EPTCS 222

Proceedings of the

Combined 23rd International Workshop on Expressiveness in Concurrency

and 13th Workshop on Structural Operational Semantics

Québec City, Canada, 22nd August 2016

Edited by: Daniel Gebler and Kirstin Peters

Published: 9th August 2016
DOI: 10.4204/EPTCS.222
ISSN: 2075-2180
Open Publishing Association

Table of Contents

Preface

This volume contains the proceedings of the Combined 23nd International Workshop on Expressiveness in Concurrency and the 13th Workshop on Structural Operational Semantics (EXPRESS/SOS 2016) which was held on 22 August 2016 in Québec, Canada, as an affiliated workshop of CONCUR 2016, the 27th International Conference on Concurrency Theory.

The EXPRESS workshops aim at bringing together researchers interested in the expressiveness of various formal systems and semantic notions, particularly in the field of concurrency. Their focus has traditionally been on the comparison between programming concepts (such as concurrent, functional, imperative, logic and object-oriented programming) and between mathematical models of computation (such as process algebras, Petri nets, event structures, modal logics, and rewrite systems) on the basis of their relative expressive power. The EXPRESS workshop series has run successfully since 1994 and over the years this focus has become broadly construed.

The SOS workshops aim at being a forum for researchers, students and practitioners interested in new developments, and directions for future investigation, in the field of structural operational semantics. One of the specific goals of the SOS workshop series is to establish synergies between the concurrency and programming language communities working on the theory and practice of SOS. Reports on applications of SOS to other fields are also most welcome, including: modelling and analysis of biological systems, security of computer systems programming, modelling and analysis of embedded systems, specification of middle-ware and coordination languages, programming language semantics and implementation, static analysis software and hardware verification, semantics for domain-specific languages and model-based engineering.

Since 2012, the EXPRESS and SOS communities have organized an annual combined EXPRESS/SOS workshop on the expressiveness of mathematical models of computation and the formal semantics of systems and programming concepts.

We received nine full paper submissions out of which the programme committee selected five for publication and presentation at the workshop. These proceedings contain these selected contributions. The workshop had an invited presentation:

What Do Probabilistic Programs Mean?,
by Joost-Pieter Katoen (RWTH Aachen University, Germany)

We would like to thank the authors of the submitted papers, the invited speaker, the members of the programme committee, and their subreviewers for their contribution to both the meeting and this volume. We also thank the CONCUR 2016 organizing committee for hosting EXPRESS/SOS 2016. Finally, we would like to thank our EPTCS editor Rob van Glabbeek for publishing these proceedings and his help during the preparation.

Daniel E. Gebler and Kirstin Peters,
August 2016

Program Committee

Additional Reviewers

A Note on the Expressiveness of BIP

Eduard Baranov and Simon Bliudze
École polytechnique fédérale de Lausanne, Station 14, 1015 Lausanne, Switzerland
firstname.lastname@epfl.ch

We extend our previous algebraic formalisation of the notion of component-based framework in order to formally define two forms—strong and weak—of the notion of full expressiveness. Our earlier result shows that the BIP (Behaviour-Interaction-Priority) framework does not possess the strong full expressiveness. In this paper, we show that BIP has the weak form of this notion and provide results detailing weak and strong full expressiveness for classical BIP and several modifications, obtained by relaxing the constraints imposed on priority models.

1 Introduction

In our previous work [1], we have formalised some of the properties that are desirable for component-based design frameworks, namely: incrementality, flattening, compositionality and modularity [17, 25]. We have also discussed the *full expressiveness* property, although without providing a formal definition for it. The formalisation is based on a very simple, abstract algebraic definition of the notion of component-based framework, which we extend below in order to also provide such formal definition of full expressiveness.

Intuitively, *flattening* requires that, for any component obtained by hierarchically applying two composition operators to a finite set of sub-components, there must exist an equivalent component obtained by applying a single composition operator to the same sub-components. *Full expressiveness w.r.t. a given set of operators*—e.g. those defined by a particular Structural Operational Semantics (SOS) rule format—requires that all operators in that set be expressible as composition operators in the component-based framework.

In [1], we have studied the satisfaction of the above properties by BIP (Behaviour-Interaction-Priority), which is a component-based framework for the design of correct-by-construction concurrent software and systems based on the separation of concerns between coordination and computation [2, 3].

BIP systems consist of components modelled as Labelled Transition Systems (LTS). Transitions are labelled by ports, which are used for synchronisation with other components. Composition operators defining such synchronisations are obtained by combining *interaction* and *priority models*. Operational semantics of the BIP composition operators is defined by SOS rules in a format, which is a restriction of GSOS [7]. Below we refer to this format as *BIP-like SOS*. We focus on the flattening and full expressiveness w.r.t. BIP-like SOS of BIP with the classical semantics defined in [4] and used in the language and code-generation tool-set developed by VERIMAG.[1]

In [1], we have provided a counter-example showing that the classical semantics of BIP does not possess flattening, which implies that it does not possess full expressiveness w.r.t. BIP-like SOS either. This shows that the often encountered informal statement: "BIP possesses the expressiveness of the universal glue" (or its equivalent in slightly different formulations) is based on an erroneous proposition in previous work [5, Proposition 4]. The fundamental reasons for this absence of full expressiveness lie in the definition of the priority models. A priority model is a strict partial order on the underlying

[1] http://www-verimag.imag.fr/New-BIP-tools.html

D. Gebler and K. Peters (Eds.): Combined Workshop on Expressiveness in Concurrency and Structural Operational Semantics (EXPRESS/SOS 2016).
EPTCS 222, 2016, pp. 1–14, doi:10.4204/EPTCS.222.1

interaction model (set of allowed interactions). This definition guarantees that applying a priority model does not introduce deadlocks in the otherwise deadlock-free system. However, such deadlocks can be introduced by certain operators respecting BIP-like SOS.

In [1], we have shown that relaxing the restrictions on priority models to allow arbitrary relations on interactions—rather than strict partial orders on interactions in the interaction model—provides full expressiveness w.r.t. the full class of BIP-like SOS operators.

In this paper, we refine this discussion as follows

- We formally define two notions—weak and strong—of full expressiveness. Weak full expressiveness means that any operator that can be defined by a set of BIP-like SOS rules can be expressed as a *hierarchical composition* of BIP composition operators, as opposed to only one composition operator for strong full expressiveness.
- We provide a syntactic characterisation of a subset of operators defined by BIP-like SOS rules.
- We show that BIP has weak full expressiveness with respect to this subset of operators.
- We show that relaxing the partial order restriction in the definition of priority models allows us to recover weak full expressiveness w.r.t. the full class of BIP-like SOS operators.

The rest of the paper is structured as follows: Section 2 presents the algebraic formalisation of the notion "component-based framework" and defines its basic properties, namely flattening, strong and weak full expressiveness. Section 3 introduces the BIP component-based framework and its formal semantics. Section 4 presents the main results of the paper as stated above. Section 5 briefly discusses some related work. Finally, Section 6 concludes the paper.

2 Algebraic formalisation of component-based frameworks

Each component-based design framework can be viewed as an algebra $\mathscr{A}$ of components equipped with a semantic mapping σ and an equivalence relation $\simeq$, satisfying a set of basic properties, which we list below. More precisely, $\mathscr{A}$ is an algebraic structure generated by a *behaviour type* $\mathscr{B}$ and a set $\mathscr{G}$ of *composition (glue) operators*:

$$\mathscr{A} ::= B \mid f(C_1,\ldots,C_n)\,,$$

with $B \in \mathscr{B}$, $C_1,\ldots,C_n \in \mathscr{A}$ and $f \in \mathscr{G}$. We call the elements of $\mathscr{A}$ *components* and the elements of $\mathscr{B}$ *behaviours*. The algebraic structure $\mathscr{A}$ represents the set of all systems constructible within the framework. Behaviour type $\mathscr{B}$ defines the *semantic nature* of the components manipulated by the framework.

The *semantic mapping* $\sigma : \mathscr{A} \to \mathscr{B}$ assigns to each component its meaning in terms of the behaviour type $\mathscr{B}$. The semantic mapping must be consistent: for all $B \in \mathscr{B}$, must hold the equality $\sigma(B) = B$. The semantic mapping is called *structural*, if it is defined by associating to each n-ary glue operator $f : \mathscr{A}^n \to \mathscr{A}$ a corresponding operator $\tilde{f} : \mathscr{B}^n \to \mathscr{B}$ and putting

$$\sigma\big(f(C_1,\ldots,C_n)\big) = \tilde{f}\big(\sigma(C_1),\ldots,\sigma(C_n)\big)\,, \qquad \text{for all } C_1,\ldots,C_n \in \mathscr{A} \text{ and } f \in \mathscr{G}\,.$$

Finally, the equivalence relation $\simeq \subseteq \mathscr{A} \times \mathscr{A}$—that allows comparing components in terms, for example, of their functionality, observable behaviour or capability of interaction with the environment—must respect the semantics: for all $C_1, C_2 \in \mathscr{A}$, must hold the implication $\sigma(C_1) = \sigma(C_2) \implies C_1 \simeq C_2$.

