $\leftrightarrow^{\mathsf{fh}}$ is strictly finer than $\leftrightarrow^{\mathsf{ci}}$.

Example 6 Let P be the TSS with inaction 0, action prefix, choice and intersection, specified by the following rules:

$$a.x \xrightarrow{a} x \qquad \frac{x \xrightarrow{a} x'}{x+y \xrightarrow{a} x'} \qquad \frac{y \xrightarrow{a} y'}{x+y \xrightarrow{a} y'} \qquad \frac{x \xrightarrow{a} x' \quad y \xrightarrow{a} y'}{x \cap y \xrightarrow{a} x' \cap y'}$$

[3] These mild conditions should ensure that $P_i + Q$ is a *conservative extension* of P_i, for $i = 1,2$, as defined in [16].

where a ranges over a set of actions A. Then $b.0+b.a.0+b.(x\cap a.0) \leftrightarrow^{\mathsf{ci}} b.0+b.a.0$, for no matter what one fills in for x, the process $x\cap a.0$ either cannot perform any transitions, or it can only do a single a. So the term $b.(x\cap a.0)$ behaves either like $b.0$ or like $b.a.0$. On the other hand, $b.0+b.a.0+b.(x\cap a.0) \not\leftrightarrow^{\mathsf{fh}} b.0+b.a.0$. Namely any bisimulation under formal hypothesis $\mathscr{R}$ relating $b.0+b.a.0+b.(x\cap a.0)$ with $b.0+b.a.0$ would also have to relate $x\cap a.0$ with either 0 or $a.0$. However, once this choice is made, substituting the wrong value for x shows that $\mathscr{R}$ relates two terms that are not equivalent.

Rensink also defines a *hypotheses-preserving bisimulation equivalence* $\leftrightarrow^{\mathsf{hp}}$ on open terms, which is situated strictly between $\leftrightarrow^{\mathsf{fh}}$ and $\leftrightarrow^{\mathsf{ci}}$. One has $b.0+b.a.0+b.(x\cap a.0) \not\leftrightarrow^{\mathsf{hp}} b.0+b.a.0$. His analysis can be reused to show that $\leftrightarrow^{\mathsf{hp}}$ is finer than $\leftrightarrow^{\mathsf{pg}}$. Note that $b.0+b.a.0+b.(x\cap a.0) \leftrightarrow^{\mathsf{pg}} b.0+b.a.0$, for the same reasons as in Example 6. Thus, under the conditions of Theorem 1, we arrive at a hierarchy

$$\leftrightarrow^{\mathsf{fh}} \quad\subseteq\quad \leftrightarrow^{\mathsf{hp}} \quad\subseteq\quad \leftrightarrow^{\mathsf{pg}} \quad\subseteq\quad \leftrightarrow^{\mathsf{ci}} \quad .$$

14 Concluding Remarks

This paper proposed a process graph semantics of TSSs as an alternative to the traditional closed-term semantics. It interprets an operator from the language as an operation on process graphs. Unlike the closed-term semantics, this interpretation is independent of the selection of processes that are expressible in the TSS as a whole. The intuitively plausible statement that an expressiveness inclusion between languages is preserved under a conservative extension of source and target language alike, fails for the closed-term semantics but holds for the proposed process graph semantics.

I reviewed five sanity requirements on languages equipped with a semantic equivalence relation $\sim$, and showed that four of them hold under the process semantics of a language if they hold under the closed-term semantics. Here I end with a few observations on when these requirements hold at all.

In [13], the *ntyft/ntyxt format with recursion* is introduced. It defines a wide class of TSSs, containing many known process algebras, including CCS, CSP, ACP, MEIJE and SCCS. It generalises the ntyft/ntyxt format of [15] by the addition of recursion as a separate language construct. The *tyft/tyxt format with recursion* is the same, but not allowing negative premises. [13] shows that all languages specified by a TSS in the ntyft/ntyxt format with recursion satisfy property (8) up to $\leftrightarrow$, saying that strong bisimilarity is a congruence. This is a stronger property than (2) up to $\leftrightarrow$, which thus also holds for the ntyft/ntyxt format. This was shown for the closed-term interpretation of TSSs. By Theorem 2 we now also have (2) up to $\leftrightarrow$ for the process graph semantics of pure TSSs in the ntyft/ntyxt format with recursion.

The same paper establishes that (3) holds up to $\leftrightarrow$ for the closed-term semantics of all TSSs in the tyft/tyxt format with recursion, thereby generalising a result from [23]. It thus also holds for the process graph semantics of all pure TSSs in the tyft/tyxt format with recursion.

It is not hard to show that also requirements (4) and (5) hold up to $\leftrightarrow$ for the closed-term interpretation of TSSs in the ntyft/ntyxt format with recursion, and thus for the process graph semantics of pure TSSs in the ntyft/ntyxt format with recursion.

Thanks to the equational nature of requirements (1), (4) and (5), once they hold up to $\leftrightarrow$, they surely hold up to any coarser equivalence. This covers most semantic equivalences found in the literature. The same cannot be said for requirements (2) and (3). These need to be reestablished for each semantic equivalence. There is a lot of work on congruence formats, ensuring (2) for a variety of semantic equivalence. See for instance [7], and references therein. Yet, besides [23] and [13] I know of no congruence formats targeting requirement (3).

Acknowledgement My thanks to the referees for their thorough proofreading and helpful suggestions.

References

[1] D. Austry & G. Boudol (1984): *Algèbre de processus et synchronisations*. Theoretical Computer Science 30(1), pp. 91–131, doi:10.1016/0304-3975(84)90067-7.

[2] P. Baldan, A. Bracciali & R. Bruni (2007): *A semantic framework for open processes*. Theoretical Computer Science 389(3), pp. 446–483, doi:10.1016/j.tcs.2007.09.004.

[3] J.A. Bergstra & J.W. Klop (1984): *The algebra of recursively defined processes and the algebra of regular processes*. In J. Paredaens, editor: Proceedings 11^{th} ICALP, Antwerpen, LNCS 172, Springer, pp. 82–94, doi:10.1007/3-540-13345-3_7.

[4] E. Bres, R.J. van Glabbeek & P. Höfner (2016): *A Timed Process Algebra for Wireless Networks with an Application in Routing*. Technical Report 9145, NICTA. Available at http://arxiv.org/abs/1606.03663. Extended abstract in P. Thiemann, editor: Programming Languages and Systems: Proceedings 25th European Symposium on Programming, ESOP'16; held as part of the European Joint Conferences on Theory and Practice of Software, ETAPS'16, LNCS 9632, Springer, 2016, pp. 95-122.

[5] S.D. Brookes, C.A.R. Hoare & A.W. Roscoe (1984): *A theory of communicating sequential processes*. Journal of the ACM 31(3), pp. 560–599, doi:10.1145/828.833.

[6] W.J. Fokkink (2000): *Introduction to Process Algebra*. Texts in Theoretical Computer Science, An EATCS Series, Springer, doi:10.1007/978-3-662-04293-9.

[7] W.J. Fokkink, R.J. van Glabbeek & B. Luttik (2017): *Divide and Congruence III: Stability & Divergence*. In R. Meyer & U. Nestmann, editors: Proceedings 28th International Conference on Concurrency Theory, CONCUR'17, Leibniz International Proceedings in Informatics (LIPIcs) 85, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, pp. 15:1–15:16, doi:10.4230/LIPIcs.CONCUR.2017.15. Available at http://theory.stanford.edu/~rvg/abstracts.html#124.

[8] F. Gadducci & U. Montanari (2000): *The tile model*. In G.D. Plotkin, C. Stirling & M. Tofte, editors: Proof, Language, and Interaction, Essays in Honour of Robin Milner, The MIT Press, pp. 133–166.

[9] R.J. van Glabbeek (1993): *Full abstraction in structural operational semantics (extended abstract)*. In M. Nivat, C. Rattray, T. Rus & G. Scollo, editors: Proceedings 3^{rd} International Conference on Algebraic Methodology and Software Technology, AMAST'93, Twente, The Netherlands, June l993, Workshops in Computing, Springer, pp. 77–84. Available at http://theory.stanford.edu/~rvg/abstracts.html#28.

[10] R.J. van Glabbeek (1994): *On the expressiveness of ACP (extended abstract)*. In A. Ponse, C. Verhoef & S.F.M. van Vlijmen, editors: Proceedings First Workshop on the Algebra of Communicating Processes, ACP94, Workshops in Computing, Springer, pp. 188–217, doi:10.1007/978-1-4471-2120-6_8. Available at http://theory.stanford.edu/~rvg/abstracts.html#31.

[11] R.J. van Glabbeek (2004): *The Meaning of Negative Premises in Transition System Specifications II*. Journal of Logic and Algebraic Programming 60–61, pp. 229–258, doi:10.1016/j.jlap.2004.03.007. Available at http://theory.stanford.edu/~rvg/abstracts.html#53.

[12] R.J. van Glabbeek (2011): *Bisimulation*. In D. Padua, editor: Encyclopedia of Parallel Computing, Springer, pp. 136–139, doi:10.1007/978-0-387-09766-4_149. Available at http://theory.stanford.edu/~rvg/abstracts.html#45.

[13] R.J. van Glabbeek (2017): *Lean and Full Congruence Formats for Recursion*. In: Proceedings 32^{nd} Annual ACM/IEEE Symposium on Logic in Computer Science, LICS'17, Reykjavik, Iceland, 2017, IEEE Computer Society Press, doi:10.1109/LICS.2017.8005142. Available at https://arxiv.org/abs/1704.03160.

[14] R.J. van Glabbeek (2018): *A Theory of Encodings and Expressiveness*. In C. Baier & U. Dal Lago, editors: Proceeding 21st International Conference on Foundations of Software Science and Computational Structures, FoSSaCS'18; held as part of the European Joint Conferences on Theory and Practice of Software, ETAPS'18, LNCS 10803, Springer, pp. 183–202, doi:10.1007/978-3-319-89366-2_10.

[15] J.F. Groote (1993): *Transition System Specifications with Negative Premises*. Theoretical Computer Science 118, pp. 263–299, doi:10.1016/0304-3975(93)90111-6.

[16] J.F. Groote & F.W. Vaandrager (1992): *Structured Operational Semantics and Bisimulation as a Congruence*. Information and Computation 100(2), pp. 202–260, doi:10.1016/0890-5401(92)90013-6.

[17] K.G. Larsen & X. Liu (1991): *Compositionality through an Operational Semantics of Contexts*. Journal of Logic and Computation 1(6), pp. 761–795, doi:10.1093/logcom/1.6.761.

[18] N.A. Lynch & F.W. Vaandrager (1996): *Action Transducers and Timed Automata*. Formal Aspects of Computing 8(5), pp. 499–538, doi:10.1007/BF01211907.

[19] Y.I. Manin (1977): *A Course in Mathematical Logic*. Graduate Texts in Mathematics 53, Springer, doi:10.1007/978-1-4757-4385-2.

[20] R. Milner (1983): *Calculi for synchrony and asynchrony*. Theoretical Computer Science 25(3), pp. 267–310, doi:10.1016/0304-3975(83)90114-7.

[21] R. Milner (1990): *Operational and algebraic semantics of concurrent processes*. In J. van Leeuwen, editor: Handbook of Theoretical Computer Science, chapter 19, Elsevier Science Publishers B.V. (North-Holland), pp. 1201–1242. Alternatively see *Communication and Concurrency*, Prentice-Hall, Englewood Cliffs, 1989, of which an earlier version appeared as *A Calculus of Communicating Systems*, LNCS 92, Springer, 1980.

[22] G.D. Plotkin (2004): *A Structural Approach to Operational Semantics*. Journal of Logic and Algebraic Programming 60–61, pp. 17–139, doi:10.1016/j.jlap.2004.05.001. Originally appeared in 1981.

[23] A. Rensink (2000): *Bisimilarity of Open Terms*. Information and Computation 156(1-2), pp. 345–385, doi:10.1006/inco.1999.2818.

[24] R. de Simone (1985): *Higher-level synchronising devices in* MEIJE-*SCCS*. Theoretical Computer Science 37, pp. 245–267, doi:10.1016/0304-3975(85)90093-3.

Conflict vs Causality in Event Structures

Daniele Gorla, Ivano Salvo, Adolfo Piperno
Sapienza University of Rome, Dpt. of Computer Science
{gorla,salvo,piperno}@di.uniroma1.it

Event structures are one of the best known models for concurrency. Many variants of the basic model and many possible notions of equivalence for them have been devised in the literature. In this paper, we study how the spectrum of equivalences for Labelled Prime Event Structures built by Van Glabbeek and Goltz changes if we consider two simplified notions of event structures: the first is obtained by removing the causality relation (Coherence Spaces) and the second by removing the conflict relation (Elementary Event Structures). As expected, in both cases the spectrum turns out to be simplified, since some notions of equivalence coincide in the simplified settings; actually, we prove that removing causality simplifies the spectrum considerably more than removing conflict. Furthermore, while the labeling of events and their cardinality play no role when removing causality, both the labeling function and the cardinality of the event set dramatically influence the spectrum of equivalences in the conflict-free setting.

1 Introduction

Event structures [23, 33] are one of the best known models for concurrency. Basically, they are collections of possible events, some of which are conflicting (i.e., the execution of an event forbids the execution of other events), while others are causally dependent (i.e., an event cannot be executed if it has not been preceded by other ones). Prime Event Structures (written PESs) are the earliest and simplest form of event structure, where causality is a partial order and conflict between events is inherited by their causal successors. Events are often labelled with actions, to represent different occurrences of the same action. In this paper, we shall focus on labelled PESs, referring to them simply as PESs, for the sake of simplicity.

Conflict and causality are fundamental concepts for concurrency; indeed, they can also be found in other well-established models for concurrent computation, like Petri nets [25, 26, 27] and process algebras [3, 19, 22] (where they are called choice and sequential composition, respectively). Not incidentally, both conflict and causality influence the evolution of an event structure, whose semantics is given by means of *configurations*: these are finite conflict-free subsets of events that are closed by causal predecessors. Configurations take note of the events occurred so far during a computation. Indeed, starting from the empty configuration, the evolution of an event structure is obtained by selecting one or more events that are causally enabled by the events executed so far, and non-conflicting with any of them. However, not all sets of events can be simultaneously executed: this yields the derived notion of *concurrent* events, that are those that are neither in conflict nor causally dependent from one another.