In this context, the flattening property mentioned in the introduction is formalised by requiring that

$$\forall i,j \in [1,n]\,(i \le j),\ \forall C_1,C_2,\ldots,C_n \in \mathscr{A},\ \forall f,g \in \mathscr{G},\ \exists h \in \mathscr{G} : \\ f\big(C_1,\ldots,C_{i-1},g(C_i,\ldots,C_j),C_{j+1},\ldots,C_n\big) \simeq h(C_1,\ldots,C_n)\,.$$

In other words, $\mathscr{G}$ must be closed under composition. Similarly, given a set of operators $\mathscr{O} \subseteq \bigcup_{n=0}^{\infty}(\mathscr{B}^n \to \mathscr{B})$, we say that the component-based framework $(\mathscr{A}, \sigma, \simeq)$ has *strong full expressiveness w.r.t.* $\mathscr{O}$ iff

$$\forall o \in \mathscr{O}^n,\ \exists \tilde{o} \in \mathscr{G} :\ \forall B_1, \ldots, B_n \in \mathscr{B},\ \sigma(\tilde{o}(B_1, \ldots, B_n)) = o(B_1, \ldots, B_n)\,,$$

where $\mathscr{O}^n = \mathscr{O} \cap (\mathscr{B}^n \to \mathscr{B})$. We say that $(\mathscr{A}, \sigma, \simeq)$ has *(weak) full expressiveness w.r.t.* $\mathscr{O}$ iff,

$$\forall o \in \mathscr{O}^n,\ \exists \tilde{o} \in \mathscr{G}[Z_1, \ldots, Z_n] :\ \forall B_1, \ldots, B_n \in \mathscr{B},\ \sigma(\tilde{o}[B_1/Z_1, \ldots, B_n/Z_n]) = o(B_1, \ldots, B_n)\,,$$

where $\mathscr{G}[Z_1, \ldots, Z_n]$ is the set of expressions on variables $Z_1, \ldots, Z_n$ obtained by hierarchically applying the glue operators from $\mathscr{G}$; whereas $\tilde{o}[B_1/Z_1, \ldots, B_n/Z_n] \in \mathscr{A}$ is the component obtained by substituting in $\tilde{o}$ the variables Z_i by components B_i, for all $i \in [1, n]$.

3 The BIP component-based framework

In this section, we briefly recall BIP and its classical operational semantics, as initially published in [4]. The behaviour type in BIP is the set of *Labelled Transition Systems* (LTS).

Definition 3.1. A *labelled transition system* (LTS) is a triple $(Q, P, \to)$, where Q is a set of *states*, P is a set of *ports*, and $\to \subseteq Q \times (2^P \setminus \{\emptyset\}) \times Q$ is a set of *transitions* labelled by *interactions*, i.e. non-empty sets of ports. For $q, q' \in Q$ and $a \in 2^P$, we write $q \xrightarrow{a} q'$ iff $(q, a, q') \in \to$. A label $a \in 2^P$ is *active* in a state $q \in Q$ (denoted $q \xrightarrow{a}$), iff there exists $q' \in Q$ such that $q \xrightarrow{a} q'$. We abbreviate $q \not\xrightarrow{a} \stackrel{def}{=} \neg(q \xrightarrow{a})$.

Note 3.2. In the rest of the paper, whenever we speak of a set of LTS $B_i = (Q_i, P_i, \to_i)$, for $i \in [1, n]$, we assume that all P_i are pairwise disjoint, i.e. $i \neq j$ implies $P_i \cap P_j = \emptyset$. We denote $P \stackrel{def}{=} \bigcup_{i=1}^{n} P_i$. When the indices are clear from the context, we drop them on transition relations and denote write $\to$.

Glue operators are separated in two categories: *interaction models* define the sets of allowed *interactions*, that is synchronisations between the transitions of their operand components; *priority models* define the scheduling—or conflict resolution—policies, reducing non-determinism when several synchronisations allowed by the interaction model are enabled simultaneously.

Interaction models An *interaction model* is a set of interactions $\gamma \subseteq 2^P \setminus \{\emptyset\}$. The semantics of the application of an interaction model γ is defined by putting $\sigma(\gamma(B_1, \ldots, B_n)) \stackrel{def}{=} (Q, P, \to_\gamma)$, with $Q = \prod_{i=1}^{n} Q_i$ and the minimal transition relation $\to_\gamma$ satisfying the rule

$$\frac{a \in \gamma \quad \left\{ q_i \xrightarrow{a \cap P_i} q_i' \,\middle|\, i \in I \right\} \quad \left\{ q_i = q_i' \,\middle|\, i \notin I \right\}}{q_1 \ldots q_n \xrightarrow{a}_\gamma q_1' \ldots q_n'}\,, \tag{1}$$

where $I = \{i \in [1, n] \,|\, a \cap P_i \neq \emptyset\}$. Intuitively, this means that an interaction a allowed by the interaction model γ can be fired when all the components involved in a are ready to fire the corresponding transitions. All the components that are not involved in a remain in their current state.

Priority models For a behaviour $B = (Q, P, \to)$, a *priority model* is a strict[2] partial order $\pi \subseteq 2^P \times 2^P$ (we write $a \prec b$ as a shorthand for $(a, b) \in \pi$). The semantics of the application of a priority model π is

[2] As opposed to a (non-strict) partial order, which is a reflexive, antisymmetric and transitive relation, a *strict* partial order is an irreflexive and transitive (hence also asymmetric) one.

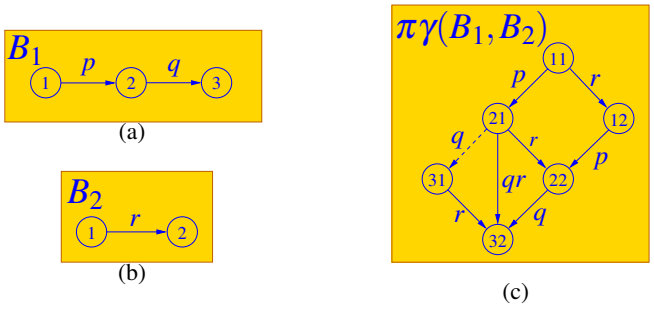


Figure 1: Component behaviours for Example 3.6

defined by putting $\sigma(\pi(B)) \stackrel{def}{=} (Q,P,\rightarrow_\pi)$, with the minimal transition relation $\rightarrow_\pi$ satisfying the rule

$$\frac{q \xrightarrow{a} q' \quad \left\{ q \not\xrightarrow{b} \,\middle|\, a \prec b \right\}}{q \xrightarrow{a}_\pi q'}. \tag{2}$$

Intuitively, this means that an interaction can be fired only if no higher-priority interaction is enabled.

Notice that the semantic mapping σ defined by (1) and (2) is structural, since it is defined by associating to both interaction and priority models operators on behaviours.

Note 3.3. The rules (1) and (2) defining the semantics of BIP operators require that a partition $\bigcup_{i=1}^n P_i = P$ be defined, but they do not depend on the specific behaviours $B_1,\ldots,B_n$.

Definition 3.4. An n-ary *BIP glue operator* is a triple $\big((P_i)_{i=1}^n,\gamma,\pi\big)$, where $(P_i)_{i=1}^n$ are disjoint sets of ports and, denoting $P \stackrel{def}{=} \bigcup_{i=1}^n P_i$, the remaining two elements $\gamma \subseteq 2^P$ and $\pi \subseteq \gamma \times \gamma$ are, respectively, interaction and priority models on P. (In the remainder of the paper, we omit the sets of ports $(P_i)_{i=1}^n$ when they are clear from the context.)

To simplify the notation, we denote the component obtained by applying the glue operator $\big((P_i)_{i=1}^n, \gamma,\pi\big)$ to sub-components $B_1,\ldots,B_n$, by $\pi\gamma(B_1,\ldots,B_n)$ instead of $\big((P_i)_{i=1}^n,\gamma,\pi\big)(B_1,\ldots,B_n)$. Furthermore, when $\pi = \emptyset$, we write directly $\gamma(B_1,\ldots,B_n)$, omitting π.

Definition 3.5. Two behaviours $B_i = (Q_i,P_i,\rightarrow)$, for $i = 1,2$ are *equivalent* if $P_1 = P_2$, and the two LTS are bisimilar, i.e. there exists a bisimulation [23] relation $R \subseteq Q_1 \times Q_2$ total on both Q_1 and Q_2.

Example 3.6. Consider the two components B_1 and B_2 shown in Figures 1(a) and 1(b), with $P_1 = \{p,q\}$ and $P_2 = \{r\}$, and put $\gamma = \{p,q,r,qr\}$ and $\pi = \{q \prec r\}$.[3] The glue operator defined by the combination of the interaction model γ and the priority model π is given by the following four rules, obtained by composing rules of forms (1) and (2) and removing premises, whereof satisfaction does not depend on the state of the operand components (e.g. the premise $a \in \gamma$ is satisfied in all states):

$$\frac{q_1 \xrightarrow{p} q_1'}{q_1q_2 \xrightarrow{p} q_1'q_2}, \qquad \frac{q_2 \xrightarrow{r} q_2'}{q_1q_2 \xrightarrow{r} q_1q_2'}, \qquad \frac{q_1 \xrightarrow{q} q_1' \quad q_2 \xrightarrow{r} q_2'}{q_1q_2 \xrightarrow{qr} q_1'q_2'}, \qquad \frac{q_1 \xrightarrow{q} q_1' \quad q_2 \not\xrightarrow{r}}{q_1q_2 \xrightarrow{q} q_1'q_2}. \tag{3}$$

The composed component $\pi\gamma(B_1,B_2)$ is shown in Figure 1(c). The dashed arrow $21 \xrightarrow{q} 31$ shows the transition present only in $\gamma(B_1,B_2)$, but not in $\pi\gamma(B_1,B_2)$. Solid arrows show the transitions of $\pi\gamma(B_1,B_2)$.