A fruitful research line is the study of different possible notions of equivalence for event structures, inspired by the richness of equivalences for process algebras [14, 15]. Indeed, apart from the classical distinction between trace and bisimulation-based equivalences, in the framework of PESs many features can be observed to distinguish two event structures. In this paper, we follow [16] and consider the following equivalences:

1. *interleaving trace and bisimulation equivalences* (written $\approx_{\text{it}}$ and $\approx_{\text{ib}}$): these are the direct counterparts of trace and bisimulation equivalence for process algebras [19, 22]; in the framework of PESs, only (the label of) one single event at a time is observed, either in a sequence forming a trace or in the bisimulation game based on coinduction.
2. *step trace and bisimulation equivalences* (written $\approx_{\text{st}}$ and $\approx_{\text{sb}}$) [28], where the units of observation are sets of concurrent (and causally enabled) events To be more precise, we do not observe sets of events but the multisets of the labels associated to the selected events (recall that the same label can be given to different events).
3. *pomset trace and bisimulation equivalences* (written $\approx_{\text{pt}}$ and $\approx_{\text{pb}}$) [4], where the units of observation are *sets of events* together with their causality and concurrency relations; again, since different events can have the same label, a set of events generates a partially ordered multiset (hence, the name *pomset*), based on the causality relation.
4. different variants of *history preserving bisimulation*, where the configurations of the two PESs related by a bisimulation must have the same causal dependencies. According to how this requirement is formalized, we have:
   (a) *weak history preserving bisimulation* (written $\approx_{\text{whb}}$) [9], where every pair of configurations is formed by isomorphic (w.r.t. their causal dependencies) pomsets;
   (b) *history preserving bisimulation* (written $\approx_{\text{hb}}$) [10, 31], where every pair of configurations is formed by isomorphic (w.r.t. their causal dependencies) pomsets and the isomorphism grows during the computation (whereas, for $\approx_{\text{whb}}$ two consecutive pairs of configurations could be related by totally different isomorphisms);
   (c) *hereditary history preserving bisimulation* (written $\approx_{\text{hhb}}$) [2], which is $\approx_{\text{hb}}$ with the additional requirement that the isomorphism is maintained also when going back in the computation.

These 9 equivalences, together with PES isomorphism $\cong$, form a well known spectrum [11, 16] that we depict in Figure 1 (where the term *autoconcurrency* means existence of a configuration containing two different concurrent events with the same label).

Orthogonally, since their birth, many variants of the basic framework have appeared in the literature. The basic model has been both extended with more sophisticated features and simplified by removing features. Richer notions of event structures include, among the others, flow event structures [5], stable/non-stable event structures [32] and configuration structures [18]. By contrast, simplified models are obtained either by removing the causality relation, yielding *coherence spaces* [12] (written CSs in this paper), or by removing the conflict relation, yielding *elementary event structures* [23] (written EESs). Both these models have interesting applications in the literature: the former one is used for giving the semantics of linear logic [12] and typed lambda-calculus [6, 7]; the latter one is a common variant of PESs ([23, 24, 16], just to cite a few).

The aim of this paper is to investigate how the spectrum of Figure 1 changes when passing from PESs to CSs and EESs. As expected, in both cases the spectrum turns out to be simplified, since some notions of equivalence coincide in the simplified settings. So, for every possible inclusion, we have to either (1) prove that the inclusion becomes an equality, or (2) provide an example in the simplified setting to distinguish the two equivalences (and confirm properness of the inclusion also in the simplified setting).

The spectrum is radically simplified in the framework of CSs, as depicted in Figure 2. As evident, removing the causality relation reduces a complex lattice to a simple chain: trace equivalences all coincide and represent the coarsest notion; they properly include bisimilarities (that all coincide, except for $\approx_{\text{hhb}}$)

$$\begin{tikzcd}
 & \approx_{\mathrm{it}} \\
\approx_{\mathrm{ib}} \arrow[ur] & \approx_{\mathrm{st}} \arrow[u] \\
\approx_{\mathrm{sb}} \arrow[u] \arrow[ur] & \approx_{\mathrm{pt}} \arrow[u] \\
\approx_{\mathrm{pb}} \arrow[u] \arrow[ur] & \approx_{\mathrm{whb}} \arrow[ul] \arrow[u] \\
\approx_{\mathrm{hb}} \arrow[u] \arrow[ur, dashed] & \\
\approx_{\mathrm{hhb}} \arrow[u] & \\
\cong \arrow[u] &
\end{tikzcd}$$

Figure 1: The spectrum of equivalences for PESs ('$\rightarrow$' means '$\subset$'; '$\dashrightarrow$' means '$=$', if no autoconcurrency is present, and means '$\subset$', otherwise)

$$\begin{tikzcd}
\approx_{\mathrm{it}} = \approx_{\mathrm{st}} = \approx_{\mathrm{pt}} \\
\approx_{\mathrm{ib}} = \approx_{\mathrm{sb}} = \approx_{\mathrm{pb}} = \approx_{\mathrm{whb}} = \approx_{\mathrm{hb}} \arrow[u] \\
\approx_{\mathrm{hhb}} \arrow[u] \\
\cong \arrow[u]
\end{tikzcd}$$

Figure 2: The spectrum for CSs

$$\begin{tikzcd}
 & \approx_{\mathrm{it}} \\
\approx_{\mathrm{ib}} \arrow[ur] & \approx_{\mathrm{st}} \arrow[u] \\
\approx_{\mathrm{sb}} \arrow[u] \arrow[ur] & \\
\approx_{\mathrm{pb}} = \approx_{\mathrm{pt}} = \approx_{\mathrm{whb}} = \approx_{\mathrm{hb}} = \approx_{\mathrm{hhb}} = \cong \arrow[u] &
\end{tikzcd}$$

Figure 3: The spectrum for finite EESs

$$\begin{tikzcd}
 & \approx_{\mathrm{it}} \\
\approx_{\mathrm{ib}} \arrow[ur] & \approx_{\mathrm{st}} \arrow[u] \\
\approx_{\mathrm{sb}} \arrow[u] \arrow[ur] & \approx_{\mathrm{pt}} \arrow[u] \\
\approx_{\mathrm{pb}} \arrow[u] \arrow[ur] & \approx_{\mathrm{whb}} \arrow[ul] \arrow[u] \arrow[l, dashed, "1"] \\
\approx_{\mathrm{hb}} \arrow[u] \arrow[ur, dashed, "2"'] & \\
\approx_{\mathrm{hhb}} \arrow[u, dashed, "3"] & \\
\cong \arrow[u, dashed, "4"] &
\end{tikzcd}$$

Figure 4: The spectrum for infinite EESs (a numbered dashed arrow denotes an open question; for questions 2, 3 and 4, the arrow becomes solid if the question has a positive answer and becomes '$=$' otherwise; for question 1, the arrow disappears il the answer is positive and becomes solid otherwise).

that in turn properly include the back-and-forth variant [8] of $\approx_{\mathrm{hb}}$. Furthermore, the labeling function plays no role in such results; so, even the "flattening" labeling (that associates the same action to every event) does not change the spectrum.

The situation is more articulated when conflict is removed, hence in the framework of EESs. A posteriori, this is not surprising because a partial order (viz., the causality relation) is a richer mathematical object than an irreflexive and symmetric relation (viz., the conflict relation). What is really surprising is the fact that having finitely or infinitely many events makes a significant difference in terms of the distinguishing power of the studied equivalences; Figures 3 and 4 give a visual account of the difference. The first easy, but still interesting, result for finite EESs is that $\approx_{\mathrm{pt}}$, $\approx_{\mathrm{pb}}$, $\approx_{\mathrm{whb}}$, $\approx_{\mathrm{hb}}$, $\approx_{\mathrm{hhb}}$ and $\cong$ all coincide. This can be justified by observing that, being finite and without conflict, the set of all the events of every such EES is a configuration of the EES itself; so, all notions of equivalence that rely on some kind of pomset isomorphism collapse to EES isomorphism. By contrast, for infinite EESs this does not hold anymore and some more inclusions that were proper in Figure 1 remain proper also in Figure 4. Four questions remain open about strictness of some inclusions for infinite EESs. However, even if the spectrum is not fully worked out, we have some examples that let us claim that cardinality of the event set matters when only causality is considered. By contrast, cardinality has no impact on the spectrum for CSs. Furthermore, we prove that restricting to "flattening" labeling functions makes $\approx_{\mathrm{it}}$ and $\approx_{\mathrm{ib}}$ collapse for EESs (again, in contrast with CSs).

For all these reasons, our results seem to suggest that causality is a more foundational building block than conflict in event structures, since it has a deeper impact on the discriminating power of equivalences for such models and because it is more sensitive than conflict to issues like the cardinality of the set of events and their labeling.

The rest of the paper is organized as follows. In Section 2, we recall the basic definitions and the spectrum for PESs, as reported in [11]. Then, we move to consider CSs (Section 3) and EESs (Section 4); for the latter model, we also distinguish what happens for finite (Section 4.1) and infinite structures (Section 4.2). Section 5 concludes the paper.

2 Background: Prime Event Structures

We start by summing up some well known notions from the theory of Event Structures [23], by following the presentation in [16].

Definition 1 (Prime Event Structures [23, 33])**.** *A (labeled)* Prime Event Structure *(PES, for short) over an alphabet $\mathscr{A}$ is a 4-tuple $\mathscr{E} = (E, \leq, \sharp, l)$ such that:*

- *E is a set of* events*;*
- *$\leq \subseteq E \times E$ is the* causality *relation, i.e. a partial order such that, for all $e \in E$, the set $\{e' : e' < e\}$ is finite;*
- *$\sharp \subseteq E \times E$ is the* conflict *relation, i.e. an irreflexive and symmetric relation such that, for all $e, e', e'' \in E$, if $e < e'$ and $e \sharp e''$, then $e' \sharp e''$;*
- *$l : E \to \mathscr{A}$ is the* labeling *function.*

Intuitively, $e' < e$ means that e cannot happen before e' (so, the execution of e causally depends on the execution of e'), whereas $e \sharp e'$ means that e and e' are mutually exclusive (so, the execution of one prevents the execution of the other). The condition $|\{e' : e' < e\}| < \infty$ ensures that every event can be executed in a finite amount of time (i.e, after the execution of finitely many events). Conflict inheritance (the condition in the third item of the previous definition) is a sort of 'sanity' condition, ensuring that every event inherits the conflicts of all its causal predecessors. Finally, labels represent actions entailed by events, and so different events can have the same label; this corresponds to the fact that the same action can occur different times during the execution of a system.

A derived notion is the *concurrency* relation, defined as follows: $e \; co \; e'$ iff $(e, e') \notin \leq \cup \geq \cup \sharp$. When convenient, we shall write a PES by using the usual process algebra notation, where '$\|$' means '*co*', ';' means '$<$' and '$+$' means '$\sharp$'; moreover, we just write the labels, assuming that the underlying events are all different.[1]

[1] We remark that we shall use this syntax only when it comes handy to describe some particular PES in a succinct way; in particular, in this paper we consider PESs as a per se semantic model, and not, e.g., as the interpretation domain for some process algebra. Furthermore, notice that PESs do not coincide with all the ESs that '$\|$', ';' and '$+$' can define: there are terms of this algebra that denote ESs that are not prime (e.g., $(a+b);c$) and there are PESs that are not definable using the given algebra (e.g., the event structure $\mathscr{E}$ in the proof of Prop. 8) [13, 29, 30].

Example 1. *The expression* $(a \parallel b) + (a;b)$ *denotes the PES* $\mathscr{E} = (E, \leq, \sharp, l)$ *such that* $E = \{e_1, e_2, e_3, e_4\}$, $e_i \leq e_i$ *(for* $i \in \{1,2,3,4\}$*),* $e_3 \leq e_4$, $\sharp = \{(e_1,e_3),(e_2,e_3),(e_3,e_1),(e_3,e_2),(e_1,e_4),(e_2,e_4),(e_4,e_1),(e_4,e_2)\}$, $l(e_1) = l(e_3) = a$, *and* $l(e_2) = l(e_4) = b$.

To be precise, $(a \parallel b) + (a;b)$ denotes the $\cong$-class of the PES $\mathscr{E}$ given in Example 1, where PES isomorphism is defined as follows.

Definition 2 (PES isomorphism)**.** *Let* $\mathscr{E} = (E, \leq_E, \sharp_E, l_E)$ *and* $\mathscr{F} = (F, \leq_F, \sharp_F, l_F)$ *be two PESs. We say that* $\mathscr{E}$ *and* $\mathscr{F}$ *are* isomorphic*, and write* $\mathscr{E} \cong \mathscr{F}$*, if there exists a biiection* $f : E \to F$ *such that. for every* $e, e' \in E$*, it holds that :*

- $e \leq_E e'$ *if and only if* $f(e) \leq_F f(e')$*;*
- $e \sharp_E e'$ *if and only if* $f(e) \sharp_F f(e')$*; and*
- $l_E(e) = l_F(f(e))$.

Essentially, PES isomorphism only abstracts away from the set of events. So, for example, any $\mathscr{F}$ isomorphic to the PES $\mathscr{E}$ of Example 1 must be such that $F = \{e'_1, e'_2, e'_3, e'_4\}$ and $\leq / \sharp / l$ are defined as in Example 1, but with e'_i in place of e_i.

The semantics of a PES $\mathscr{E}$ is defined in terms of the possible states that the system modeled by the PES can pass through during its evolution, where such states are defined as follows.

Definition 3 (Configurations)**.** *A* configuration *of a PES* $\mathscr{E} = (E, \leq, \sharp, l)$ *is any* $X \subseteq_{\text{fin}} E$ *such that*

- $e \not\sharp\, e'$*, for every* $e, e' \in X$*; and*
- $\{e' : e' < e\} \subseteq X$*, for every* $e \in X$.

We denote with $\mathit{Conf}(\mathscr{E})$ *the set of all configurations of* $\mathscr{E}$.

Configurations collect the events executed from the outset of the system; so, they must be finite (they have to represent states reachable in a finite time), conflict-free (two conflicting events cannot be executed in the same system evolution)) and closed w.r.t. causal predecessors (an event can happen only if all its predecessors happened before). For examples, the configurations of $\mathscr{E}$ from Example 1 are $\emptyset, \{e_1\}, \{e_2\}, \{e_3\}, \{e_1,e_2\}, \{e_3,e_4\}$; notice that $\{e_4\}$ is not a configuration because e_4 cannot stay in any configuration that misses its causal predecessor e_3, and that $\{e_1,e_3\}$ is not a configuration because $e_1 \sharp e_3$.

The way in which (the system modeled by) a PES evolves is usually given through some *labeled transition systems* (LTSs), on top of which we can build different notions of equivalence between PESs. We now recall both the main transition relations and the main equivalences built on top of them.

The first transition relation between configurations states that $X \xrightarrow{a} X'$ whenever $X \subset X'$ and $X' \setminus X = \{e\}$, with $l(e) = a$; notation $X \longrightarrow$ (resp., $X \not\longrightarrow$) means that there exist a and X' (resp., no a and X') such that $X \xrightarrow{a} X'$. Coming back to Example 1, we have that the possible transitions for $\mathscr{E}$ are:

$$\begin{tikzcd}
 & & & \{e_1\} \arrow[dr, "b"] & \\
\{e_3,e_4\} & \{e_3\} \arrow[l, "b"'] & \emptyset \arrow[l, "a"'] \arrow[ur, "a"] \arrow[dr, "b"'] & & \{e_1,e_2\} \\
 & & & \{e_2\} \arrow[ur, "a"'] &
\end{tikzcd}$$

The two most basic equivalences we shall consider are derived from process algebras and are *bisimulation* and *trace equivalence*. To define the latter, we use the notion of *(sequential) trace* of a PES $\mathscr{E}$, that is a sequence $a_1 \ldots a_k \in \mathscr{A}^*$ such that there exist $X_0, \ldots, X_k \in \mathit{Conf}(\mathscr{E})$ such that $X_0 = \emptyset$ and $X_i \xrightarrow{a_{i+1}} X_{i+1}$, for every $i = 0, \ldots, k-1$. We denote with $\mathit{SeqTr}(\mathscr{E})$ the set of the sequential traces of $\mathscr{E}$.

Definition 4 (Interleaving Trace Equivalence [19]). *$\mathscr{E} \approx_{\mathrm{it}} \mathscr{F}$ if $SeqTr(\mathscr{E}) = SeqTr(\mathscr{F})$.*

Definition 5 (Interleaving Bisimulation [22]). *A relation $R \subseteq Conf(\mathscr{E}) \times Conf(\mathscr{F})$ is an* interleaving bisimulation *beween $\mathscr{E}$ and $\mathscr{F}$ if*

- *$(\emptyset;\emptyset) \in R$;*
- *if $(X,Y) \in R$ and $X \xrightarrow{a} X'$, then $Y \xrightarrow{a} Y'$, for some Y' such that $(X',Y') \in R$; and*
- *if $(X,Y) \in R$ and $Y \xrightarrow{a} Y'$, then $X \xrightarrow{a} X'$, for some X' such that $(X',Y') \in R$.*

$\mathscr{E} \approx_{\mathrm{ib}} \mathscr{F}$ if there is an interleaving bisimulation between $\mathscr{E}$ and $\mathscr{F}$.

Transitions involving a single action can be generalized to *steps*, i.e. sets of events that can be executed simultaneously. Again, for the sake of abstraction, a step transition will be labeled with the multiset of labels associated to the chosen concurrent events. Formally, we write $X \xrightarrow{A} X'$ if $X \subset X'$, $X' \setminus X = G$, $\forall e,e' \in G.e\ co\ e'$, and A is the multiset over $\mathscr{A}$ formed by the labels of the events in G. For example, for $\mathscr{E}$ in Example 1, we now also have that $\emptyset \xrightarrow{\{a,b\}} \{e_1,e_2\}$. This yields the obvious generalization of interleaving bisimulation and trace equivalence, where step traces of $\mathscr{E}$, written $StepTr(\mathscr{E})$, are defined as expected (i.e., like sequential traces, but with steps in place of single events).

Definition 6 (Step Trace Equivalence [28]). *$\mathscr{E} \approx_{\mathrm{st}} \mathscr{F}$ if $StepTr(\mathscr{E}) = StepTr(\mathscr{F})$.*

Definition 7 (Step Bisimulation [28]). *A relation $R \subseteq Conf(\mathscr{E}) \times Conf(\mathscr{F})$ is a* step bisimulation *beween $\mathscr{E}$ and $\mathscr{F}$ if*

- *$(\emptyset;\emptyset) \in R$;*
- *if $(X,Y) \in R$ and $X \xrightarrow{A} X'$, then $Y \xrightarrow{A} Y'$, for some Y' such that $(X',Y') \in R$; and*
- *if $(X,Y) \in R$ and $Y \xrightarrow{A} Y'$, then $X \xrightarrow{A} X'$, for some X' such that $(X',Y') \in R$.*

$\mathscr{E} \approx_{\mathrm{sb}} \mathscr{F}$ if there exists a step bisimulation between $\mathscr{E}$ and $\mathscr{F}$.

Because of their definition, configurations are actually partially ordered sets (posets, for short), where the ordering is given by $\leq$. Indeed, we write $poset(X)$ to denote the labeled poset $(X, \leq|_X, l|_X)$, where $\leq|_X$ and $l|_X$ are the restrictions of $\leq$ and l to X. A more abstract view of a run is obtained by replacing events with their labels. This turns a poset into a partially ordered multiset (*pomset*, for short). Formally, the pomset associated to a configuration X, written $pomset(X)$, is the isomorphism class of $poset(X)$. We can then observe not just multisets, but multisets together with their ordering, i.e. pomsets; this generalizes the step semantics because, by observing pomsets, we are allowed to observe in one single transition also events that are not concurrent. To this aim, we denote with $Pom(\mathscr{E})$ the set of all pomsets of $\mathscr{E}$ and we label a transition with a pomset p, where $X \xrightarrow{p} X'$ if $X \subset X'$, $X' \setminus X = H$ and $p = pomset(H)$. Always referring to $\mathscr{E}$ in Example 1, we also have that $\emptyset \xrightarrow{a;b} \{e_3,e_4\}$.

Definition 8 (Pomset Trace Equivalence [4]). *$\mathscr{E} \approx_{\mathrm{pt}} \mathscr{F}$ if $Pom(\mathscr{E}) = Pom(\mathscr{F})$.*

Definition 9 (Pomset Bisimulation [4]). *A relation $R \subseteq Conf(\mathscr{E}) \times Conf(\mathscr{F})$ is a* pomset bisimulation *beween $\mathscr{E}$ and $\mathscr{F}$ if*

- *$(\emptyset;\emptyset) \in R$;*
- *if $(X,Y) \in R$ and $X \xrightarrow{p} X'$, then $Y \xrightarrow{p} Y'$, for some Y' such that $(X',Y') \in R$; and*
- *if $(X,Y) \in R$ and $Y \xrightarrow{p} Y'$, then $X \xrightarrow{p} X'$, for some X' such that $(X',Y') \in R$.*

$\mathscr{E} \approx_{\mathrm{pb}} \mathscr{F}$ if there exists a pomset bisimulation between $\mathscr{E}$ and $\mathscr{F}$.

An orthogonal way to generalise the interleaving bisimulation is to keep track of the causal dependencies and only relate configurations with the same causal history. This is done by requiring that the two configurations have isomorphic associated posets, where we also denote poset isomorphism with $\cong$.

Definition 10 (Weak History Preserving Bisimulation [9])**.** *A relation $R \subseteq Conf(\mathscr{E}) \times Conf(\mathscr{F})$ is a* weak history preserving bisimulation *beween $\mathscr{E}$ and $\mathscr{F}$ if*

- $(\emptyset;\emptyset) \in R$, *and*
- *if $(X,Y) \in R$ then*
  - $poset(X) \cong poset(Y)$;
  - *if $X \xrightarrow{a} X'$, then $Y \xrightarrow{a} Y'$, for some Y' such that $(X',Y') \in R$;*
  - *if $Y \xrightarrow{a} Y'$, then $X \xrightarrow{a} X'$, for some X' such that $(X',Y') \in R$.*

$\mathscr{E} \approx_{\text{whb}} \mathscr{F}$ *if there exists a weak history preserving bisimulation between $\mathscr{E}$ and $\mathscr{F}$.*

A stronger requirement is that the isomorphism relating $poset(X')$ and $poset(Y')$ cannot be arbitrary, but must extend the isomorphism relating $poset(X)$ and $poset(Y)$. This leads to the following definition.

Definition 11 (History Preserving Bisimulation [10, 31])**.** *A relation $R \subseteq Conf(\mathscr{E}) \times Conf(\mathscr{F}) \times 2^{Conf(\mathscr{E}) \times Conf(\mathscr{F})}$ is a* history preserving bisimulation *beween $\mathscr{E}$ and $\mathscr{F}$ if*

- $(\emptyset;\emptyset;\emptyset) \in R$, *and*
- *if $(X,Y,f) \in R$ then*
  - *f is an isomorphism between $poset(X)$ and $poset(Y)$;*
  - *if $X \xrightarrow{a} X'$, then $Y \xrightarrow{a} Y'$, for some Y' such that $(X',Y',f') \in R$, where $f'|_X = f$; and*
  - *if $Y \xrightarrow{a} Y'$, then $X \xrightarrow{a} X'$, for some X' such that $(X',Y',f') \in R$, where $f'|_X = f$.*

$\mathscr{E} \approx_{\text{hb}} \mathscr{F}$ *if there exists a history preserving bisimulation between $\mathscr{E}$ and $\mathscr{F}$.*

The notion of history preserving bisimulation can be finally generalised by also asking for a 'backwards' bisimulation game, along the way of back-and-forth bisimulation [8].

Definition 12 (Hereditary History Preserving Bisimulation [2])**.** *A history preserving bisimulation R beween $\mathscr{E}$ and $\mathscr{F}$ is* hereditary *if, for every $(X,Y,f) \in R$, it holds that $X' \xrightarrow{a} X$ implies $(X', f(X'), f|_{X'}) \in R$ and $Y' \xrightarrow{a} Y$ implies $(f^{-1}(Y'), Y', f|_{f^{-1}(Y')}) \in R$.*

$\mathscr{E} \approx_{\text{hhb}} \mathscr{F}$ if there exists a hereditary history preserving bisimulation between $\mathscr{E}$ and $\mathscr{F}$.

All the equivalences presented so far form a well-known spectrum [11, 16], depicted in Figure 1 (the only inclusions that are not present in [16] are $\approx_{\text{whb}} \subset \approx_{\text{sb}}$ and $\approx_{\text{whb}} \subset \approx_{\text{pt}}$, that are proved in [11]). There, the term *autoconcurrency* means existence of a configuration containing two different concurrent events with the same label.

3 Conflict without Causality: Coherence Spaces

We now consider the first restriction of PESs, obtained by considering an empty causality relation. This leads to *Coherence Spaces* [12], a model largely studied, e.g., in the field of linear logic and in the semantics of typed lambda-calculus [6, 7, 12].

Definition 13. *A* coherence space *(written CS) over an alphabet $\mathscr{A}$ is a PES $\mathscr{E}$ where the causality relation is empty.*

Thus, we shall usually omit $\leq$ from the definition of a CS. In the setting of CSs, several definitions are radically simplified. For example, a configuration is simply a finite and conflict-free subset of E; similarly, two events are concurrent if they are not in conflict. Moreover, a step and a pomset are simply multisets and, hence, the two notions do coincide.

Consequently, the spectrum of Figure 1 can be simplified, but it is still not trivial. Indeed, removing the causality relation reduces a complex lattice to a simple chain: trace equivalences all coincide and represent the coarsest notion; they properly include bisimulations (that all coincide, except for $\approx_{\text{hhb}}$) that in turn properly include the back-and-forth variant [8] of $\approx_{\text{hb}}$ and the latter is still strictly coarser than isomorphism. The spectrum is depicted in Figure 2 and it is the first main result of this paper; the following propositions are needed to establish it.

Proposition 1. *For CSs, if $\mathscr{E} \approx_{\text{ib}} \mathscr{F}$ then $\mathscr{E} \approx_{\text{hb}} \mathscr{F}$.*

Proof. Let R be an interleaving bisimulation between $\mathscr{E}$ and $\mathscr{F}$, and consider the following relation:

$$R' \triangleq \bigcup_{(X,Y)\in R} \{(X,Y,f) \ : \ f \text{ is an isomorphism between } X \text{ and } Y\}$$

Trivially, $(\emptyset,\emptyset,\emptyset) \in R'$, because $(\emptyset,\emptyset) \in R$ and every set is isomorphic to itself. Let $(X,Y,f) \in R'$; by construction, $f : X \to Y$ is a bijection such that $l(x) = l(f(x))$, for every $x \in X$.[2] Now, let $X \xrightarrow{a} X'$; this means that $X' = X \uplus \{e\}$ and $l(e) = a$. Since $(X,Y) \in R$, there exists Y' such that $Y \xrightarrow{a} Y' = Y \uplus \{e'\}$, where $l(e') = a$, and $(X',Y') \in R$. It is easy to see that $f' = f \cup \{(e,e')\}$ is an isomorphism between X' and Y' and so $(X',Y',f') \in R$. $\square$

Proposition 2. *For CSs, if $\mathscr{E} \approx_{\text{it}} \mathscr{F}$ then $\mathscr{E} \approx_{\text{pt}} \mathscr{F}$.*

Proof. Since a pomset is just a multiset (i.e., a step), it suffices to prove that $\mathscr{E} \approx_{\text{it}} \mathscr{F}$ implies $\mathscr{E} \approx_{\text{st}} \mathscr{F}$. Let $A_1 \dots A_k \in \mathit{StepTr}(\mathscr{E})$; we have to show that $A_1 \dots A_k \in \mathit{StepTr}(\mathscr{F})$.

By definition, there exist $X_0,\dots,X_k \in \mathit{Conf}(\mathscr{E})$ such that $X_0 = \emptyset$ and $X_{i-1} \xrightarrow{A_i} X_i$, for every $i = 1,\dots,k$. This means that $X_{i-1} \subset X_i$, $X_i \setminus X_{i-1} = \{e^i_1,\dots,e^i_{j_i}\}$, $\forall h \neq q.\ e^i_h\ co\ e^i_q$ and A_i is the multiset formed by $l(e^i_1),\dots,l(e^i_{j_i})$. Thus, $X_{i-1} \xrightarrow{l(e^i_1)} \dots \xrightarrow{l(e^i_{j_i})} X_i$ and so $l(e^1_1)\dots l(e^1_{j_1})\dots l(e^k_1)\dots l(e^k_{j_k}) \in \mathit{SeqTr}(\mathscr{E})$. By hypothesis, $l(e^1_1)\dots l(e^1_{j_1})\dots l(e^k_1)\dots l(e^k_{j_k}) \in \mathit{SeqTr}(\mathscr{F})$; i.e., there exist $Y_0,\dots,Y_{j_1+\dots+j_k} \in \mathit{Conf}(\mathscr{F})$ such that $Y_0 = \emptyset$ and $Y_0 \xrightarrow{l(e^1_1)} Y_1 \dots \xrightarrow{l(e^k_{j_k})} Y_{j_1+\dots+j_k}$. Since $Y_{j_1+\dots+j_k}$ is a configuration and configurations in CSs are conflict-free sets, we have that all the events occurring in it are concurrent. Thus, we can group single transitions into steps and obtain $A_1 \dots A_k \in \mathit{StepTr}(\mathscr{F})$. $\square$

Proposition 3. *There exist CSs $\mathscr{E}$ and $\mathscr{F}$ such that $\mathscr{E} \approx_{\text{it}} \mathscr{F}$ but $\mathscr{E} \not\approx_{\text{ib}} \mathscr{F}$.*

Proof. Consider $\mathscr{E} = a + (a \parallel a)$ and $\mathscr{F} = a \parallel a$: they have the same traces (viz., $\{\varepsilon, a, aa\}$) but $\mathscr{E}$, after the leftmost a, is stuck, whereas $\mathscr{F}$, after every a, is not. $\square$

Proposition 4. *There exist CSs $\mathscr{E}$ and $\mathscr{F}$ such that $\mathscr{E} \approx_{\text{hb}} \mathscr{F}$ but $\mathscr{E} \not\approx_{\text{hhb}} \mathscr{F}$.*

[2] Indeed, notice that, for CSs, the poset associated to a configuration is just a collection of (labeled) events (i.e., the ordering relation is empty) and, hence, poset isomorphism has only to respect the labeling.

Proof. Consider

$$\mathscr{E} = a \parallel (a + (a \parallel a)) \qquad \text{and} \qquad \mathscr{F} = (a \parallel (a + (a \parallel a))) + (a \parallel a)$$

and their LTSs (the events have been numbered in increasing order, from left to right, both in $\mathscr{E}$ and in $\mathscr{F}$):

$$\begin{tikzcd}
 & & \{e_1,e_3,e_4\} & \\
 & \{e_1,e_3\} \arrow[ur] & \{e_1,e_4\} \arrow[u] & \{e_3,e_4\} \arrow[ul] \\
 & \{e_1\} \arrow[u] \arrow[ur] \arrow[dl] & \{e_3\} \arrow[ul] \arrow[ur] & \{e_4\} \arrow[ul] \arrow[u] \\
\{e_1,e_2\} & & \emptyset \arrow[ul] \arrow[u] \arrow[ur] \arrow[dl] \arrow[r, dashed] \arrow[d, dashed] & \{e_6\} \arrow[d, dashed] \\
 & \{e_2\} \arrow[ul] & \{e_5\} \arrow[r, dashed] & \{e_5,e_6\}
\end{tikzcd}$$

Here, states are configurations, arrows are a-labeled transitions and the LTS for $\mathscr{E}$ is the solid part, whereas the LTS for $\mathscr{F}$ also includes the dashed part.