[3] To simplify the notation we use the juxtaposition $\gamma = \{p,q,r,qr\}$ instead of the set notation $\gamma = \big\{\{p\},\{q\},\{r\},\{q,r\}\big\}$ for interactions. Similarly, we directly write $\pi = \{q \prec r\}$ instead of $\pi = \{(q,r)\}$

Among the transitions labelled by q, only the transition $22 \xrightarrow{q} 32$ is enabled and not $21 \xrightarrow{q} 31$ (Figure 1(c)). Indeed, the negative premise in the fourth rule of (3), generated by the priority $q \prec r$, suppresses the interaction q when a transition labelled r is possible in the second component. □

After merging rules of forms (1) and (2) and the simplification by removing the constant premises, all rules used to define the semantics of BIP glue operators follow the format

$$\frac{\left\{q_i \xrightarrow{a\cap P_i} q_i' \,\middle|\, i \in I\right\} \quad \left\{q_i = q_i' \,\middle|\, i \notin I\right\} \quad \left\{q_j \not\xrightarrow{b_j^k} \,\middle|\, j \in J, k \in K_j\right\}}{q_1 \dots q_n \xrightarrow{a} q_1' \dots q_n'}, \tag{4}$$

where $I = \{i \in [1,n] \,|\, a \cap P_i \neq \emptyset\}$, whereas $J, K_j \subseteq [1,n]$ and, for each $j \in J$ and $k \in K_j$, holds $b_j^k \in 2^{P_j}$.

Let us now recall an important property of the BIP glue operators with the above semantics, which was originally shown in [16]: application of a priority model does not introduce deadlocks.

Definition 3.7. Let $B = (Q, P, \to)$ be a behaviour. A state $q \in Q$ is a *deadlock* iff holds $\forall a \subseteq P$, $q \not\xrightarrow{a}$.

Lemma 3.8 ([16])**.** *Let $B_i = (Q_i, P_i, \to)$, for $i \in [1,n]$, be a set of behaviours, γ and π be respectively interaction and priority models on $P = \bigcup_{i=1}^n P_i$. A state $q \in \prod_{i=1}^n Q_i$ is a deadlock in $\pi\gamma(B_1, \dots, B_n)$ if and only if it is a deadlock in $\gamma(B_1, \dots, B_n)$.*

Proof. The "if" implication is trivial. To prove the "only if" implication, assume that, for some $a \in \gamma$, we have $q \xrightarrow{a}_\gamma$. Let $b \subseteq P$ be an interaction, maximal w.r.t. π, such that $b \in \gamma$, $a \prec b$ and $q \xrightarrow{b}_\gamma$. If such b exists, holds $q \xrightarrow{b}_\pi$. Otherwise holds $q \xrightarrow{a}_\pi$. In both cases, q is not a deadlock in $\pi\gamma(B_1, \dots, B_n)$. □

Notice that this proof does not rely on π being a strict partial order. The lemma can be generalised to any *acyclic* relation $\pi \subseteq \gamma \times \gamma$.

4 Expressiveness

We now consider full expressiveness of BIP w.r.t. the set $\mathscr{O}$ of operators defined as pairs $\left((P_i)_{i=1}^n, \mathscr{R}\right)$, where n is the arity of the operator, $(P_i)_{i=1}^n$ are pair-wise disjoint sets of ports and $\mathscr{R}$ is a set of SOS rules in the format (4). In [1], we have shown that the operator defined by the following four rules, which respect the format (4), cannot be expressed as a BIP glue operator in the classical semantics:

$$\frac{q_1 \xrightarrow{p} q_1' \quad q_2 \not\xrightarrow{r}}{q_1q_2q_3 \xrightarrow{p} q_1'q_2q_3}, \quad \frac{q_1 \xrightarrow{q} q_1'}{q_1q_2q_3 \xrightarrow{q} q_1'q_2q_3}, \quad \frac{q_2 \xrightarrow{s} q_2'}{q_1q_2q_3 \xrightarrow{s} q_1q_2'q_3}, \quad \frac{q_2 \xrightarrow{r} q_2' \quad q_3 \xrightarrow{t} q_3'}{q_1q_2q_3 \xrightarrow{rt} q_1q_2'q_3'}. \tag{5}$$

We conclude that the classical semantics of BIP does not have neither flattening, nor strong full expressiveness w.r.t. $\mathscr{O}$.[4]

Furthermore, the example below shows that the classical semantics of BIP does not have even weak full expressiveness.

Example 4.1. Consider a composition operator defined by the following two rules:

$$\frac{q_1 \xrightarrow{p} q_1' \quad q_1 \not\xrightarrow{r}}{q_1 \xrightarrow{p} q_1'}, \qquad \frac{q_1 \xrightarrow{r} q_1' \quad q_1 \not\xrightarrow{p}}{q_1 \xrightarrow{r} q_1'}, \tag{6}$$

applied to the component in Figure 2. Assume that there exists a hierarchy of BIP glues, such that applying them to the component in Figure 2 results in an equivalent composed component. States 1

[4] For the details of this example and the associated discussion, we refer the reader to [1].

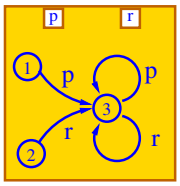


Figure 2: Component behaviour for Example 4.1

and 2 of the composed component have outgoing transitions p and r, respectively, thus all interaction models in the glues have to contain both interactions p and q. State 3 of the composed component is a deadlock. Interaction models do not forbid any transition from this state and priority models cannot introduce deadlock by Lemma 3.8. This contradicts the assumption and, consequently, the set of rules (6) is not expressible in BIP.

The two fundamental reasons for the lack of expressiveness are related to the definition of the priority model:

- the information used by the priority model refers only to interactions authorised by the underlying interaction model—all the information about transitions enabled in sub-components is lost [1];
- the priority model π must be a strict partial order.

As we explain below, among these two reasons, the first one is easily addressed to achieve weak, rather than strong, full expressiveness, whereas the second one presents the main difficulty.

What can be done without changing the BIP glue? Consider an n-ary operator $o : LTS^n \to LTS$ defined by $(P_i)_{i=1}^n$ and the set of rules

$$\frac{\left\{q_i \xrightarrow{a^l \cap P_i} q_i' \,\middle|\, i \in I^l\right\} \quad \left\{q_i = q_i' \,\middle|\, i \notin I^l\right\} \quad \left\{q_j \not\xrightarrow{b^l_{j,k}} \,\middle|\, j \in J^l, k \in K^l_j\right\}}{q_1 \dots q_n \xrightarrow{a^l} q_1' \dots q_n'}, \qquad \text{for } l \in [1,m], \tag{7}$$

where, as above, $I^l = \left\{i \in [1,n] \,\middle|\, a^l \cap P_i \neq \emptyset\right\}$. For an interaction $a \in \{a^l \,|\, l \in [1,m]\}$, denote $R_a \stackrel{def}{=} \{l \in [1,m] \,|\, a = a^l\}$ the set of rules with the conclusion labelled by a. Clearly, for the interaction a to be inhibited by the negative premises, one such premise must be involved for each rule in R_a. We denote by $j : R_a \rightsquigarrow J$ the *choice mappings* $j : R_a \to \bigcup_{l=1}^m J^l$, such that $j(l) \in J^l$, for all $l \in R_a$.[5]

We define the *inhibiting relation* $\pi \subseteq 2^P \times 2^P$ (where $P = \bigcup_{i=1}^n P_i$) by putting

$$\pi \;=\; \bigcup_{l=1}^{m} \left\{(a^l, b) \,\middle|\, b = \bigcup_{s \in R_{a^l}} b^s_{j(s),k(s)}, \text{ for some } j : R_{a^l} \rightsquigarrow J,\ k(s) \in K^s_{j(s)}\right\}. \tag{8}$$

Proposition 4.2. *If π has cycles, then the operator o cannot be realised by any hierarchical composition of BIP glue operators.*

Proof. Consider a cycle in the inhibiting relation $\pi : a_1 \prec a_2 \prec \dots \prec a_l \prec a_1$.

Let $P = \bigcup_{j=1}^n P_j$, where $P_j = \{p_1^j, \dots, p_m^j\}$. Let $c_i^j = a_i \cap P_j$ for $i \in [1,l]$, $j \in [1,n]$ and $C_j = \left\{c_i^j \,\middle|\, c_i^j \neq \emptyset\right\}$. For each j consider a behaviour as shown in Figure 3. There are no transitions from state 0; from each

[5] The notion of choice mappings could also be defined as a co-product of mappings $\{l\} \to J^l$ from singleton subsets $\{l\} \subseteq R_a$.