The only possible history preserving bisimulation between $\mathscr{E}$ and $\mathscr{F}$ is the one that acts as the identity on the common configurations and that associates $\{e_5,e_6\}$ with $\{e_1,e_2\}$ and both $\{e_5\}$ and $\{e_6\}$ with $\{e_2\}$. However, it is not hereditary because from $\{e_1,e_2\}$ we can backtrack to $\{e_1\}$ and from here we can perform two a's in sequence; by contrast, every backtrack from $\{e_5,e_6\}$ leads to a configuration that can only perform one single a. □

Proposition 5. *There exist CSs $\mathscr{E}$ and $\mathscr{F}$ such that $\mathscr{E} \approx_{\text{hhb}} \mathscr{F}$ but $\mathscr{E} \not\cong \mathscr{F}$.*

Proof. The example given in [2] for proving a similar claim (viz., $\mathscr{E} = a$ and $\mathscr{F} = a + a$) is in fact made up from two CSs. □

Quite surprisingly, the proofs of Propositions 1 and 2 do not rely on the fact that labels are different or not, and the examples provided in Propositions 3, 4 and 5 are built on CSs where all events have the same label. Hence, in the setting of CSs, the labeling function has no impact on the spectrum of Figure 2.

4 Causality without Conflict: Elementary ESs

A second restriction of PESs is obtained by considering an empty conflict relation; this yields *Elementary Event Structures* [23].

Definition 14. *An* elementary event structure *(written EES) over an alphabet $\mathscr{A}$ is a PES $\mathscr{E}$ where the conflict relation is empty.*

Consequently, we shall omit $\sharp$ from the definition of an EES. EESs are a particular kind of directed acyclic graphs, where every path from u to v entails the existence of a directed edge (u,v); this comes from the fact that causality is transitive. For the sake of simplicity, we shall sometimes represent EESs with the transitive reduction[3] of their causality relation. For example,

$$\begin{tikzcd}
c & \\
b \arrow[u] & b \arrow[ul] \\
a \arrow[u] \arrow[ur] & a \arrow[u]
\end{tikzcd}$$

represents the (isomorphism class of the) EES $\mathscr{E} = (E, \leq, l)$, where $E = \{e_1, e_2, e_3, e_4, e_5\}$, $l(e_1) = l(e_2) = a$, $l(e_3) = l(e_4) = b$, $l(e_5) = c$, $e_i \leq e_i$, $e_1 \leq e_3$, $e_1 \leq e_4$, $e_1 \leq e_5$, $e_2 \leq e_4$, $e_2 \leq e_5$, $e_3 \leq e_5$ and $e_4 \leq e_5$.

We now present the results needed to adapt the spectrum of Figure 1 to EESs; this is the second main contribution of our work. Surprisingly, the spectrum changes according to whether the set of events is finite or not. However, there are a few common results, that we now present.

For interleaving and step equivalences, the spectrum for EESs is the same as that for PESs: the inclusions depicted in the upper part of Figure 1 also hold for EESs; what changes are the counterexamples needed to distinguish them. We now provide the distinguishing examples in the framework of EESs.

Proposition 6. *For EESs, there exist $\mathscr{E}$ and $\mathscr{F}$ such that $\mathscr{E} \approx_{\mathrm{ib}} \mathscr{F}$ and $\mathscr{E} \approx_{\mathrm{it}} \mathscr{F}$, whereas $\mathscr{E} \not\approx_{\mathrm{sb}} \mathscr{F}$ and $\mathscr{E} \not\approx_{\mathrm{st}} \mathscr{F}$.*

Proof. Consider the EESs $\mathscr{E} = a;a$ and $\mathscr{F} = a \parallel a$. □

Proposition 7. *For EESs, there exist $\mathscr{E}$ and $\mathscr{F}$ such that $\mathscr{E} \approx_{\mathrm{it}} \mathscr{F}$, whereas $\mathscr{E} \not\approx_{\mathrm{ib}} \mathscr{F}$.*

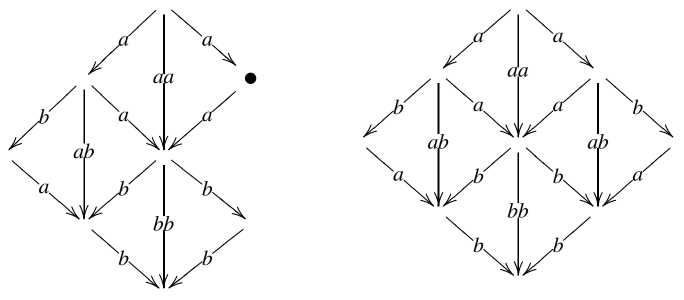
Proof. Consider the EESs $\mathscr{E} = (a \parallel b);(a \parallel b)$ and $\mathscr{F} = (a;b) \parallel (b;a)$. Trivially, $\mathscr{E} \approx_{\mathrm{it}} \mathscr{F}$, since $SeqTr(\mathscr{E}) = SeqTr(\mathscr{F}) = \{\varepsilon, a, b, ab, ba, aba, abb, baa, bab, abab, abba, baab, baba\}$. By contrast $\mathscr{E} \not\approx_{\mathrm{ib}} \mathscr{F}$, since in $\mathscr{F}$ we can reach, after executing the leftmost a and b, a state where only b is possible, whereas in $\mathscr{E}$, after every a and b, both a and b are always enabled. □

Proposition 8. *For EESs, there exist $\mathscr{E}$ and $\mathscr{F}$ such that $\mathscr{E} \approx_{\mathrm{st}} \mathscr{F}$ whereas $\mathscr{E} \not\approx_{\mathrm{ib}} \mathscr{F}$.*

Proof. Consider the EESs

$$\mathscr{E} = \begin{tikzcd} b & b \\ a \arrow[u] \arrow[ur] & a \arrow[u] \end{tikzcd} \qquad \mathscr{F} = \begin{tikzcd} b & b \\ a \arrow[u] & a \arrow[u] \end{tikzcd}$$

The step LTSs resulting from these EESs (where states are configurations and arrows represent transitions) are:



[3] The *transitive reduction* of a DAG D is the (unique) smallest DAG D' which preserves the reachability relation of D. Note that two transitively reduced DAGs are isomorphic if and only if their transitive closures are isomorphic.

From them, checking $\approx_{\mathrm{st}}$ is immediate. On the other hand, $\approx_{\mathrm{ib}}$ does not hold because there exists a configuration in the left-hand side LTS (marked with '•') reachable after an a that cannot perform a b; by contrast, every configuration reachable after an a in the right-hand side LTS can always perform a b. ☐

An easy corollary of the previous result is that, for EESs, $\approx_{\mathrm{st}}$ is not contained in $\approx_{\mathrm{pt}}$ and $\approx_{\mathrm{sb}}$.

Proposition 9. *For EESs, there exist $\mathscr{E}$ and $\mathscr{F}$ such that $\mathscr{E} \approx_{\mathrm{sb}} \mathscr{F}$ whereas $\mathscr{E} \not\approx_{\mathrm{whb}} \mathscr{F}$ and $\mathscr{E} \not\approx_{\mathrm{pt}} \mathscr{F}$.*

Proof. Consider the EESs

$$\mathscr{E} = \begin{tikzcd} & & & e_7 & & & \\ & & & e_6 & & & \\ e_2 \arrow[uurrr] & & e_3 \arrow[ur] & & e_4 \arrow[ul] & & e_5 \arrow[uulll] \\ & e_0 \arrow[ul] \arrow[ur] \arrow[urrr] & & & & e_1 \arrow[ul] \arrow[ur] \arrow[ulll] & \end{tikzcd} \qquad \mathscr{F} = \begin{tikzcd} & e'_6 & & & e'_7 & \\ e'_2 \arrow[ur] & & e'_3 \arrow[ul] & e'_4 \arrow[ur] & & e'_5 \arrow[ul] \\ & e'_0 \arrow[ul] \arrow[ur] \arrow[urr] & & & e'_1 \arrow[ull] \arrow[ul] \arrow[ur] & \end{tikzcd}$$

where all events $e_0,\dots,e_7,e'_0,\dots,e'_7$ have the same label. It can be readily checked that $\mathscr{E} \ncong \mathscr{F}$ because events e_6 and e_7 in $\mathscr{E}$ have no isomorphic correspondence in $\mathscr{F}$. Thus, trivially, $\mathscr{E}$ and $\mathscr{F}$ cannot be in $\approx_{\mathrm{pt}}$; moreover, they cannot either be in $\approx_{\mathrm{whb}}$ because every weak history preserving bisimulation must contain the pair (E,F), but this is not possible since $\mathscr{E} \ncong \mathscr{F}$.

By contrast, we shall now prove that $\mathscr{E} \approx_{\mathrm{sb}} \mathscr{F}$. To this aim, for a generic EES $\mathscr{E} = (E,\leq,l)$ and for every $X \in Conf(\mathscr{E})$, we denote with $\mathscr{E}_X$ the EES $(E\setminus X, \leq|_{E\setminus X}, l|_{E\setminus X})$. It is now easy to check that:

- $\mathscr{E}_{\{e_0\}} \cong \mathscr{F}_{\{e'_0\}}$, via some isomorphism f_{e_0,e'_0} (for example: $(e_1,e'_1),(e_2,e'_2),(e_3,e'_4),(e_4,e'_5),(e_5,e'_3)$, $(e_6,e'_7),(e_7,e'_6)$);
- $\mathscr{E}_{\{e_1\}} \cong \mathscr{F}_{\{e'_1\}}$, via some isomorphism f_{e_1,e'_1} (for example: $(e_0,e'_0),(e_2,e'_4),(e_3,e'_2),(e_4,e'_3),(e_5,e'_5)$, $(e_6,e'_6),(e_7,e'_7)$);
- $\mathscr{E}_{\{e_0,e_1\}} \cong \mathscr{F}_{\{e'_0,e'_1\}}$, via some isomorphism $f_{e_0e_1,e'_0e'_1}$ (for example: $(e_2,e'_2),(e_3,e'_4),(e_4,e'_5),(e_5,e'_3)$, $(e_6,e'_7),(e_7,e'_6)$).

Notationally, let $f(X)$ denote $\{f(x) : x \in X\}$; it can be now verified that the relation

$$\begin{aligned} R = \; & \{(\emptyset,\emptyset)\} \cup \\ & \cup \textstyle\bigcup_{X\in Conf(\mathscr{E}_{\{e_0\}})} \{(\{e_0\}\cup X, \{e'_0\}\cup f_{e_0,e'_0}(X))\} \\ & \cup \textstyle\bigcup_{X\in Conf(\mathscr{E}_{\{e_1\}})} \{(\{e_1\}\cup X, \{e'_1\}\cup f_{e_1,e'_1}(X))\} \\ & \cup \textstyle\bigcup_{X\in Conf(\mathscr{E}_{\{e_0,e_1\}})} \{(\{e_0,e_1\}\cup X, \{e'_0,e'_1\}\cup f_{e_0e_1,e'_0e'_1}(X))\} \end{aligned}$$

is a step bisimulation between $\mathscr{E}$ and $\mathscr{F}$. ☐

An easy corollary of the previous result is that, for EESs, $\approx_{\mathrm{sb}}$ is not contained in $\approx_{\mathrm{pb}}$, $\approx_{\mathrm{hb}}$, $\approx_{\mathrm{hhb}}$, and $\cong$. Furthermore, notice that the examples provided in Propositions 6 and 9 use EESs with a "flattening" labeling function (mapping all events to the same label); by contrast, this is not the case in Propositions 7 and 8. This is not incidental, since, for EESs with all events labeled the same, $\approx_{\mathrm{ib}}$ and $\approx_{\mathrm{it}}$ coincide; to prove this, we first need a lemma.

Lemma 1. *Let $\mathscr{E} = (E,\leq,l)$ be an EES and let $X \in Conf(\mathscr{E})$; then either $X \xrightarrow{l(e)} X\cup\{e\}$, for some $e \in E\setminus X$, or $X = E$.*

Proof. Let $e \in E$; by induction on $|\{e' : e' < e\}|$, we prove that either $e \in X$ or there exists an $e' \leq e$ such that $X \xrightarrow{l(e')} X \cup \{e'\}$. The base case is trivial. For the inductive case, let us assume e with at least one predecessor. If $e \in X$, we are done. If X contains all the predecessors of e (but not e), then $X \xrightarrow{l(e)} X \cup \{e\}$. Otherwise, consider any $e' < e$ not contained in X; the claim follows by induction, since e' has less predecessors than e (indeed, every predecessor of e' is also a predecessor of e). □

Proposition 10. *For EESs with labeling set* $\mathscr{A} = \{a\}$, $\approx_{\text{ib}} = \approx_{\text{it}}$.

Proof. Lemma 1 entails that $SeqTr(\mathscr{E})$ is $\{a^n : 0 \leq n \leq |E|\}$, if E is finite, or $\{a^n : n \geq 0\}$, otherwise. The same holds for $\mathscr{F}$; hence, if $\mathscr{E} \approx_{\text{it}} \mathscr{F}$, then $|E| = |F|$.

Now, let $\mathscr{E} \approx_{\text{it}} \mathscr{F}$ and $R = \{(X,Y) : X \in Conf(\mathscr{E}), Y \in Conf(\mathscr{F}), |X| = |Y|\}$. Trivially, $(\emptyset, \emptyset) \in R$. Furthermore, if $(X,Y) \in R$ and $X \xrightarrow{a} X'$, then $|Y| = |X| < |E| = |F|$; again by Lemma 1, there exists $Y \xrightarrow{a} Y'$ and, by construction, $(X',Y') \in R$. □

Hence, differently from CSs, in the framework of EESs the labeling function has an impact on the distinguishing power of the equivalences studied.

To complete the hierarchy of equivalences for EESs, we surprisingly discovered that there is a deep difference if we consider finite or infinite event structures. For the former ones, we have been able to completely define the spectrum; for the latter ones, we still have some open questions, mostly on the history preserving bisimulations.

4.1 Finite EESs

For finite EESs, we have the following results that lead to the spectrum in Figure 3.

Proposition 11. *Let* $\mathscr{E}$ *and* $\mathscr{F}$ *be finite EESs such that* $\mathscr{E} \approx_{\text{pt}} \mathscr{F}$*; then* $\mathscr{E} \cong \mathscr{F}$.

Proof. The key observation is that, in every finite EES $\mathscr{E}$, the set of all the events E is a configuration (it is finite, conflict-free and closed by causal predecessors). Hence, $Pomset(E) = \mathscr{E}$. So, if $\mathscr{E} \approx_{\text{pt}} \mathscr{F}$, it holds that $\mathscr{E}$ and $\mathscr{F}$ have the same pomsets; in particular, the pomsets corresponding to E and F must be the same. Hence, the two EESs are isomorphic. □

Proposition 12. *For finite EESs,* $\cong\, = \,\approx_{\text{hhb}} \,=\, \approx_{\text{hb}} \,=\, \approx_{\text{whb}} \,=\, \approx_{\text{pb}} \,=\, \approx_{\text{pt}}$.

Proof. For all equivalences but $\approx_{\text{whb}}$ the claim is an easy corollary of the previous proposition, by the fact that $\cong \,\subseteq\, \approx_{\text{pt}}$ for PESs (and, hence, also for EESs). For $\approx_{\text{whb}}$, take $\mathscr{E} \approx_{\text{whb}} \mathscr{F}$ and consider a sequence of transitions $\emptyset \xrightarrow{a_1} \ldots \xrightarrow{a_n} E$ (whenever $|E| = n$). The only possible reply to this sequence is some $\emptyset \xrightarrow{a_1} \ldots \xrightarrow{a_n} F'$ such that $F' = F$, otherwise $F' \longrightarrow$ whereas $E \not\longrightarrow$. Thus, $\mathscr{E} \cong \mathscr{F}$, since $\mathscr{E} = poset(E)$ and $\mathscr{F} = poset(F)$. □

4.2 Infinite EESs

For infinite EESs, we first notice that, if we consider EESs of different cardinality, Proposition 11 does not hold. To see this, consider $\mathscr{E}$ and $\mathscr{F}$ made up, respectively, by a numerable and by a non-numerable set of concurrent copies of the same pomset; clearly, the two structures have the same (finite) pomsets and, hence, are pomset trace equivalent, but of course they are not isomorphic.

Moreover, Proposition 11 does not hold either for EESs of the same cardinality, as the following Propositions entail.

Proposition 13. *There exist $\mathscr{E}$ and $\mathscr{F}$ infinite EESs such that $\mathscr{E} \approx_{\mathrm{pb}} \mathscr{F}$, $\mathscr{E} \not\approx_{\mathrm{hb}} \mathscr{F}$ and $\mathscr{E} \not\approx_{\mathrm{whb}} \mathscr{F}$.*

Proof. Assume two numerable sets of events, $E = \{e_{ij}\}_{0 \leq j \leq i}$ and $F = \{e'_{ij}\}_{i,j \geq 0}$. Let $\leq_E$ (resp., $\leq_F$) be such that $e_{ij} \leq_E e_{ik}$ (resp., $e'_{ij} \leq_F e'_{ik}$) if and only if $j \leq k$. Finally, let every event be labeled with the same label a, both in $\mathscr{E}$ and in $\mathscr{F}$. Pictorially:

$$\mathscr{E} = \begin{tikzcd} & & e_{22} & \\ & e_{11} & e_{21} \arrow[u] & \dots \\ e_{00} & e_{10} \arrow[u] & e_{20} \arrow[u] & \end{tikzcd} \qquad \mathscr{F} = \begin{tikzcd} \dots & \dots & \dots & \\ e'_{02} \arrow[u] & e'_{12} \arrow[u] & e'_{22} \arrow[u] & \\ e'_{01} \arrow[u] & e'_{11} \arrow[u] & e'_{21} \arrow[u] & \dots \\ e'_{00} \arrow[u] & e'_{10} \arrow[u] & e'_{20} \arrow[u] & \end{tikzcd}$$

To show that $\mathscr{E} \not\approx_{\mathrm{hb}} \mathscr{F}$ and $\mathscr{E} \not\approx_{\mathrm{whb}} \mathscr{F}$, consider $\emptyset \xrightarrow{a} \{e_{00}\}$: the only possible reply in $\mathscr{F}$ is $\emptyset \xrightarrow{a} \{e'_{i0}\}$, for some i. However, $\{e_{00}\}$ and $\{e'_{i0}\}$ cannot be related by any history or weak history preserving bisimulation: indeed, the challenge $\{e'_{i0}\} \xrightarrow{a} \{e'_{i0}, e'_{i1}\}$ has no possible reply, since there is no event in E causally dependent on e_{00} (whereas $e'_{i0} <_F e'_{i1}$).

To prove that $\mathscr{E} \approx_{\mathrm{pb}} \mathscr{F}$, consider

$$\begin{aligned} R_0 &= \{(\emptyset, \emptyset)\} \\ R_{n+1} &= \{(X', Y') \ : \ \exists (X,Y) \in R_n \exists p \,.\, X \xrightarrow{p} X' \wedge Y \xrightarrow{p} Y'\} \\ R &= \textstyle\bigcup_{n \geq 0} R_n \end{aligned}$$

We now prove that R is a pomset bisimulation. By construction, $(\emptyset, \emptyset) \in R$. Let $(X,Y) \in R$. If $X \xrightarrow{p} X'$, then p is a finite collection of finite chains (w.r.t. $\leq_E$) and, hence, can be embedded into $\{e'_{ij}\}_{i>n, j \geq 0}$, where n is the largest integer such that $e'_{n0} \in Y$. Let $\hat{Y} \subset \{e'_{ij}\}_{i>n, j \geq 0}$ be such that $Pomset(\hat{Y}) = p$; then, $Y \xrightarrow{p} Y \uplus \hat{Y} = Y'$ and $(X', Y') \in R$ by construction. If $Y \xrightarrow{p} Y'$, then p is a finite collection of finite chains (w.r.t. $\leq_F$); let h be the shortest of such chains. Now, p can be embedded into $\{e_{ij}\}_{i>m, 0 \leq j \leq i}$, where $m = \max\{h, n\}$ and n is the largest integer such that $e_{n0} \in X$. Let $\hat{X} \subset \{e_{ij}\}_{i>m, 0 \leq j \leq i}$ be such that $Pomset(\hat{X}) = p$; then, $X \xrightarrow{p} X \uplus \hat{X} = X'$ and $(X', Y') \in R$ by construction. $\square$

Proposition 14. *There exist $\mathscr{E}$ and $\mathscr{F}$ infinite EESs such that $\mathscr{E} \approx_{\mathrm{pt}} \mathscr{F}$ but $\mathscr{E} \not\approx_{\mathrm{ib}} \mathscr{F}$.*

Proof. Let us consider

$$\mathscr{E} = \begin{tikzcd} & b & b & b & \\ a & a \arrow[u] & a \arrow[u] & a \arrow[u] & \dots \end{tikzcd} \qquad \mathscr{F} = \begin{tikzcd} b & b & b & \\ a \arrow[u] & a \arrow[u] & a \arrow[u] & \dots \end{tikzcd}$$

We have that $\mathscr{E} \approx_{\mathrm{pt}} \mathscr{F}$ because

$$pomsets(\mathscr{E}) = pomsets(\mathscr{F}) = \left\{ \underbrace{a \ a \ \dots \ a}_{m} \ \underbrace{\begin{array}{cccc} b & b & & b \\ \uparrow & \uparrow & & \uparrow \\ a & a & \dots & a \end{array}}_{n} \right\}_{m,n \geq 0}$$

By contrast, the singleton a in $\mathscr{E}$ cannot be replied to by any a in $\mathscr{F}$ because the latter enables a b, whereas the former does not. $\square$

An easy corollary of the last Proposition is that $\approx_{\text{pt}}$ does not imply $\approx_{\text{whb}}$, $\approx_{\text{pb}}$ and $\approx_{\text{sb}}$. We conclude this section by a list of questions that remain to be answered.

Open questions: Are there $\mathscr{E}$ and $\mathscr{F}$ (infinite EESs) such that

1. $\mathscr{E} \approx_{\text{whb}} \mathscr{F}$ but $\mathscr{E} \not\approx_{\text{pb}} \mathscr{F}$?
2. $\mathscr{E} \approx_{\text{whb}} \mathscr{F}$ but $\mathscr{E} \not\approx_{\text{hb}} \mathscr{F}$?
3. $\mathscr{E} \approx_{\text{hb}} \mathscr{F}$ but $\mathscr{E} \not\approx_{\text{hhb}} \mathscr{F}$?
4. $\mathscr{E} \approx_{\text{hhb}} \mathscr{F}$ but $\mathscr{E} \ncong \mathscr{F}$?

If all these open questions have a positive answer, the spectrum for infinite EESs, depicted in Figure 4, is the same as the one for general PESs, depicted in Figure 1. Notice that, if open question 1 has a positive answer, the same holds also for open question 2. However, we conjecture that also in the setting of infinite EESs all history-preserving bisimulation equivalences coincide and coincide with isomorphism; however, we still do not have enough evidences for formally proving this claim.

5 Conclusion

In this paper we studied how the spectrum of equivalences for PESs defined in [11, 16] changes when alternatively removing causality and conflict. In both cases, equivalences that are properly included in one another for PESs turn out to coincide and this is more evident in CSs than in EESs. Moreover, both the labeling function and the cardinality of the event set influence the spectrum for EESs, whereas they have no impact on the spectrum for CSs. For these reasons, we argue that causality is a more foundational building block than conflict in event structures, since it has a deeper impact on the discriminating power of equivalences for such models and because it is more sensitive than conflict to issues like the cardinality of the set of events and their labeling.

Surely, our results can be also related to the fact that the equivalences considered are causality-based (apart from the interleaving ones). Maybe, conflict could have a deeper impact than causality on other kinds of equivalences or on different models (for instance, variants of ESs with asymmetric choice, or with two different kinds of choices – external and internal, or nondeterministic and probabilistic). This is a first interesting line for future research.

Another possible extension of our work is the investigation of other equivalences (like, e.g., those presented in [17, Sect. 3]) and their impact on the spectra presented in this paper. However, we do not believe that this would change the message conveyed by this paper. By contrast, a challenging direction for future research would be the adaptation to CSs and EESs of the logical characterizations given by [1] to the equivalences studied in this paper. In particular, it would be nice to see how the logical operators defined in [1] can be simplified for capturing the equivalences in the simplified frameworks.

Finally, it is interesting to note that the pair of EESs in the proof of Proposition 9 has been obtained through an exhaustive search on transitively reduced DAGs, using the tools in the `nauty/Traces` [20, 21] distribution. More precisely, it is the smallest (with respect to number of vertices) pair of non-isomorphic transitively reduced DAGs having the same multiset of source-deleted subgraphs.

Acknowledgements We are grateful to Silvia Crafa and Paolo Baldan for fruitful discussions and to Rob van Glabbeek for the counterexample of Proposition 4.

References

[1] Paolo Baldan & Silvia Crafa (2014): *A Logic for True Concurrency*. J. ACM 61(4), pp. 24:1–24:36, doi:10.1145/2629638.

[2] Marek A. Bednarczyk (1991): *Hereditary history preserving bisimulations or what is the power of the future perfect in program logics*. Technical report, Polish Academy of Sciences.

[3] Jan A. Bergstra & Jan Willem Klop (1984): *Process Algebra for Synchronous Communication*. Information and Control 60(1-3), pp. 109–137, doi:10.1016/S0019-9958(84)80025-X.

[4] Gérard Boudol & Ilaria Castellani (1987): *On the Semantics of Concurrency: Partial Orders and Transition Systems*. In: Proc. of TAPSOFT'87, pp. 123–137, doi:10.1007/3-540-17660-8_52.

[5] Gérard Boudol & Ilaria Castellani (1988): *Permutation of transitions: An event structure semantics for CCS and SCCS*. In: Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, LNCS 354, Springer, pp. 411–427, doi:10.1007/BFb0013028.

[6] Antonio Bucciarelli & Thomas Ehrhard (1991): *Extensional Embedding of a Strongly Stable Model of PCF*. In: Proc. of ICALP, LNCS 510, Springer, pp. 35–46, doi:10.1007/3-540-54233-7_123.

[7] Antonio Bucciarelli & Thomas Ehrhard (1991): *Sequentiality and Strong Stability*. In: Proc. of LICS, IEEE Computer Society, pp. 138–145, doi:10.1109/LICS.1991.151638.

[8] Rocco De Nicola, Ugo Montanari & Frits W. Vaandrager (1990): *Back and Forth Bisimulations*. In: Proc. of CONCUR, LNCS 458, Springer, pp. 152–165, doi:10.1007/BFb0039058.

[9] Pierpaolo Degano, Rocco De Nicola & Ugo Montanari (1987): *Observational equivalences for concurrency models*. In: Formal Description of Programming Concepts - III, North-Holland, pp. 105–134.

[10] Pierpaolo Degano, Rocco De Nicola & Ugo Montanari (1988): *Partial orderings descriptions and observations of nondeterministic concurrent processes*. In: Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, LNCS 354, Springer, pp. 438–466, doi:10.1007/BFb0013030.

[11] Harald Fecher (2004): *A completed hierarchy of true concurrent equivalences*. Inf. Process. Lett. 89(5), pp. 261–265, doi:10.1016/j.ipl.2003.11.008.

[12] Jean-Yves Girard (1987): *Linear Logic*. Theor. Comput. Sci. 50, pp. 1–102, doi:10.1016/0304-3975(87)90045-4.

[13] J.L. Gischer (1984): *Partial orders and the axiomatic theory of shuffle*. Ph.D. thesis, Stanford University.

[14] Rob J. van Glabbeek (1990): *The Linear Time-Branching Time Spectrum (Extended Abstract)*. In: Proc. of CONCUR, LNCS 458, Springer, pp. 278–297, doi:10.1007/BFb0039066.

[15] Rob J. van Glabbeek (1993): *The Linear Time - Branching Time Spectrum II*. In: Proc. of CONCUR, LNCS 715, Springer, pp. 66–81, doi:10.1007/3-540-57208-2_6.

[16] Rob J. van Glabbeek & Ursula Goltz (2001): *Refinement of actions and equivalence notions for concurrent systems*. Acta Informatica 37(4/5), pp. 229–327, doi:10.1007/s002360000041.

[17] Rob J. van Glabbeek, Ursula Goltz & Jens-Wolfhard Schicke-Uffmann (2013): *On Characterising Distributability*. Logical Methods in Computer Science 9(3), doi:10.2168/LMCS-9(3:17)2013.

[18] Rob J. van Glabbeek & Gordon D. Plotkin (1995): *Configuration Structures*. In: Proc. of LICS, IEEE Computer Society, pp. 199–209.

[19] C. A. R. Hoare (1978): *Communicating Sequential Processes*. Commun. ACM 21(8), pp. 666–677, doi:10.1145/359576.359585.

[20] Brendan D. McKay & Adolfo Piperno: *nauty/Traces software distribution*. Available at http://cs.anu.edu.au/~bdm/nauty and http://pallini.di.uniroma1.it.

[21] Brendan D. McKay & Adolfo Piperno (2014): *Practical graph isomorphism, II*. Journal of Symbolic Computation 60, pp. 94–112, doi:10.1016/j.jsc.2013.09.003.

[22] R. Milner (1989): *Communication and Concurrency*. Prentice Hall.

[23] Mogens Nielsen, Gordon D. Plotkin & Glynn Winskel (1981): *Petri Nets, Event Structures and Domains, Part I*. Theor. Comput. Sci. 13, pp. 85–108, doi:10.1016/0304-3975(81)90112-2.

[24] Mogens Nielsen & P. S. Thiagarajan (2002): *Regular Event Structures and Finite Petri Nets: The Conflict-Free Case*. In: Applications and Theory of Petri Nets, LNCS 2360, Springer, pp. 335–351, doi:10.1007/3-540-48068-4_20.

[25] J. L. Peterson (1977): *Petri Nets*. ACM Computing Surveys 9(3), pp. 223–252, doi:10.1145/356698.356702.

[26] J. L. Peterson (1981): *Petri Net Theory and the Modeling of Systems*. Prentice Hall.