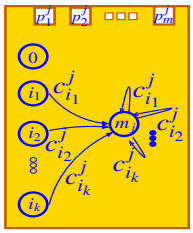


Figure 3: Component behaviour for Proposition 4.2

state i such that $c_i^j \neq \emptyset$, there is a single transition to state m_j with labels $c_i^j \in C_j$, respectively, and loop transitions in state m_j with labels $c_i^j \in C_j$.

The composition of such behaviours with the operator o allows a single transition a_i from the state $q_1 \dots q_n$, where $q_j = i$ if $c_i^j \neq \emptyset$ or $q_j = 0$ otherwise. In order to allow these transitions, an interaction model of a BIP glue must contain all a_i. In the state $q_1 \dots q_n$, with $q_j = m_j$, all interactions $a_1, \dots, a_l$ are available. The operator o forbids all of them from this state. Interaction models of BIP glues allow all these interactions and priority models cannot introduce deadlock in this state Lemma 3.8. Thus, this system is not expressible in BIP. □

Proposition 4.3. *If π is acyclic, then the operator o can be realised by a hierarchical composition of BIP glue operators.*

Proof. Since π is acyclic, we can associate a depth $d(a)$ to each interaction a involved in π as the length of the longest path leading to a in the directed acyclic graph defined by π. Denote $d \stackrel{def}{=} \max_a d(a)$. Furthermore, for $i \in [1,d]$, denote $\pi_i \stackrel{def}{=} \{(a,b) \in \pi \,|\, d(a) = i-1\}$.

Clearly all π_i are strict partial orders. Furthermore $\pi_i \subseteq \pi \subseteq \gamma_1 \times \gamma_1$, for all $i \in [1,d]$ and

$$\gamma_1 = \gamma_2 \cup \bigcup_{l=1}^{m} \left\{ \bigcup_{s \in R_{a^l}} b^s_{j(s),k(s)} \,\middle|\, j : R_{a^l} \rightsquigarrow J,\ k(s) \in K^s_{j(s)} \right\},$$
$$\gamma_2 = \left\{ a^l \,\middle|\, l \in [1,m] \right\}.$$

Hence, for all $i \in [1,d]$, (γ_1, π_i) is a BIP glue operator.

The operator o is equivalent to the composition $(\gamma_2, \emptyset) \circ (\gamma_1, \pi_d) \circ \dots \circ (\gamma_1, \pi_1)$. We show that for any set of behaviours $B_i = (Q_i, P_i, \rightarrow)$, with $i \in [1,n]$, holds $\sigma\big(\gamma_2\big(\pi_d \gamma_1 \big(\dots \pi_1 \gamma_1 (B_1, \dots, B_n)\big) \dots\big)\big) = o(B_1, \dots, B_n)$. We denote

$$B_o = o(B_1, \dots, B_n), \qquad B_{\pi\gamma} = \sigma\big(\gamma_2\big(\pi_d \gamma_1 \big(\dots \pi_1 \gamma_1 (B_1, \dots, B_n)\big) \dots\big)\big).$$

The sets of states and ports of these behaviours are the same, thus we only need to check that their transitions coincide.

Let $q_1 \dots q_n \xrightarrow{a} q'_1 \dots q'_n$ in B_o. This means that, among the rules defining o, i.e. for some $l \in [1,m]$, there is a rule

$$\frac{\left\{ q_i \xrightarrow{a \cap P_i} q'_i \,\middle|\, i \in I^l \right\} \quad \left\{ q_i = q'_i \,\middle|\, i \notin I^l \right\} \quad \left\{ q_j \xrightarrow{b^l_{j,k}} \,\middle|\, j \in J^l, k \in K^l_j \right\}}{q_1 \dots q_n \xrightarrow{a} q'_1 \dots q'_n}, \tag{9}$$

such that $q_i \xrightarrow{a\cap P_i}$, for all $i \in I$, and $q_j \overset{b^l_{j,k}}{\not\longrightarrow}$ for all $j \in J^l, k \in K^l_j$. By construction both γ_1 and γ_2 contain a. Hence, a is enabled in the state $q_1 \dots q_n$ of $\gamma_1(B_1,\dots,B_n)$ and in the same state of $B_{\pi\gamma}$, provided that it is not disabled by any of priorities $\pi_1,\dots,\pi_d$. Thus, we have to show that no interaction available from this state has higher priority. By construction, priority rules that contain a in the left-hand side can appear only in $\pi_{d(a)-1}$, thus other priority models cannot block a. Priority rules of the form $a \prec b$ have $b = \bigcup_{s\in R_a} b^s_{j(s),k(s)}$, for some $j : R_a \rightsquigarrow J$ and $k(s) \in K^s_{j(s)}$. Since all the premises of (9) are satisfied in $q_1 \dots q_n$, interaction $b^l_{j(l),k(l)}$ is disabled. Hence, b is also disabled. Thus $q_1 \dots q_n \xrightarrow{a} q'_1 \dots q'_n$ in $B_{\pi\gamma}$.

Let $q_1 \dots q_n \xrightarrow{a} q'_1 \dots q'_n$ in $B_{\pi\gamma}$. This means that both γ_1 and γ_2 contain the interaction a. Therefore, by the construction of γ_2, there is at least one rule

$$\frac{\left\{q_i \xrightarrow{a\cap P_i} q'_i \,\middle|\, i \in I\right\} \quad \left\{q_i = q'_i \,\middle|\, i \notin I\right\} \quad \left\{q_j \overset{b_{j,k}}{\not\longrightarrow} \,\middle|\, j \in J, k \in K_j\right\}}{q_1 \dots q_n \xrightarrow{a} q'_1 \dots q'_n}, \tag{10}$$

among the rules defining o. Furthermore, the priority model $\pi_{d(a)-1}$ contains priorities of the form $a \prec b$, with $b = \bigcup_{s\in R_a} b^s_{j(s),k(s)}$, for all $j : R_a \rightsquigarrow J$ and $k(s) \in K^s_{j(s)}$. Notice that a priority rule $b \prec c$ such that $a \prec b$ cannot appear in priorities $\pi_1,\dots,\pi_{d(a)-1}$ since $d(b) \geq d(a)+1$. Assume that none of rules defining o, with the conclusion labelled by a, applies in $q_1 \dots q_n \xrightarrow{a} q'_1 \dots q'_n$. This necessarily means that each of these rules has a negative premise that is not satisfied. Let $b = \bigcup_{s\in R_a} b^s_{j(s),k(s)}$ with $b^s_{j(s),k(s)}$, for all $s \in R_a$, being the labels of dissatisfied premises. Then b is an enabled interaction in $\gamma_1(B_1,\dots,B_n)$ such that $a \prec b$ and b cannot be blocked by priorities $\pi_1,\dots,\pi_{d(a)-1}$. Consequently, b is enabled in $\pi_{d(a)-1}\gamma_1\big(\dots\pi_1\gamma_1(B_1,\dots,B_n)\dots\big)$ and blocks a, which contradicts the assumption $q_1 \dots q_n \xrightarrow{a} q'_1 \dots q'_n$ in $B_{\pi\gamma}$. Hence, there is at least one rule of the form (10) in the definition of o with all premises satisfied in $q_1 \dots q_n$ and, therefore, $q_1 \dots q_n \xrightarrow{a} q'_1 \dots q'_n$ in B_o. □

Thus, we conclude that BIP has weak full expressiveness w.r.t. the class of BIP-like SOS operators with acyclic inhibiting relations.

What can be done in the general case? In [1], we have proposed the following notion of relaxed priority model.

Definition 4.4. Let P be a set of ports. A *relaxed priority model* on P is a relation $\pi \subseteq 2^P \times (2^P \setminus \{\emptyset\})$. A *relaxed BIP operator* is a triple $\big((P_i)^n_{i=1}, \gamma, \pi\big)$, with $P = \bigcup^n_{i=1} P_i$, such that $\gamma \subseteq 2^P \setminus \{\emptyset\}$ is an interaction model and $\pi \subseteq \gamma \times \gamma$ is a relaxed priority model.

The semantics of relaxed priority models is defined exactly as that of classical priority models, by (2). Notice that we do not require the relation π to be acyclic. If all interactions involved in a cyclic dependency in π are enabled simultaneously, they block each other, potentially introducing a deadlock.

Given a BIP-like SOS operator o, we consider its inhibiting relation π (see (8)) and the interaction models γ_1, γ_2 as in the proof of Proposition 4.3. Since $\pi \subseteq \gamma_1 \times \gamma_1$, the operator (γ_1, π) is a relaxed BIP operator. The operator o is then equivalent to the composition $(\gamma_2, \emptyset) \circ (\gamma_1, \pi)$, where π is considered as a relaxed priority model.

Proposition 4.5. *For any set of behaviours $B_i = (Q_i, P_i, \to)$, with $i \in [1,n]$, holds*

$$\sigma\big(\gamma_2\big(\pi\gamma_1(B_1,\dots,B_n)\big)\big) = o(B_1,\dots,B_n)\,.$$

Proof. For a set of behaviours $B_i = (Q_i, P_i, \to)$, with $i \in [1,n]$, denote

$$B_o = o(B_1, \ldots, B_n), \qquad B_{\pi\gamma} = \sigma\big(\gamma_2\big(\pi\gamma_1(B_1, \ldots, B_n)\big)\big).$$

The sets of states and ports of these behaviours are the same, thus we only need to check that their transitions coincide.