[27] C. A. Petri (1962): *Kommunikation mit Automaten*. Ph.D. thesis, University of Bonn.

[28] Lucia Pomello (1985): *Some equivalence notions for concurrent systems. An overview*. In: Advances in Petri Nets, LNCS 222, Springer, pp. 381–400, doi:10.1007/BFb0016222.

[29] V. R. Pratt (1985): *The pomset model of parallel processes: unifying the temporal and the spatial*. In: Proceedings of Seminar on Concurrency, LNCS 197, Springer, pp. 180–196, doi:10.1007/3-540-15670-4_9.

[30] V. R. Pratt (1986): *Modelling concurrency with partial orders*. International Journal of Parallel Programming 15(1), doi:10.1007/BF01379149.

[31] A. Rabinovich & B. A. Trakhtenbrot (1988): *Behaviour structures and nets*. Fundamenta Informaticae 11(4), pp. 357–404.

[32] Glynn Winskel (1986): *Event Structures*. In: Petri Nets: Central Models and Their Properties (Advances in Petri Nets), LNCS 255, Springer, pp. 325–392, doi:10.1007/3-540-17906-2_31.

[33] Glynn Winskel & Mogens Nielsen (1995): *Models for Concurrency*. In S. Abramsky, Dov M. Gabbay & T. S. E. Maibaum, editors: Handbook of Logic in Computer Science (Vol. 4), Oxford University Press, Inc., New York, NY, USA, pp. 1–148.

Immediate Observation in Mediated Population Protocols

Tobias Prehn
Modelle und Theorie Verteilter Systeme
TU Berlin, Germany
tobias.prehn@tu-berlin.de

Myron Rotter
TU Berlin, Germany
m.rotter@campus.tu-berlin.de

In this paper we analyze the computational power of variants of population protocols (PP), a formalism for distributed systems with anonymous agents having very limited capabilities. The capabilities of agents are enhanced in mediated population protocols (MPP) by recording the states in the edges of the interaction graph. Restricting the interactions to the communication model of immediate observation (IO) reduces the computational power of the resulting formalism. We show that this enhancement and restriction, when combined, yield a model (IOMPP) at least as powerful as the basic PP. The proof requires a novel notion of configurations in the MPP model allowing differentiation of agents and uses techniques similar to methods of analyzing encoding criteria, namely operational correspondence. The constructional part of the proof is generic in a way that all protocols can be translated into the new model without losing the desirable properties they might have besides a stable output. Furthermore, we illustrate how this approach could be utilized to prove our conjecture of IOMPP model being even as expressive as the MPP model. If our conjecture holds, this would result in a sharp characterization of the computational power and reveal the nonnecessity of two-way communication in the context of mediated population protocols.

1 Introduction

Population protocols have been introduced in 2004 as a computational model for passively mobile finite state sensors by Angluin et al. [2, 3]. They feature a finite state space, making them suitable for computation units with very limited capabilities and full anonymity, resulting directly from this restriction. Since the number of possible states that each agent could be in may not grow with the number of participating agents, there is no space for memorizing the ids of already met communication partners or similar constructs. Therefore, the outcome of any binary communication does not depend on whether the participants have communicated before. Another feature is the fully distributed approach of the base version for population protocols that does not need a base station, leader, or scheduler of any kind. The impact of such extensions has been studied [5, 7, 1].

It is well known that predicates computable by population protocols are exactly the semilinear predicates. The first study on the computational power of this model was in 2007 by Angluin et al. [4]. In this context, also several different communication patterns have been modeled in population protocols and their computational power have been studied as well. One of those mechanisms has been the immediate observation model, which is a special kind of one-way communication as opposed to the two-way communication that comes with the base model. The idea is that an agent may observe another agent without it noticing being observed. Clearly the observed agent cannot change its state in such an interaction whereas the observer can use the information given by its own and the observed agent's state. In contrast to stronger mechanisms no synchronization between the communication partners is needed. Consequently the communication in such a model is asynchronous and applicable to a broader variety of systems. With the fully distributed setting in mind the immediate observation communication seems to be a desirable feature. But these qualities come with a price. Protocols with this limitation to the communication can only compute predicates in $\texttt{COUNT}_*$, i.e. predicates that count multiplicities of input values and compare them to previously given thresholds.

Another approach to altering population protocols has been the work of Michail et al. [12]. In their model agents are allowed to store distinct information for different communication partners. To achieve this they extended the base formalism by states for each pair of agents residing in the edges of the interaction graph. Some previous work on mediated population protocols modelled directed interaction graphs with one state per edge [12] and others used undirected graphs where each edge has a state for each of its two endpoints [8]. This extension is a reasonable compromise between maintaining the anonymity of each agent and being able to memorize the already met communication partners. An agent is capable of telling an agent, that it has not yet communicated with, apart from one it has already met. But two other agents being in the same state and with the same communication history are still indistinguishable. Aside from this the edge states can be used for storing several other information. This mediated population protocols are able to compute all symmetric predicates in $NSPACE(n^2)$.

We now present a model in this paper that combines the extended storage possibilities of mediated population protocols and the limited communication model of immediate observation protocols. The computational power of our resulting formalism has to be studied as it is unclear how this extension and restriction interact.

In section 2 the basic formalisms and existing models are defined. We are using a representation of population that allows the distinction of agents from a global point of view. This does not interfere with the anonymity of the agents and is for analysis purposes only. Based on this we define our model of immediate observation mediated population protocols (IOMPP) in section 3. Subsequently we study the computational power of our model in section 4. We take the approach simulating population protocols in immediate observation mediated population protocols in 3.1. Additionally, we give a translation of configurations from one model to the other and define criteria such a translation has to meet for it to express desirable attributes in 4.1. Our work is inspired by and makes use of the encodability criteria stated by Gorla in [11]. To the best of our knowledge this technique is novel to population protocols in the way we utilize it in 4.2 to prove that immediate observation mediated population protocols are at least as expressive as the base model of population protocols. In 4.3 we conjecture that our approach could also be used to show that immediate observation does not restrict the computational power of mediated population protocols. We conclude our paper by a discussion on the given results and possible application of the used techniques in section 5. We also give an outlook on future work and open questions.

2 Technical preliminaries

First we introduce populations, which form the base of all population protocols. They are often modeled as multisets to emphasize the indistinguishability of the participating agents. We will use vectors where each entry represents the state of a specific agent, because we want to efficiently compare populations in different models from a global viewpoint. Note that this will not give agents a distinct id they could make use from their local point of view. A different kind of vector representation can be found in [10] and is not to be confused with ours. They use vectors where each entry describes for an agent state the multiplicities of agents in that state. Their vector representation efficiently stores sets of agents with their states, making it easy to identify equivalent protocol states. In contrast to this, our representation can be used to compare sets of agents in population protocols with sets of agents in extended variants like mediated population protocols.

Definition 1 (Populations)**.** Let A be a nonempty finite set and $n \in \mathbb{N}$. Then A^n describes the set of n-tuples over A also referred to as vectors of length n. A population over A, denoted by $POP(A)$, is the set of all vectors of arbitrary but finite length over A. If $v \in A^n$ we use $|v| = n$ to denote the length of v and $(v)_i$ to reference the i^{th} element of v (with $i \in \mathbb{N}^+_{\leq n}$).

Next we define how calculations in population protocols are modeled from a local point of view. Each agent has the same set of states and rules how to change states according to a communication partner's state. Additionally, functions to map input to an initial state and a state to some output are defined to be identical for each agent.

Definition 2 (Population Protocols [6])**.** A population protocol P is a 5-tuple $P = (Q, \Sigma, I, O, \delta)$ where Q is a finite set of agent states, Σ a finite input alphabet, $I : \Sigma \to Q$ describes the input function, $O : Q \to \{0,1\}$ is the output function, and $\delta : Q^2 \to Q^2$ is referred to as transition function and describes all possible pairwise interactions. We also making use of a set representation of $\delta \subseteq Q^4$ whenever we write $t = (p,q) \to_\delta (p',q')$ referring to a specific transition $t \in \delta$.

To analyze protocols we need a global perspective. Here states are configurations holding state information for each agent. This kind of global view has been used to study the computational power of population protocols [4].

Definition 3 (Global Protocols [4])**.** Let $P = (Q, \Sigma, I, O, \delta)$ be a population protocol. The global protocol to P is a 5-tuple $G_P = (\mathcal{C}, \Sigma, \mathcal{I}, \mathcal{O}, \longrightarrow)$ where $\mathcal{C} = POP(Q)$ is the set of configurations, i.e., vectors of agent states Q, $\mathcal{I} : POP(\Sigma) \to \mathcal{C}$ maps input vectors to initial configurations, $\mathcal{O} : \mathcal{C} \to \{0,1,\perp\}$ maps configurations to outputs, and $\longrightarrow : \mathcal{C} \to \mathcal{C}$ is the global transition function with $\longrightarrow^*$ being its reflexive and transitive closure. For $C, C' \in \mathcal{C}$ it holds that $C \longrightarrow C'$ iff there is a transition $t \in \delta$ and $i, j \in \mathbb{N}^+$ with $i \neq j$ such that $t = ((C)_i, (C)_j) \to_\delta ((C')_i, (C')_j)$ and $(C)_k = (C')_k$ for every $k \in \mathbb{N}^+ \setminus \{i, j\}$. We also write $C \xrightarrow{t_{i,j}} C'$ and call agent i the *initiator* of t and j the *responder* or (if the state of i is not changed by t) *observer*. The global input function takes use of I to get an agent state for each single value in its input and $\mathcal{O}$ aggregates the outputs of the agents according to O. It holds that $\mathcal{O}(C) = x \in \{0,1\}$ iff $O((C)_i) = x$ for each $i \in \mathbb{N}^+_{\leq |C|}$ and $\mathcal{O}(C) = \perp$ in every other case. When the underlying protocol P is clear from the context we often omit the index of G_P and simply state that G is the global protocol to P.

Based on a global protocol we can describe what it means for a protocol to compute some predicate. For this we need to define executions and fairness.

Definition 4 (Computation)**.** Let $G = (\mathcal{C}, \Sigma, \mathcal{I}, \mathcal{O}, \longrightarrow)$ be a global protocol. A configuration $C \in \mathcal{C}$ is output stable with output $x \in \{0,1\}$ iff $\mathcal{O}(C') = x$ for each $C' \in \mathcal{C}$ with $C \longrightarrow^* C'$. We call a sequence of configurations $C_0, C_1, C_2, \dots \in \mathcal{C}$ with $C_i \longrightarrow C_{i+1}$ for each $i \in \mathbb{N}$ an execution. An execution is fair iff for each $C \in \mathcal{C}$ with $C_i = C$ for infinitely many $i \in \mathbb{N}$ it holds that if there is a transition $C \longrightarrow C'$ then also $C_j = C'$ for infinitely many $j \in \mathbb{N}$. A population protocol P is well-specified if for each input *Inp*, it holds that all fair executions of P starting in $\mathcal{I}(Inp)$ reach a configuration that is output stable. P computes a predicate if this reached configuration is output stable with output 1 if *Inp* satisfies the predicate and with output 0 otherwise.

In the context of population protocols several communication mechanisms have been studied [4]. Immediate observation is one of those mechanisms. It reduces the class of computable predicates to predicates counting multiplicities of input values $\texttt{COUNT}_*$. To model this kind of communication, restrictions to the allowed form of transitions are made.

Definition 5 (Immediate Observation)**.** Let $P = (Q, \Sigma, I, O, \delta)$ be a population protocol. P is an immediate observation protocol, if there is no transition that changes the state of the initiator. In other words, all transitions $t \in \delta$ have to be of the form $t = (p,q) \to_\delta (p,q')$.

Another extension to population protocols is the mediated variant. The idea is to introduce states in all edges of the communication graph. Since the most general graph is the complete graph, each pair of agents is given such a state. In the context of an immediate observation communication mechanism, it is not reasonable to assume a storage that both agents can write to. Therefore, we introduce a pair of edge states for each pair of agents. Edge states are always initialized with the same value.

Definition 6 (Mediated Population Protocols [12])**.** A mediated population Protocol P is a 7-tuple $P = (Q, \Sigma, S, s_0, I, O, \delta)$ where Q, Σ, I and O are analogous to population protocols. The set of edge states S includes the initial edge state $s_0 \in S$ and the transition function $\delta : (Q \times S)^2 \to (Q \times S)^2$ incorporates the edge states for each pair of agents.

Configurations in mediated population protocols cannot be represented by simple vectors. We need to introduce matrices as configurations containing the agent states on the diagonal and the states of the edge between agents a and b in fields $C_{a,b}$ (side of agent a) and $C_{b,a}$ (side of agent b).

Definition 7 (Mediated Populations)**.** $A^{n \times n}$ describes the set of square matrices over A of size $n \times n$. A mediated population over A denoted by $POP_{\mathrm{M}}(A)$ is the set of all matrices of arbitrary but finite length over A. If $m \in A^{n \times n}$ we use $(m)_{i,j}$ to reference the element of m at column i and row j with $i, j \in \mathbb{N}^+_{\leq n}$ and $|m| = n$ to denote the length as well as the height of a square matrix m.

We can now proceed with lifting our global protocol definitions to represent mediated population protocols as well.

Definition 8 (Global Protocols for Mediated Population Protocols)**.** Let $P = (Q, \Sigma, S, s_0, I, O, \delta)$ be a mediated population protocol. The global protocol to P is again a 5-tuple $G = (\mathcal{C}, \Sigma, \mathcal{I}, \mathcal{O}, \longrightarrow)$. In contrast to global protocols for simple population protocols $\mathcal{C} = POP_{\mathrm{M}}(Q)$ is the set of configurations and $\mathcal{I} : POP(\Sigma) \to \mathcal{C}$ maps input vectors to initial configurations, initializing the diagonal fields with the corresponding agent states and every other field with s_0. The output function $\mathcal{O} : \mathcal{C} \to \{0, 1, \bot\}$ ignores all fields not on the diagonal and $\longrightarrow : \mathcal{C} \to \mathcal{C}$ now also changes the respective edge states. For $C, C' \in \mathcal{C}$ it holds that $C \longrightarrow C'$ iff there is a transition $t \in \delta$ and $i, j \in \mathbb{N}^+$ with $i \neq j$ such that $t = ((C)_{i,i}, (C)_{i,j}, (C)_{j,j}, (C)_{j,i}) \to_\delta ((C')_{i,i}, (C')_{i,j}, (C')_{j,j}, (C')_{j,i})$ and $(C')_{k,l} = (C')_{k,l}$ for every $k, l \in \mathbb{N}^+ \setminus \{i, j\}$.

3 Modelling immediate observation in mediated population protocols

From the technical preliminaries in section 2 we can easily combine the models for mediated population protocols and immediate observation conform communication. We get our model of population protocols with two edge states in every edge, one per communication partner, and transitions that keeps the states of the initiator unaltered and changes the states of the observer.

Definition 9 (Immediate Observation Mediated Population Protocols)**.** An immediate observation mediated population protocol P is a 7-tuple $P = (Q, \Sigma, S, s_0, I, O, \delta)$ where Q is a finite set of agent states, Σ a finite input alphabet, $I : \Sigma \to Q$ describes the input function, $O : Q \to \{0, 1\}$ is the output function, and $\delta : (Q \times S)^2 \to (Q \times S)^2$ is referred to as transition function and describes all possible pairwise interactions. We also making use of a set representation of $\delta \subseteq (Q \times S)^4$ whenever we write $t = (p, s, q, r) \to_\delta (p', s', q', r')$ referring to a specific transition $t \in \delta$. Since our model uses the immediate observation communication mechanism, all transitions $t = (p, s, q, r) \to_\delta (p', s', q', r')$ have to satisfy $p = p'$ and $s = s'$.

3.1 Simulating population protocols by immediate observation mediated population protocols

We can now simulate protocols in the basic population protocol model by immediate observation mediated population protocols. The main idea is to split every two-way communication with an initiator and a responder into 4 steps. Two steps are required to signal the request and the acknowledgement of a communication and another two steps are needed to finish the communication resolving all pending state changes. Additionally, a reset transition is given for the case of an unsuccessful communication.

Simulation 10. Let $P = (Q, \Sigma, I, O, \delta)$ be a population protocol. The following immediate observation mediated population protocol P' simulates the protocol P and is given by the tuple $(Q', \Sigma', S', s'_0, I', O', \delta')$ where

$$\begin{aligned}
\Sigma' &:= \Sigma, \\
Q' &:= \{L, U\} \times Q, \\
S' &:= \{s_{init}, s_{ponr}\} \cup Q, \\
s'_0 &:= s_{init}, \\
I'(\sigma) &:= (U, I(\sigma)) \text{ for all } \sigma \in \Sigma, \\
O'(l, q) &:= O(q) \text{ for all } (l, q) \in Q'.
\end{aligned}$$

The states of the agents are of the form (l, q) where $l \in \{L, U\}$ indicates whether the agent is locked or unlocked and $q \in Q$ is the computation state according to the original population protocol P. For easier referencing we call the first component of an agent state (l, q) the locking state l of this agent and the second component its computation state q. We use the formulation of an agent being locked whenever its locking state is L and say this agent is unlocked otherwise. W.l.o.g. we assume that $\{s_{init}, s_{ponr}\} \cap Q = \emptyset$. The input function I maps each input symbol $\sigma \in \Sigma$ to the state $(U, I(\sigma))$. The output function O' maps each state $(l, q) \in Q$ to $O(q)$ independent of the locking indicator l. We specify for each transition $t = (p, q) \to_\delta (p', q')$ of δ with $p, q, p', q' \in Q$ the following transitions for δ'.

$$\begin{aligned}
t^{(1)} = ((U,p), s_{init}, (U,q), s_{init}) &\to_{\delta'} ((U,p), s_{init}, (L,q'), q) && (1) \\
t^{(2)} = ((L,q'), q, (U,p), s_{init}) &\to_{\delta'} ((L,q'), q, (L,p'), s_{ponr}) && (2) \\
t^{(3)} = ((L,p'), s_{ponr}, (L,q'), q) &\to_{\delta'} ((L,p'), s_{ponr}, (U,q'), s_{init}) && (3) \\
t^{(4)} = ((x,y), s_{init}, (L,p'), s_{ponr}) &\to_{\delta'} ((x,y), s_{init}, (U,p'), s_{init}) && (4) \\
& \quad \text{for every } (x,y) \in Q' \\
t^{(5)} = ((x,y), z, (L,q'), q) &\to_{\delta'} ((x,y), z, (U,q), s_{init}) && (5) \\
& \quad \text{for every } (x,y) \in Q' \\
& \quad \text{and } z \in S' \setminus \{s_{ponr}\}
\end{aligned}$$

The locking state of each agent prohibits simultaneous participation in several different communications. Whenever an agent took part in a two-way communication t in the original protocol, it could be the observer of a transition of type (1) in the simulation. A $t^{(1)}$ transition locks the observing agent and puts its old computation state in the edge state this agent controls on the edge with the observed agent. This has two reasons: First it signals the interest in a communication with the other agent and second it backups the old state for a potential future reset. If the other agent observes the change in the edge state, it may signal the acknowledgement of requested communication by locking itself, changing its computation state according to the transition t and putting s_{ponr} in its edge state. This is achieved by transition $t^{(2)}$. Now the two agents have to reset their edge states to s_{init} and unlock themselves. The agent mimicking the responder of the original transition t starts by taking $t^{(3)}$, followed by the simulator of the original initiator taking $t^{(4)}$. If a communication was not successful, either because the initiators surrogate has taken an other transition with another agent in the meantime or because responder simulating agent observes its partner before it could acknowledge the communication, $t^{(5)}$ is taken. This transition assures that in the described cases an agent can give up on a communication attempt and reset its state, readying itself for another attempt, potentially with a different partner.

Note that if a transition $t = (p,q) \rightarrow_\delta (p,q')$ is already immediate observation compliant we do not need to add the whole set of transitions. We could instead add a slightly altered version of the original transition as follows.

$$t^{(6)} = ((U,p), s_{init}, (U,q), s_{init}) \quad \rightarrow_{\delta'} \quad ((U,p), s_{init}, (U,q'), s_{init}) \tag{6}$$

Clearly the result would be the same. This kind of transition is only needed if the simulation needs to be more efficient in the sense of steps needed to get to an output stable configuration. We will therefore omit this type of transitions in our analyses.

Observation 11 (Output Changing Transitions)**.** By definition of O' the output of an agent only depends on its computation state. As transitions $t^{(3)}$, $t^{(4)}$ do not change the computation states, only transitions $t^{(1)}$, $t^{(2)}$, and $t^{(5)}$ can have an impact on the output of an agent. Since Simulation 10 is an immediate observation protocol, this agent has to be the observer of such transitions.

Observation 12 (Number of Started Conversations)**.** Every agent has at most one started and not yet concluded conversation at any point in time. Starting a conversation by taking transition $t^{(1)}$ as responder brings an agent to a locked state. Therefore, no other conversation can be started or acknowledged by this agent until the conversation is concluded with transition $t^{(3)}$ or aborted with transition $t^{(5)}$. Acknowledging a conversation by taking transition $t^{(2)}$ as responder also brings an agent to a locked state. Again no other conversation can be started or acknowledged by this agent until the conversation is concluded with transition $t^{(4)}$ or aborted with transition $t^{(5)}$.

Observation 13 (Point of no Return)**.** Every occurrence of transition $t^{(2)}_{i,j}$ with acting agents i and j is eventually followed by transitions $t^{(3)}_{j,i}$ and $t^{(4)}_{i,j}$. After execution of $t^{(2)}_{i,j}$ agent i is locked with $q \in Q$ in its edge state to j and agent j is locked with s_{ponr} in its edge state to i. From Observation 12 we know that neither i nor j can be observer of any transition with some agent different from i and j. From the transitions with i and j only $t^{(3)}_{j,i}$ is enabled and will be taken at some point because of the fairness assumptions in population protocols. After that agent j is still locked with s_{ponr} in its edge state to i. Therefore, j can only be observer of transitions $t^{(3)}$,$t^{(4)}$, or $t^{(5)}$. Again from Observation 12 we know that only $t^{(4)}_{i,j}$ is possible.

4 Computational power

We now show that our model can compute all predicates computable in population protocols by giving a translation, that relates configurations from a protocol to configurations from its simulation representing the same state of computation. Additionally, we identify requirements imposed on such a translation to be helpful in proving the equality of computed predicates. We will ultimately show how this proof is executed.

4.1 Translation and criteria

We provide a translation, that constructs configurations in mediated population protocols from configurations of population protocols by giving all agents an unlocked state and setting all edges to the neutral s_{init} state.

Definition 14 (Translation of Configurations)**.** Let $P = (Q, \Sigma, I, O, \delta)$ be a population protocol and $P' = (Q', \Sigma', S', s'_0, I', O', \delta')$ a immediate observation mediated population protocol constructed from P using Simulation 10. By $[\![\cdot]\!] : POP(Q) \rightarrow POP_{\mathrm{M}}(Q')$ we denote the translation of configurations $C \in POP(Q)$ in the population protocol into configurations $D \in POP_{\mathrm{M}}(Q')$ from the mediated population protocol. This translation is defined as follows:

$$[\![(q_0, q_1, \ldots, q_n)]\!] = \begin{pmatrix} (U, q_1) & s_{init} & \cdots & s_{init} \\ s_{init} & (U, q_2) & \ddots & \vdots \\ \vdots & \ddots & \ddots & s_{init} \\ s_{init} & \cdots & s_{init} & (U, q_n) \end{pmatrix}$$

From the criteria for *good encodings* defined by Gorla [11] we adopt the notion of operational correspondence.

Definition 15 (Operational Correspondence)**.** A translation $[\![\cdot]\!] : POP(Q) \rightarrow POP_{\mathrm{M}}(Q')$ is operationally corresponding if it is

(1) (operationally) complete, i.e., for all $C, C' \in POP(Q)$ with $C \longrightarrow^* C'$ it holds that $[\![C]\!] \longrightarrow^* [\![C']\!]$, and

(2) (operationally) sound, i.e., for all $C \in POP(Q)$ and $D \in POP_{\mathrm{M}}(Q')$ with $[\![C]\!] \longrightarrow^* D$ there exists a $C' \in POP(Q)$ with $D \longrightarrow^* [\![C']\!]$ and $C \longrightarrow^* C'$.

If our translation in Def. 14 instantiated with concrete population protocol P and mediated population protocol P' is operationally corresponding, we get that every configuration reachable in P is also reachable in P' and vice versa.

Definition 16 (Input/Output Correspondence)**.** Let P be a population protocol, P' be a mediated population protocol and G, G' be the global protocols to P and P' respectively. A translation $[\![\cdot]\!] : POP(Q) \rightarrow POP_{\mathrm{M}}(Q')$ is I/O corresponding if it is

(1) input corresponding, i.e., for all $V \in POP(\Sigma)$ it holds that $[\![\mathcal{I}(V)]\!] = \mathcal{I}'(V)$, and

(2) output corresponding, i.e., for all $C \in POP(Q)$ it holds that $\mathcal{O}(C) = \mathcal{O}'([\![C]\!])$.

Input/Output Correspondence gives us the assurance that input and output functions of the protocols related by a translation behave in a similar way. If it holds, translating an input configuration of the original protocol or directly using the input function of the corresponding mediated population protocol yields the same result. Additionally, configurations are always translated into configurations with the same output.

Definition 17 (Output Stability Preservation)**.** Let P be a population protocol, P' be a mediated population protocol and G, G' be the global protocols to P and P' respectively. A translation $[\![\cdot]\!] : POP(Q) \rightarrow POP_{\mathrm{M}}(Q')$ is output stability preserving if for each $C \in POP(Q)$ it holds that $[\![C]\!]$ is output stable iff C is output stable.

From the output stability preservation we get that each output stable configuration is translated into a configuration also being output stable.

Lemma 18. *Let $P = (Q,\Sigma,I,O,\delta)$ be a population protocol and $P' = (Q',\Sigma',S',s'_0,I',O',\delta')$ a mediated population protocol. If there is a translation $[\![\cdot]\!] : POP(Q) \to POP_M(Q')$ that is operationally corresponding, input/output corresponding, and output stability preserving, then P and P' compute the same predicate.*

Proof. Since the translation is input corresponding every initial configuration in P has a corresponding initial configuration in P'. From the operational correspondence we know that a configuration is reachable from an initial configuration in P' iff it has a corresponding configuration reachable from the corresponding initial configuration in P. With output correspondence both configurations clearly have the same output and, since the translation is output stability preserving, the output is either stable in both configurations or in none. Therefore, the protocol P calculates the same semilinear predicate as P'. □

4.2 All semilinear predicates can be computed by immediate observation mediated population protocols

Lemma 19. *The translation $[\![\cdot]\!]$ given in Def. 14 is operationally corresponding for any population protocol $P = (Q,\Sigma,I,O,\delta)$ and the mediated population protocol $P' = (Q',\Sigma',S',s'_0,I',O',\delta')$ constructed using Simulation 10.*

Proof. To prove operational completeness assume that $C, C' \in POP(Q)$ with $C \longrightarrow^* C'$. Since $\longrightarrow^*$ is defined as reflexive-transitive closure of $\longrightarrow$ we get that $C_1, C_2, \ldots, C_n \in POP(Q)$ exist with $C \longrightarrow C_1 \longrightarrow C_2 \longrightarrow \ldots \longrightarrow C_n \longrightarrow C'$. We can always simulate a step $C_i \longrightarrow C_{i+1}$ in P by making 4 steps in P'. Assume that the step is due to transition $t \in \delta$ and agents at a and b are acting as initiator and responder respectively in $C_i \xrightarrow{t_{a,b}} C_{i+1}$. This can be simulated as $[\![C_i]\!] \xrightarrow{t^{(1)}_{a,b}} C_i^2 \xrightarrow{t^{(2)}_{b,a}} C_i^3 \xrightarrow{t^{(3)}_{a,b}} C_i^4 \xrightarrow{t^{(4)}_{b,a}} [\![C_{i+1}]\!]$ and thus $[\![C_i]\!] \longrightarrow^4 [\![C_{i+1}]\!]$ holds. Thus, $[\![C]\!] \longrightarrow^4 [\![C_1]\!] \longrightarrow^4 \ldots \longrightarrow^4 [\![C_n]\!] \longrightarrow^4 [\![C']\!]$ exemplifies $[\![C]\!] \longrightarrow^* [\![C']\!]$.

To prove operational soundness assume that $C \in POP(Q)$ and $D \in POP_M(Q')$ with $[\![C]\!] \longrightarrow^* D$. We construct C' as follows. For this assume $(D)_{i,i} = (l_i, d_i)$.

$$(C')_i = \begin{cases} g & \text{, if there is exactly one } j \in \mathbb{N}^+_{\leq |D|, \neq i} \text{ such that } (D)_{i,j} = g \text{ and } (D)_{j,i} \neq s_{ponr} \\ d_i & \text{, otherwise} \end{cases}$$

We can show that $D \longrightarrow^* [\![C']\!]$ by taking the appropriate transitions for all $i, j \in \mathbb{N}^+_{\leq |D|}$ with $i \neq j$ as follows.

if $(D)_{i,j} = q$	and	$(D)_{j,i} = s_{ponr}$	take transitions $t^{(3)}_{j,i}, t^{(4)}_{i,j}$
if $(D)_{i,j} = s_{init}$	and	$(D)_{j,i} = s_{ponr}$	take transition $t^{(4)}_{i,j}$
if $(D)_{i,j} = q$	and	$(D)_{j,i} \neq s_{ponr}$	take transition $t^{(5)}_{j,i}$
	otherwise		do nothing

Since $[\![C]\!] \longrightarrow^* D$ by assumption and $D \longrightarrow^* [\![C']\!]$ we get that $[\![C]\!] \longrightarrow^* [\![C']\!]$. We can construct $C \longrightarrow C_1 \longrightarrow C_2 \longrightarrow \ldots \longrightarrow C'$ from the path $[\![C]\!] \longrightarrow D_1 \longrightarrow D_2 \longrightarrow \ldots \longrightarrow [\![C']\!]$ as follows. Whenever there is a step $D_x \xrightarrow{t^{(2)}_{i,j}} D_y$ in this path, take transition $C_v \xrightarrow{t_{i,j}} C_w$ in the path of $C \longrightarrow^* C'$. By Observation 12 we get that each such step of type (2) is eventually followed by transitions of type (3) and (4) and since $[\![C']\!]$ has only edge states s_{init}, this has to happen before $[\![C']\!]$ is reached. Therefore, there exists a $C' \in POP(Q)$ with $D \longrightarrow^* [\![C']\!]$ and $C \longrightarrow^* C'$. □

Lemma 20. *The translation $[\![\cdot]\!]$ given in Def. 14 is I/O corresponding for any population protocol $P = (Q, \Sigma, I, O, \delta)$ and the mediated population protocol $P' = (Q', \Sigma', S', s'_0, I', O', \delta')$ constructed using Simulation 10.*

Proof. The input function I' of P' makes use of the input function I of P by putting an agent in the unlocked state $(U, I(\sigma))$ iff the same agent would be in state $I(\sigma)$ in P. Also every edge state is initialized with s_{init}. This matches $[\![\cdot]\!]$ that translates each state q to (U, q) and sets every edge state to s_{init}. Thus, $[\![\mathcal{I}(V)]\!] = \mathcal{I}'(V)$ and $[\![\cdot]\!]$ is input corresponding.