Let $q_1 \ldots q_n \xrightarrow{a} q'_1 \ldots q'_n$ in B_o. This means that, among the rules defining o, i.e. for some $l \in [1,m]$, there is a rule

$$\frac{\left\{q_i \xrightarrow{a\cap P_i} q'_i \,\middle|\, i \in I^l\right\} \quad \left\{q_i = q'_i \,\middle|\, i \notin I^l\right\} \quad \left\{q_j \not\xrightarrow{b^l_{j,k}} \,\middle|\, j \in J^l, k \in K^l_j\right\}}{q_1 \ldots q_n \xrightarrow{a} q'_1 \ldots q'_n}, \tag{11}$$

such that $q_i \xrightarrow{a\cap P_i}$, for all $i \in I$, and $q_j \not\xrightarrow{b^l_{j,k}}$ for all $j \in J^l, k \in K^l_j$. By construction both γ_1 and γ_2 contain a. Hence, a is enabled in the state $q_1 \ldots q_n$ of $\gamma_1(B_1, \ldots, B_n)$ and in the same state of $\gamma_2\big(\pi\gamma_1(B_1, \ldots, B_n)\big)$, provided that it is not disabled by the priority π. Thus, we have to show that no interaction available from this state has higher priority. Priority rules in π that contain a are of the form $a \prec b$, with $b = \bigcup_{s\in R_a} b^s_{j(s),k(s)}$, for some $j : R_a \rightsquigarrow J$ and $k(s) \in K^s_{j(s)}$. Since all the premises of (11) are satisfied in $q_1 \ldots q_n$, interaction $b^l_{j(l),k(l)}$ is disabled. Hence, b is also disabled. Thus $q_1 \ldots q_n \xrightarrow{a} q'_1 \ldots q'_n$ in $B_{\pi\gamma}$.

Let $q_1 \ldots q_n \xrightarrow{a} q'_1 \ldots q'_n$ in $B_{\pi\gamma}$. This means that bot γ_1 and γ_2 contain the interaction a. Therefore, by the construction of γ_2, there is at least one rule

$$\frac{\left\{q_i \xrightarrow{a\cap P_i} q'_i \,\middle|\, i \in I\right\} \quad \left\{q_i = q'_i \,\middle|\, i \notin I\right\} \quad \left\{q_j \not\xrightarrow{b_{j,k}} \,\middle|\, j \in J, k \in K_j\right\}}{q_1 \ldots q_n \xrightarrow{a} q'_1 \ldots q'_n}, \tag{12}$$

among the rules defining o. Furthermore, the priority model π has to contain priorities of the form $a \prec b$, with $b = \bigcup_{s\in R_a} b^s_{j(s),k(s)}$, for all $j : R_a \rightsquigarrow J$ and $k(s) \in K^s_{j(s)}$. Assuming now that none of rules defining o, with the conclusion labelled by a, applies in $q_1 \ldots q_n \xrightarrow{a} q'_1 \ldots q'_n$. Since $q_1 \ldots q_n \xrightarrow{a} q'_1 \ldots q'_n$ in $B_{\pi\gamma}$, this necessarily means that each of these rules has a negative premise that is not satisfied. Let $b = \bigcup_{s\in R_a} b^s_{j(s),k(s)}$ with $b^s_{j(s),k(s)}$, for all $s \in R_a$, being the labels of dissatisfied premises. Then b is an enabled interaction such that $a \prec b$, which contradicts the assumption $q_1 \ldots q_n \xrightarrow{a} q'_1 \ldots q'_n$ in $B_{\pi\gamma}$. Hence, there is at least one rule of the form (12) in the definition of o with all premises satisfied in $q_1 \ldots q_n$ and, therefore, $q_1 \ldots q_n \xrightarrow{a} q'_1 \ldots q'_n$ in B_o. □

Thus, we conclude that BIP with relaxed priority models has weak full expressiveness w.r.t. the set of all BIP-like SOS operators.

Notice that the relaxed priority model does not allow recovering strong full expressiveness. For instance, consider the operator defined by the single rule

$$\frac{q_1 \xrightarrow{p} q'_1 \quad q_1 \not\xrightarrow{r}}{q_1 \xrightarrow{p} q'_1}, \tag{13}$$

applied to the behaviour in Figure 2. The composed component has a single transition $1 \xrightarrow{p} 3$. The interaction model of BIP cannot contain r, as it is not possible to exclude transition $2 \xrightarrow{r} 3$ with a priority model. The transition $3 \xrightarrow{p} 3$ has to be excluded by the priority model, however it cannot use r in the priority relation.

Further relaxation of the definition of the BIP operator by removing the restriction $\pi \subseteq \gamma \times \gamma$ requires a slight modification of the semantics. Clearly, the behaviour $\gamma(B_1,\ldots,B_n)$ does not have transitions that are not in γ and priority rules that can be applied to this behaviour are in $\gamma \times \gamma$. Thus, we need to apply interaction and priority models simultaneously. The semantics of the simultaneous application of an interaction model γ and a priority model π is defined by putting $\sigma(\pi\gamma(B_1,\ldots,B_n)) \stackrel{def}{=} (Q,P,\rightarrow_{\pi\gamma})$, with $Q = \prod_{i=1}^{n} Q_i$ and the minimal transition relation $\rightarrow_{\pi\gamma}$ inductively defined by the set of rules

$$\left\{ \frac{\left\{q_i \xrightarrow{a\cap P_i} q_i' \,\middle|\, i \in I\right\} \quad \left\{q_i = q_i' \,\middle|\, i \notin I\right\} \quad \left\{q_j \not\xrightarrow{b\cap P_j} \,\middle|\, b \in K_a\right\}}{q_1 \ldots q_n \xrightarrow{a}_{\pi\gamma} q_1' \ldots q_n'} \;\middle|\; a \in \gamma, j : K_a \rightsquigarrow [1,n] \right\}, \qquad (14)$$

where $I = \{i \in [1,n] \,|\, a \cap P_i \neq \emptyset\}$, $K_a = \{b | a \prec b\}$ and $j : K_a \rightsquigarrow [1,n]$ is a choice mapping $j : K_a \rightarrow [1,n]$, such that, for all $b \in K_a$, holds $b \cap P_{j(b)} \neq \emptyset$.

With this relaxation we obtain strong full expressiveness, since the operator o is then clearly equivalent to (γ_2, π).

What cannot be achieved? Consider another relaxation of the definition of BIP glue operators, by considering operators $\left((P_i)_{i=1}^n, \gamma, \pi\right)$, with $P = \bigcup_{i=1}^n P_i$, such that the priority model $\pi \subseteq 2^P \times (2^P \setminus \{\emptyset\})$ is a strict partial order, without requiring that it refer only to interactions (i.e. we do not impose $\pi \subseteq \gamma \times \gamma$). This relaxation does not recover even weak full expressiveness w.r.t. BIP-like SOS operators. Indeed, Example 4.1 is still no expressible.

5 Related Work

The results in this paper build mainly on our previous work. However, the following related work should also be mentioned.

Usually, comparison between formalisms and models is by flattening structure and reduction to a behaviorally equivalent model, e.g. automata and Turing machine. In this manner, all finite state formalisms turn out to be expressively equivalent independently of the features used for the composition of behaviors. Many models and languages are Turing-expressive, while their coordination capabilities are tremendously different. [5]

A first framework formally capturing meanings of expressiveness for sequential programming languages and taking into account not only the semantics but also the primitives of languages was provided in [13]. It allows formal reasoning about and distinguishing between *core elements* of a language and *syntactic sugar*. Although a number of studies have taken a similar approach in the context of concurrency, we will only point to [15] and the references therein. The key difference of our approach lies in the strong separation between the computation and coordination aspects of the behaviour of concurrent systems. Indeed, we consider that all sequential computation resides within the components of the system that are not subject to any kind of modification. Thus, we focus on the following question: *what system behaviour can be obtained by coordination of a given set of concurrent components?* In particular, this precludes the expression of parallel composition by choice operators, as in the expansion law [20].

An extensive overview of SOS formats is provided in [22], including some results comparing their expressiveness. More results comparing different formats of SOS can be found in [21]. The expressiveness property is closely related to the translation between languages. One of the definitions of encoding compared with other approaches can be found in [14]. It should be noted, however, that the above mentioned separation of concerns principle also leads to a very simple rule format. Indeed, the format that we

consider is a small subset of GSOS. Our focus in this paper, is more on the expressiveness of coordination mechanism provided by BIP than on that of the various SOS rule features.

There exist several works comparing BIP with various connector frameworks. A comparative study of three connector frameworks—tile model [8], wire calculus [26] and BIP [3]—was presented in [9]. Recently an attempt to relate BIP and Reo has been done [11]. From the operational semantics perspective, these comparisons only take in account operators with positive premises. In particular, priority in BIP is not considered.