The output function O' of P' makes use of the output function O of P by ignoring the locking state of an agent and giving back the output of O for the computation state. As seen above, $[\![\cdot]\!]$ translates every agents state q to a tuple with q being the second component i.e. its computation state. For each agent a it holds that $O((C)_a) = O'\left((D)_{a,a}\right)$. As $\mathcal{O}$ aggregates the outputs of all agents, which are the same in P and P', $\mathcal{O}(C) = \mathcal{O}'([\![C]\!])$ and consequently $[\![\cdot]\!]$ is output corresponding. □

Lemma 21. *The translation $[\![\cdot]\!]$ given in Def. 14 is output stability preserving for any population protocol $P = (Q, \Sigma, I, O, \delta)$ and the mediated population protocol $P' = (Q', \Sigma', S', s'_0, I', O', \delta')$ constructed using Simulation 10.*

Proof. Let C be a configuration that is output stable in P. Assume towards contradiction $[\![C]\!]$ is not output stable in P'. From the definition of output stability follows that a configuration D exists with $\mathcal{O}'([\![C]\!]) \neq \mathcal{O}'(D)$ and D is reachable from $[\![C]\!]$, i.e. there is a path $[\![C]\!] \longrightarrow D_1 \longrightarrow D_2 \longrightarrow \ldots \longrightarrow D$. W.l.o.g assume that D is the first such configuration in this path, i.e. $\mathcal{O}'([\![C]\!]) = \mathcal{O}'(D_i)$ for each such D_i. Since $\mathcal{O}'$ aggregates the values of O' for each agent and because P' is an immediate observation protocol, there has to be a single agent a that has changed its output because of the transition leading to D. From Observation 11 we know that this transition has to be of type (1), (2), or (5) and a has to be its observer.

If it is (1) or (2) we can construct a configuration in the same way as C' was constructed in the proof of operational soundness for Lemma 19. This C' is reachable from C in P and $[\![C']\!]$ is reachable from D in P'. Note that on the path from D to $[\![C']\!]$ agent a is never an observer of any transition with type (1), (2), or (5) and therefore does not change its output. Because our translation maintains outputs for each agent $O((C)_a) = O'\left(([\![C]\!])_{a,a}\right) \neq O'\left((D)_{a,a}\right) = O'\left(([\![C']\!])_{a,a}\right) = O((C')_a)$ holds. This results in $\mathcal{O}(C) \neq \mathcal{O}(C')$, a contradiction to C being output stable.

If the transition agent a took was of type (5), there has to be a configuration along the path from $[\![C]\!]$ to D where agent a took a transition of type (1). Observation 12 states that there has to be such a transition in advance and by the definition of $[\![\cdot]\!]$ this has to be after $[\![C]\!]$. But if the transition of type (5) changes the output of a, the corresponding type (1) transition must also have changed it, contradicting our assumption of D being the first configuration with an output different from $\mathcal{O}'([\![C]\!])$. This is due to a type (5) transition resetting the computation state of an agent back to the state it had before the corresponding type (1) transition.

For the other direction assume towards contradiction $[\![C]\!]$ is output stable in P' and C is not output stable in P. Then there exists a configuration $\overline{C}$ that is reachable from C with $\mathcal{O}(C) \neq \mathcal{O}\left(\overline{C}\right)$. Because $[\![\cdot]\!]$ is operationally complete by Lemma 19 it holds that $[\![\overline{C}]\!]$ is reachable from $[\![C]\!]$. From the output correspondence of $[\![\cdot]\!]$ in Lemma 20 follows that $\mathcal{O}'([\![C]\!]) = \mathcal{O}(C) \neq \mathcal{O}\left(\overline{C}\right) = \mathcal{O}'\left([\![\overline{C}]\!]\right)$. This is a contradiction to the output stability of $[\![C]\!]$ in P'. □

Theorem 22. *For any population protocol $P = (Q, \Sigma, I, O, \delta)$, the immediate observation mediated population protocol P' constructed from P using Simulation 10 calculates the same semilinear predicate.*

Proof. We have shown that $[\![\cdot]\!]$ from Definition 14 is operationally corresponding, I/O corresponding and output stability preserving in Lemmas 19, 20, and 21. By Lemma 18 the statement directly follows. $\square$

Corollary 23. *Immediate observation mediated population protocols can compute every semilinear predicate and are therefore at least as expressive as population protocols.*

4.3 Immediate observation does not restrict the computational power of mediated population protocols

The approach in the previous sections can be used to prove that two-way communication in mediated population protocols does not add to the computational power of the model. To achieve this we define a simulation of mediated population protocols into the variant with immediate observation communication.

Simulation 24. Let $P = (Q, \Sigma, S, s_0, I, O, \delta)$ be a mediated population protocol. The following immediate observation mediated population protocol P' simulates the protocol P and is given by the tuple $(Q', \Sigma', S', s'_0, I', O', \delta')$ where

$$\begin{aligned}
\Sigma' &:= \Sigma, \\
Q' &:= \{L, U\} \times Q, \\
S' &:= (\{s_{init}, s_{ponr}\} \cup (Q \times S)) \times S, \\
s'_0 &:= (s_{init}, s_0), \\
I'(\sigma) &:= (U, I(\sigma)) \text{ for all } \sigma \in \Sigma, \\
O'(l,q) &:= O(q) \text{ for all } (l,q) \in Q'.
\end{aligned}$$

W.l.o.g. we assume that $\{s_{init}, s_{ponr}\} \cap (Q \cup S) = \emptyset$. In contrast to Simulation 10 the edge state has two components. The first component again signals the current state of the simulated communication and serves as a backup for the condition prior to the communication. Here we need to save both, computation state and edge state. The second component represents the actual edge state present in the original protocol. We specify for each transition $t = (p, r, q, s) \rightarrow_\delta (p', r', q', s')$ of δ with $p, q, p', q' \in Q$ and $r, s, r', s' \in S$ the following transitions for δ'.

$$\begin{aligned}
t^{(1)} = ((U,p),\ (s_{init},r),\ (U,q),\ (s_{init},s)) &\rightarrow_{\delta'} ((U,p),\ (s_{init},r),\ (L,q'),\ ((q,s),s')) \\
t^{(2)} = ((L,q'),\ ((q,s),s'),\ (U,p),\ (s_{init},r)) &\rightarrow_{\delta'} ((L,q'),\ ((q,s),s'),\ (L,p'),\ (s_{ponr},r')) \\
t^{(3)} = ((L,p'),\ (s_{ponr},r'),\ (L,q'),\ ((q,s),s')) &\rightarrow_{\delta'} ((L,p'),\ (s_{ponr},r'),\ (U,q'),\ (s_{init},s')) \\
t^{(4)} = ((x,y),\ (s_{init},s'),\ (L,p'),\ (s_{ponr},r')) &\rightarrow_{\delta'} ((x,y),\ (s_{init},s'),\ (U,p'),\ (s_{init},r')) \\
&\qquad \text{for every } (x,y) \in Q' \\
t^{(5)} = ((x,y),\ (v,w),\ (L,q'),\ ((q,s),s')) &\rightarrow_{\delta'} ((x,y),\ (v,w),\ (U,q),\ (s_{init},s)) \\
&\qquad \text{for every } (x,y) \in Q' \\
&\qquad \text{and } (v,w) \in S' \setminus \{(s_{ponr},r')\}
\end{aligned}$$

This simulation follows the same ideas as the Simulation 10. The only difference is the edge state of the original protocol that needs to be taken into account by transitions of the simulation and that needs to be backed up for possible future resets. We can now give a translation similar to Definition 14 required for our line of argumentation.

Definition 25 (Translation of Mediated Configurations)**.** Let $P = (Q, \Sigma, S, s_0, I, O, \delta)$ be a mediated population protocol and $P' = (Q', \Sigma', S', s_0', I', O', \delta')$ a immediate observation mediated population protocol constructed from P using Simulation 24. By $[\![\cdot]\!] : POP_{\mathrm{M}}(Q) \to POP_{\mathrm{M}}(Q')$ we denote the translation of configurations $C \in POP_{\mathrm{M}}(Q)$ in the population protocol into configurations $D \in POP_{\mathrm{M}}(Q')$ from the mediated population protocol. This translation is defined as follows:

$$\left[\!\!\left[\begin{pmatrix} q_1 & s_{2,1} & \cdots & s_{n,1} \\ s_{1,2} & q_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & s_{n,n-1} \\ s_{1,n} & \cdots & s_{n-1,n} & q_n \end{pmatrix}\right]\!\!\right] = \begin{pmatrix} (U,q_1) & (s_{init},s_{2,1}) & \cdots & (s_{init},s_{n,1}) \\ (s_{init},s_{1,2}) & (U,q_2) & \ddots & \vdots \\ \vdots & \ddots & \ddots & (s_{init},s_{n,n-1}) \\ (s_{init},s_{1,n}) & \cdots & (s_{init},s_{n-1,n}) & (U,q_n) \end{pmatrix}$$

With this simulation and translation we conjecture that each mediated population protocol can be simulated by a immediate observation mediated population protocol that shares several attributes, especially computing the same predicates.

Conjecture 26. For any mediated population protocol $P = (Q, \Sigma, S, s_0, I, O, \delta)$, the immediate observation mediated population protocol $P' = (Q', \Sigma', S', s_0', I', O', \delta')$ constructed from P using Simulation 24 calculates the same semilinear predicate.

5 Conclusion and future work

We have given a proof for the model of immediate observation mediated population protocols to compute all semilinear predicates. Thus they are as least as powerful in computation as population protocols. Additionally, we have given arguments why we believe this model is even equivalent to the model of mediated population protocols with two-way communication. Consequently allowing the initiator of a transition to change its agent and edge states does not contribute to the computational power. The proof of our Conjecture 26 can hopefully be done in our future research.

Our approach asks for a simulation and a translation which might seem overly complicated for a proof of equal computational power. But additionally several other attributes, besides the computation of the same predicate, carry over from the one protocol to the other if our Simulation 10 and translation from Definition 14 are used. Consider for example livelock freedom, i.e. no configuration is reached that has no successor besides itself. The simulation can reach a livelock iff the original protocol can reach such a configuration. This can easily be derived from the operational correspondence in Definition 15. In the context of protocols computing some predicate, a livelock is only possible if an output is reached. Otherwise the requirements for a well-specified protocol are not met. A livelock can be a desirable state as the computation can clearly be stopped in such a configuration. If the protocol does something else than computing a predicate, livelocks can be even more important to be reached or avoided, depending on the situation.

Another example is the analysis of a required communication structure. Whereas some protocols need a full interaction graph to carry out a computation, a path structure would suffice for others to get a correct result. The interaction graphs supporting a protocol do also support its simulation.

As a last example consider failure resistance [9]. If a protocol is designed to tolerate a certain number and type of faults, the simulation of this protocol could be capable of a comparable behaviour. This however depends on the type of failure and the chosen strategy to handle it. Crash failures, where an agent may leave the population at any time, should be manageable in the simulation with the same mechanisms as the original protocol did. Message losses could lead to new problems in the simulation like deadlocked communication partners. Some error handling and error masking strategies could lead to the number of failures tolerable by the simulation being reduced in contrast to the original protocol.

A study on desirable attributes and how they carry over from one protocol to another by our simulation is something we wish to address in the future.

References

[1] Dan Alistarh, Rati Gelashvili & Milan Vojnović (2015): *Fast and Exact Majority in Population Protocols*. In: *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, PODC '15, ACM, New York, NY, USA, pp. 47–56, doi:10.1145/2767386.2767429.

[2] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer & René Peralta (2004): *Computation in networks of passively mobile finite-state sensors*. In: *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, ACM, pp. 290–299, doi:10.1145/1011767.1011810.

[3] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer & René Peralta (2006): *Computation in networks of passively mobile finite-state sensors*. *Distributed Computing* 18(4), pp. 235–253, doi:10.1007/s00446-005-0138-3.

[4] Dana Angluin, James Aspnes, David Eisenstat & Eric Ruppert (2007): *The computational power of population protocols*. *Distributed Computing* 20(4), pp. 279–304, doi:10.1007/s00446-007-0040-2.

[5] Dana Angluin, James Aspnes, Michael J. Fischer & Hong Jiang (2008): *Self-stabilizing Population Protocols*. *ACM Trans. Auton. Adapt. Syst.* 3(4), pp. 13:1–13:28, doi:10.1145/1452001.1452003.

[6] James Aspnes & Eric Ruppert (2009): *An Introduction to Population Protocols*, pp. 97–120. Springer Berlin Heidelberg, Berlin, Heidelberg, doi:10.1007/978-3-540-89707-1_5.

[7] Shantanu Das, Giuseppe Antonio Di Luna, Paola Flocchini, Nicola Santoro & Giovanni Viglietta (2017): *Mediated Population Protocols: Leader Election and Applications*. In T V Gopal, Jäger Gerhard & Silvia Steila, editors: *Theory and Applications of Models of Computation*, Springer International Publishing, Cham, pp. 172–186, doi:10.1007/978-3-319-55911-7_13.

[8] Shantanu Das, Giuseppe Antonio Di Luna, Paola Flocchini, Nicola Santoro & Giovanni Viglietta (2017): *Mediated Population Protocols: Leader Election and Applications*. In: *Theory and Applications of Models of Computation*, Lecture Notes in Computer Science, Springer, Cham, pp. 172–186, doi:10.1007/978-3-319-55911-7_13.

[9] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui & Eric Ruppert (2006): *When Birds Die: Making Population Protocols Fault-tolerant*. In: *Proceedings of the Second IEEE International Conference on Distributed Computing in Sensor Systems*, DCOSS'06, Springer-Verlag, Berlin, Heidelberg, pp. 51–66, doi:10.1007/11776178_4.

[10] Javier Esparza, Pierre Ganty, Jérôme Leroux & Rupak Majumdar (2017): *Verification of population protocols*. *Acta Informatica* 54(2), pp. 191–215, doi:10.1007/s00236-016-0272-3.

[11] Daniele Gorla (2010): *Towards a unified approach to encodability and separation results for process calculi*. *Information and Computation* 208(9), pp. 1031 – 1053, doi:10.1016/j.ic.2010.05.002.

[12] Othon Michail, Ioannis Chatzigiannakis & Paul G. Spirakis (2011): *Mediated population protocols*. *Theoretical Computer Science* 412(22), pp. 2434–2450, doi:10.1016/j.tcs.2011.02.003.