Finally, in our formalisation of component-based frameworks, we rely on the notion of "behaviour type". This can cover a very large spectrum, ranging from programs and labelled transition systems, through OSGi bundles and browser plug-ins, to systems of differential equations etc. Behaviour types can be organised in type systems and studied separately, as, for example, in the co-algebra theory [24]. However, this notion should be distinguished, for instance, from classes in object-oriented programming or session [10, 18] and behavioural [19] types for communication protocols. For instance, the notion of a class could be compared to that of a behaviour type in our sense as follows: a program would typically comprise a multitude of classes, whereas a component framework has only one underlying behaviour type. Although, in principle, component-based frameworks can be heterogeneous, e.g. Ptolemy II [12], that is rely on several distinct behaviour types for the design process, those aimed at the design of executable systems must have an underlying unifying behaviour type allowing the study and manipulation of a system as a whole.

6 Conclusion

Our previous investigations [1] of several properties that we consider fundamental for component-based design frameworks have revealed that the often encountered informal statement: "BIP possesses the expressiveness of the universal glue" (or its equivalent in slightly different formulations) is based on an erroneous proposition in previous work [5, Proposition 4]. We have, therefore, undertook an additional study of BIP expressiveness, whereof the results have been presented in this paper.

To achieve this goal, we rely on the algebraic formalisation of the notion of component-based design framework introduced in [1]. We have defined two new properties, *weak* and *strong full expressiveness* w.r.t. a given set of composition operators, which characterise whether these can be expressed by using the composition operators of the component-based framework under consideration. These two properties are very close to the *weakly more expressive* and *strongly more expressive* preorders introduced in [5]. In particular, for a component-based framework $(\mathscr{A}, \sigma, \simeq)$, with the underlying structure $\mathscr{A}$ being generated by a set of glue operators $\mathscr{G}$, the strong full expressiveness property w.r.t. a set of operators $\mathscr{O}$ coincides with the statement that $\mathscr{G}$ is strongly more expressive than $\mathscr{O}$ in terms of [5]. However, the formal definition that we have provided in Section 2 is novel and has the advantage of fitting elegantly with that of the component-based design frameworks in [1]. Furthermore, the notion of weak full expressiveness is different from the *weakly more expressive* preorder: the former relaxes the strong form of the property by allowing hierarchical composition of glue operators, whereas the latter considers only flat operators, but allows a limited use of additional coordinating behaviour. Studying the combination of the two relaxations could be an interesting direction for future work.

We have studied the weak and full expressiveness of BIP w.r.t. operators defined by SOS rules in a particular format, which we call *BIP-like SOS*. The set of all the operators that can be defined in this format is the "universal glue", w.r.t. which full expressiveness has been erroneously claimed in [5].

We observe that there are two obstacles to achieving strong full expressiveness: 1) a priority model is required to be a *strict partial order* on interactions and 2) by the definition of the BIP operational seman-

tics, priorities can only be applied to interactions that appear in the interaction model. The combination of these two requirements ensures that priorities cannot introduce new deadlocks. However, negative premises in BIP-like SOS rules—which correspond to priorities in BIP glue operators—can introduce deadlocks. To characterise this situation, we consider, for a set of BIP-like SOS rules, a corresponding *inhibiting relation*. In order to introduce deadlocks, this relation must have cycles. We show that BIP glue operators have weak full expressiveness w.r.t. BIP-like SOS operators that have acyclic inhibiting relations, with at most $d+1$ layers of glue necessary to encode a BIP-like SOS operator, whereof the depth of the inhibiting relation is d.

A relaxation of both of the above requirements together recovers *strong* full expressiveness w.r.t. all BIP-like SOS. However, it calls for a definition of the operational semantics of BIP glue operators, which combines the interaction and the priority models, as opposed to the classical definition, where the interaction model is applied first, then the priority model is applied to the resulting component.

A relaxation of only the first requirement, which does not require any other modifications of the BIP semantics, leads to *weak* full expressiveness w.r.t. the set of all BIP-like SOS operators. Moreover, we have shown that at most two layers of glue are necessary to encode any operator.

As mentioned above, studying the combination of the two weak forms of full expressiveness—allowing both hierarchical glue and limited use of additional coordinating behaviour—could be an interesting direction for future work. Another direction for future work would consist in exploring the expressiveness of the full BIP framework, including the data manipulation and transfer, which has been recently formalised in [6]. Finally, a third extension could consist in studying larger SOS formats, including, for instance, *witness premises*, i.e. positive premises that allow testing the possibility of an action that does not, however, contribute to the conclusion of the rule.

References

[1] Eduard Baranov & Simon Bliudze (2015): *Offer semantics: Achieving compositionality, flattening and full expressiveness for the glue operators in BIP*. *Science of Computer Programming* 109(0), pp. 2–35, doi:10.1016/j.scico.2015.05.011. Selected Papers of the 6th Interaction and Concurrency Experience (ICE 2013).

[2] Ananda Basu, Saddek Bensalem, Marius Bozga, Jacques Combaz, Mohamad Jaber, Thanh-Hung Nguyen, Joseph Sifakis et al. (2011): *Rigorous component-based system design using the BIP framework*. *IEEE Software* 28(3), pp. 41–48, doi:10.1109/MS.2011.27.

[3] Ananda Basu, Marius Bozga & Joseph Sifakis (2006): *Modeling Heterogeneous Real-time Components in BIP*. In: *4th IEEE Int. Conf. on Software Engineering and Formal Methods (SEFM06)*, pp. 3–12, doi:10.1109/SEFM.2006.27. Invited talk.

[4] Simon Bliudze & Joseph Sifakis (2007): *The Algebra of Connectors — Structuring Interaction in BIP*. In: *Proc. of the EMSOFT'07*, ACM SigBED, pp. 11–20, doi:10.1145/1289927.1289935.

[5] Simon Bliudze & Joseph Sifakis (2008): *A Notion of Glue Expressiveness for Component-Based Systems*. In Franck van Breugel & Marsha Chechik, editors: *CONCUR 2008*, *LNCS* 5201, Springer, pp. 508–522, doi:10.1007/978-3-540-85361-9_39.

[6] Simon Bliudze, Joseph Sifakis, Marius Dorel Bozga & Mohamad Jaber (2014): *Architecture Internalisation in BIP*. In: *Proceedings of the 17th International ACM Sigsoft Symposium on Component-based Software Engineering*, CBSE '14, ACM, New York, NY, USA, pp. 169–178, doi:10.1145/2602458.2602477.

[7] Bard Bloom (1989): *Ready Simulation, Bisimulation, and the Semantics of CCS-Like Languages*. Ph.D. thesis, Massachusetts Institute of Technology.

[8] Roberto Bruni, Ivan Lanese & Ugo Montanari (2006): *A basic algebra of stateless connectors*. *Theor. Comput. Sci.* 366(1), pp. 98–120, doi:10.1016/j.tcs.2006.07.005.

[9] Roberto Bruni, Hernn Melgratti & Ugo Montanari (2012): *Connector Algebras, Petri Nets, and BIP*. In Edmund Clarke, Irina Virbitskaite & Andrei Voronkov, editors: *Perspectives of Systems Informatics*, *Lecture Notes in Computer Science* 7162, Springer Berlin Heidelberg, pp. 19–38, doi:10.1007/978-3-642-29709-0_2.

[10] Mariangiola Dezani-Ciancaglini & Ugo de'Liguoro (2010): *Sessions and Session Types: An Overview*. In Cosimo Laneve & Jianwen Su, editors: *Web Services and Formal Methods: 6th International Workshop, WS-FM 2009*, *Lecture Notes in Computer Science* 6194, Springer Berlin Heidelberg, pp. 1–28, doi:10.1007/978-3-642-14458-5_1.

[11] Kasper Dokter, Sung-Shik T. Q. Jongmans, Farhad Arbab & Simon Bliudze (2015): *Relating BIP and Reo*. In Sophia Knight, Ivan Lanese, Alberto Lluch-Lafuente & Hugo Torres Vieira, editors: *Proceedings 8th Interaction and Concurrency Experience, ICE 2015, Grenoble, France, 4-5th June 2015.*, *EPTCS* 189, pp. 3–20, doi:10.4204/EPTCS.189.3.

[12] J. Eker, J.W. Janneck, E.A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs & Y. Xiong (2003): *Taming Heterogeneity: The Ptolemy Approach*. *Proceedings of the IEEE* 91(1), pp. 127–144, doi:10.1109/JPROC.2002.805829.

[13] Matthias Felleisen (1990): *On the expressive power of programming languages*. In: 3^{rd} *European Symposium on Programming (ESOP'90)*, *LNCS* 432, Springer, pp. 134–151, doi:10.1007/3-540-52592-0_60.

[14] Rob J. van Glabbeek (2012): *Musings on Encodings and Expressiveness*. In Bas Luttik & Michel A. Reniers, editors: *Proceedings Combined 19th International Workshop on Expressiveness in Concurrency and 9th Workshop on Structured Operational Semantics, EXPRESS/SOS 2012, Newcastle upon Tyne, UK, September 3, 2012.*, *EPTCS* 89, pp. 81–98, doi:10.4204/EPTCS.89.7.

[15] 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.

[16] Gregor Gößler & Joseph Sifakis (2003): *Priority Systems*. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf & Willem P. de Roever, editors: *Formal Methods for Components and Objects, Second International Symposium, FMCO 2003, Leiden, The Netherlands, November 4-7, 2003, Revised Lectures*, *Lecture Notes in Computer Science* 3188, Springer, pp. 314–329, doi:10.1007/978-3-540-30101-1_15.

[17] Gregor Gössler & Joseph Sifakis (2005): *Composition for component-based modeling*. *Science of Computer Programming* 55(1–3), pp. 161–183, doi:10.1016/j.scico.2004.05.014. Formal Methods for Components and Objects: Pragmatic aspects and applications.

[18] Kohei Honda, Vasco T. Vasconcelos & Makoto Kubo (1998): *Language primitives and type discipline for structured communication-based programming*. In Chris Hankin, editor: *Programming Languages and Systems*, *LNCS* 1381, Springer Berlin Heidelberg, pp. 122–138, doi:10.1007/BFb0053567.

[19] Hans Hüttel, Ivan Lanese, Vasco T. Vasconcelos, Luís Caires, Marco Carbone, Pierre-Malo Deniélou, Dimitris Mostrous, Luca Padovani, António Ravara, Emilio Tuosto, Hugo Torres Vieira & Gianluigi Zavattaro (2016): *Foundations of Session Types and Behavioural Contracts*. *ACM Comput. Surv.* 49(1), pp. 3:1–3:36, doi:10.1145/2873052.

[20] Robin Milner (1989): *Communication and Concurrency*. Prentice Hall International Series in Computer Science, Prentice Hall.

[21] MohammadReza Mousavi, Iain Phillips, Michel A. Reniers & Irek Ulidowski (2009): *Semantics and expressiveness of ordered SOS*. *Information and Computation* 207, pp. 85–119, doi:10.1016/j.ic.2007.11.008.

[22] 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.

[23] David M. R. Park (1981): *Concurrency and Automata on Infinite Sequences*. *Proceedings of the 5th GI-Conference on Theoretical Computer Science*, pp. 167–183, doi:10.1007/BFb0017309.

[24] Jan J. M. M. Rutten (2000): *Universal coalgebra: a theory of systems*. *Theor. Comput. Sci.* 249(1), pp. 3–80, doi:10.1016/S0304-3975(00)00056-6.

[25] Joseph Sifakis (2005): *A Framework for Component-based Construction*. In: 3^{rd} *IEEE Int. Conf. on Software Engineering and Formal Methods (SEFM05)*, pp. 293–300, doi:10.1109/SEFM.2005.3. Keynote talk.

[26] Pawel Sobocinski (2009): *A non-interleaving process calculus for multi-party synchronisation*. In Filippo Bonchi, Davide Grohmann, Paola Spoletini & Emilio Tuosto, editors: *ICE*, *EPTCS* 12, pp. 87–98, doi:10.4204/EPTCS.12.6.

Higher-order Processes with Parameterization over Names and Processes

Xian Xu *
East China University of Science and Technology, China
xuxian@ecust.edu.cn

Parameterization extends higher-order processes with the capability of abstraction and application (like those in lambda-calculus). This extension is strict, i.e., higher-order processes equipped with parameterization is computationally more powerful. This paper studies higher-order processes with two kinds of parameterization: one on names and the other on processes themselves. We present two results. One is that in presence of parameterization, higher-order processes can encode first-order (name-passing) processes in a quite neat fashion, in contrast to the fact that higher-order processes without parameterization cannot encode first-order processes at all. In the other result, we provide a simpler characterization of the (standard) context bisimulation for higher-order processes with parameterization, in terms of the normal bisimulation that stems from the well-known normal characterization for higher-order calculus. These two results demonstrate more essence of the parameterization method in the higher-order paradigm toward expressiveness and behavioural equivalence.

keywords: Parameterization, Context bisimulation, Higher-order, First-order, Processes

1 Introduction

In concurrent systems, higher-order means that processes communicate by means of process-passing (i.e., program-passing), whereas first-order means that processes communicate through name-passing (i.e., reference-passing). Parameterization originates from lambda-calculus (which is itself of higher-order nature), and enables processes, in a concurrent setting, to do abstraction and application in a way similar to that of lambda-calculus. Say P is a higher-order process, then an abstraction $\langle U\rangle P$ means abstracting the variable U in P to obtain somewhat a function (like $\lambda U.P$ in terms of lambda-calculus), and correspondingly an application $(\langle U\rangle P)\langle K\rangle$ means applying process K to the abstraction and obtaining an instantiation $P\{K/U\}$ (i.e., replacing each variable U in P with K, like $(\lambda U.P)K$ in terms of lambda-calculus). There are basically two kinds of parameterization: parameterization on names and parameterization on processes. In the former, U is a name variable and K is a concrete name. In the latter, U is a process variable and K is a concrete process. Parameterization is a natural way to extend the capacity of higher-order processes and this extension is strict, that is, the computational power strictly increases with the help of parameterization [12]. In this paper, we study higher-order processes in presence of parameterization.

Comparison between higher-order and first-order processes is a frequent topic in concurrency theory. Such comparison, for example, asks whether higher-order processes can correctly express first-order processes, or vice versa. It is well known that first-order processes can elegantly encode higher-order processes [19, 22]; the converse is however not quite the case. As the first issue, this paper addresses how to encode first-order processes with higher-order processes (equipped with parameterization).

*This work has been supported by project ANR 12IS02001 PACE and NSF of China (61261130589, 61472239, 61572318).

D. Gebler and K. Peters (Eds.): Combined Workshop on Expressiveness in
Concurrency and Structural Operational Semantics (EXPRESS/SOS 2016).
EPTCS 222, 2016, pp. 15–29, doi:10.4204/EPTCS.222.2

The very early work on using higher-order process to interpret first-order ones is contributed by Thomsen [24], who proposed a prototype encoding of first-order processes with higher-order processes with the relabelling operator (like that in CCS [16]). This encoding uses a gadget called wire to mimic the function of a name in the higher-order setting, and essentially employs the relabelling to make the wires work properly so as to fulfill the role of names. Due to the arbitrary ability of changing names (e.g., from global to local), the encoding has a correct operational correspondence (i.e., the correspondence between the processes before and after the encoding), but is very hard to analyze for full abstraction (i.e., the first-order processes are equivalent if and only if their encodings are; the 'if only' direction is called soundness and the other direction is called completeness). Unfortunately, without the relabelling operator, the basic higher-order process (which has the elementary operators including input, output, parallel composition and restriction) is not capable of encoding first-order processes [25]. In the literature, several variants of higher-order processes are exploited to encode first-order processes. In [22], an asynchronous higher-order calculus with parameterization on names is used to compile the asynchronous localized π-calculus (a variant of the first-order π-calculus [17]). This encoding depends heavily on the notions of 'localized' which means only the output capability of a name can be communicated during interactions, and 'asynchronous' which means the output is non-blocking. Though technically a nice reference, intuitively because this variant of π-calculus is less expressive than the full π-calculus, it is not very surprising that the higher-order processes with parameterization on names can interpret it faithfully, i.e., fully abstract with respect to barbed congruence. Then in [28], we explore the encoding of the full π-calculus using higher-order processes with parameterization on names. In that effort, we construct an encoding that harnesses the idea of Thomsen's encoding and show that it is complete. In [2], Bundgaard et al. use the HOMER to translate the name-passing π-calculus. This translation is possible because a HOMER process can, in a way quite different from parameterization, operates names in the continuation processes (resources), and this allows flexibility so that names can be communicated in an intermediate fashion. In [11], Kouzapas et al. propose fully abstract encodings concerning first-order processes and session typed higher-order processes. Their encodings use session types to govern communications and show that in the context of session types, first-order and higher-order processes are equally expressive. This work is well related to those mentioned above (and that in this paper), though the context is quite different (i.e., session typed processes).

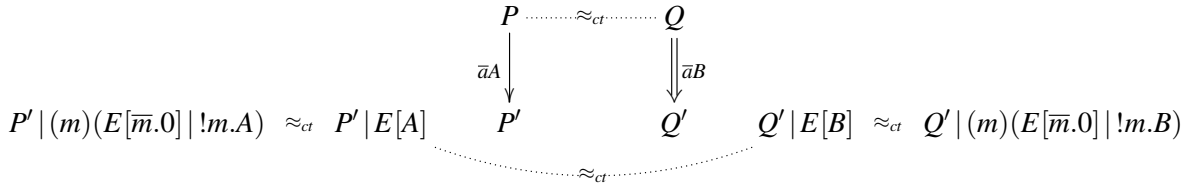
Despite the extensive research on encoding first-order processes with (variant) higher-order processes, the following question has remained open: *Is there an encoding of first-order processes by the higher-order processes with the capability of parameterization?* This question is important in two aspects. One is that parameterization brings about the core of lambda-calculus to higher-order concurrency, so it appears reasonable for such an extension to be able to express first-order processes which has long been shown to be capable of expressing the lambda-calculus. Knowing how this can be achieved would be interesting. The other is that the converse has a almost standard encoding method, i.e., encoding variants of higher-order processes with first-order processes. Yet higher-order processes are still short of an effective way to express first-order ones. Resolving this can also provide (technical) reference for practical work beyond the encoding itself.

Closely related with the first issue on expressiveness, the second issue this paper deals with is the characterization of bisimulation on higher-order processes. Bisimulation theory is a pivotal part of a process model, including the higher-order models, concerning which the almost standard behavioral equivalence is the context bisimulation [19]. The central idea of context bisimulation is that when comparing output actions, the transmitted process and the residual process (i.e., the process obtained after sending a process) are considered at the same time, rather than separately (like in the applicative higher-order bisimulation proposed by Thomsen [23, 24]). For example (for simplicity we do not consider local

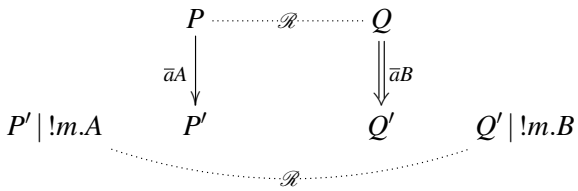
names), if P and Q are context bisimilar and $P \xrightarrow{\overline{a}A} P'$ (i.e., P outputs A on a and becomes P'), then $Q \xRightarrow{\overline{a}B} Q'$ (i.e., Q outputs B on a possibly involving some internal actions and becomes Q'), and for every (receiving) environment $E[\cdot]$, $P' \mid E[A]$ and $Q' \mid E[B]$ are still context bisimilar (here $\mid$ denotes concurrency, and $E[A]$ means putting A in the environment E). However, in its original form, context bisimulation suffers from inconvenience to use, because it calls for checking with regard to every possible receiving environment. This leads to works on the simpler characterization, called normal bisimulation, of the context bisimulation. The central idea of normal bisimulation, proposed by Sangiorgi [19, 22], is that instead of checking with a general process in input and a general context in output, one only needs to comply with the matching of some special process or context, specifically a class of terms called triggers. To meet this challenge, a crucial so-called factorization theorem is used to circumvent technical difficulty. We briefly explain how normal bisimulation is designed in the basic higher-order processes. In particular, the factorization states the following property, where $\approx_{ct}$ denotes context bisimulation, and $\overline{m}.P$ and $m.P$ are CCS-like prefixes in which the communicated contents are not important [22].

$$E[A] \approx_{ct} (m)(E[\overline{m}.0] \mid !m.A)$$

One can clearly identify the reposition of the process A of interest, which in fact captures the core of the property: move A to a new position as a repository, which in turn can be retrieved as many times as needed in the original environment E, with the help of the pointer undertaken by the fresh channel m (called trigger). Inspired by the factorization, normal bisimulation can be developed. We take the output as an example (input is similar), and restriction operation in output is omitted for the sake of simplicity. As stated above, context bisimulation requires the following chasing diagram, which is now extended with an application of the factorization.



Since context bisimulation $\approx_{ct}$ is a congruence, one can cancel the common part of (the leftmost) $P' \mid (m)(E[\overline{m}.0] \mid !m.A)$ and (the rightmost) $Q' \mid (m)(E[\overline{m}.0] \mid !m.B)$, and simply requires that $P' \mid !m.A$ and $Q' \mid !m.B$ are related, without fearing losing any discriminating power. This in turn leads to the following requirement in normal bisimulation (assuming $\mathscr{R}$ is a normal bisimulation).



Subsequent works attempt to extend the normal bisimulation to variants of higher-order processes. In Sangiorgi's initial work [19], normal bisimulation is also obtained for higher-order processes with parameterization. That characterization , however, is made in the presence of first-order processes (i.e., name-passing), and thus not very convincing with regard to the inner complexity of context bisimulation in presence of parameterization. In [26], we revisited this issue and show that in a purely higher-order setting (viz., no name-passing at all), parameterization on processes does not deprive one of the convenience of normal bisimulation. Although the idea is inspired by the original work of Sangiorgi, the proof approach is more direct. In [14, 15], Lenglet et al. study higher-order processes with passivation (i.e., the process in the output position may evolve), and report a normal bisimulation for a sub-calculus without the restriction operator, but that characterization has somewhat a different flavor, since the higher-order

bisimulation [24] rather than the context bisimulation is taken. Though these works carry out insightful research and give meaningful references, it is currently still not clear how to construct a simple characterization of context bisimulation based on parameterization over names, and this raises the following fundamental question: *Does higher-order processes with parameterization on names have a normal bisimulation?* In the second part of this paper, we move further from [19, 26], and offer a normal bisimulation for higher-order processes in the setting of parameterization over both names and processes.

Contribution In summary, our contribution of this work is as follows.

- We show that the extension with parameterization (on both names and processes) allows higher-order processes to interpret first-order processes in a surprisingly concise yet elegant manner. Such kind of encoding is of a somewhat dissimilar flavor, and moreover not possible in absence of parameterization. We give the detailed encoding strategy, and prove that it satisfies a number of desired properties well-known in the field.
  The idea of the encoding in this paper is quite different from our abovementioned work in [28], where we build an encoding that allows parameterization merely on names (i.e., no parameterization on processes). The soundness of that encoding is not very satisfying, which in a sense defeats some purpose of the encoding, and this actually precipitates the work here.
- We establish the normal bisimulation, as an effectively simpler characterization of context bisimulation, for higher-order processes with both kinds of parameterization. This normal bisimulation extends those for higher-order processes without parameterization, particularly in the manipulation of abstractions on names. As far as we are concerned, similar characterization has not been reported before.
  That the processes are purely higher-order (that is, without name-passing) improves the result in [19], and articulates that the characterization based on normal bisimulation is a property independent of first-order name-passing. Moreover, this does not contradict the argument in [26] that there is little hope that normal bisimulation exists in higher-order processes with (only) parameterization on names, because here the processes are capable of parameterization on processes as well (though still higher-order).

Organization The remainder of this paper is organized as below. In Section 2, we introduce the calculi and a notion of encoding used in this paper. In Section 3, we present the encoding from first-order processes to higher-order processes with parameterization, and discuss its properties. In Section 4, we define the normal bisimulation for higher-order processes with parameterization, and prove that it truly characterizes context bisimulation. Section 5 concludes this work and point out some further directions.

2 Preliminary

In this section we present the basic definitions and notations used in this work.

2.1 Calculus π

The first-order (name-passing) pi-calculus, π, is proposed by Milner et al. [17]. For the sake of simplicity, throughout the paper, names (ranged over by m,n,u,v,w) are divided into two classes: name constants (ranged over by a,b,c,d,e) and name variables (ranged over by x,y,z) [3, 4, 6]. The grammar is as below with the constructs having their standard meaning. We note that guarded input replication is used instead of general replication, and this does not decrease the expressiveness [21] [7].

$$P,Q := 0 \;\Big|\; m(x).P \;\Big|\; \overline{m}n.P \;\Big|\; (c)P \;\Big|\; P \,|\, Q \;\Big|\; !m(x).P$$

A name constant a is bound (or local) in $(a)P$ and free (or global) otherwise. A name variable x is bound in $a(x).P$ and free otherwise. Respectively $\mathrm{fn}(\cdot), \mathrm{bn}(\cdot), \mathrm{n}(\cdot), \mathrm{fnv}(\cdot), \mathrm{bnv}(\cdot), \mathrm{nv}(\cdot)$ denote free name constants, bound name constants, names, free name variables, bound name variables, and name variables in a set of processes. A name is fresh if it does not appear in any process under discussion. By default, closed processes are considered, i.e., those having no free variables. As usual, here are a few derived operators: $\overline{a}(d).P \stackrel{\text{def}}{=} (d)\overline{a}d.P$, $a.P \stackrel{\text{def}}{=} a(x).P\ (x \notin fv(P))$, $\overline{a}.P \stackrel{\text{def}}{=} \overline{a}(d).P\ (d \notin fn(P))$; $\tau.P \stackrel{\text{def}}{=} (a)(a.P \mid \overline{a}.0)$ (a fresh). A trailing 0 process is usually omitted. We denote tuples by a tilde. For tuple $\widetilde{n}$: $|\widetilde{n}|$ denotes its length; $m\widetilde{n}$ denotes incorporating m. Multiple restriction $(c_1)(c_2)\cdots(c_k)E$ is abbreviated as $(\widetilde{c})E$. Substitution $P\{n/m\}$ is a mapping that replaces m with n in P while keeping the rest unchanged. A context C is a process with some subprocess replaced by the hole $[\cdot]$, and $C[A]$ is the process obtained by filling in the hole by A.

The semantics of π is defined by the LTS (Labelled Transition System) below.

$$\frac{}{a(x).P \xrightarrow{a(b)} P\{b/x\}} \qquad \frac{}{\overline{a}b.P \xrightarrow{\overline{a}b} P} \qquad \frac{}{!a(x).P \xrightarrow{a(b)} P\{b/x\} \mid !a(x).P} \qquad \frac{P \xrightarrow{\lambda} P'}{(c)P \xrightarrow{\lambda} P'}\ c \notin n(\lambda)$$

$$\frac{P \xrightarrow{\overline{a}c} P'}{(c)P \xrightarrow{\overline{a}(c)} P'}\ c \neq a \qquad \frac{P \xrightarrow{\lambda} P'}{P \mid Q \xrightarrow{\lambda} P' \mid Q}\ bn(\lambda) \cap fn(Q) = \emptyset \qquad \frac{P \xrightarrow{a(b)} P' \quad Q \xrightarrow{\overline{a}b} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad \frac{P \xrightarrow{a(b)} P' \quad Q \xrightarrow{\overline{a}(b)} Q'}{P \mid Q \xrightarrow{\tau} (b)(P' \mid Q')}$$

Actions, ranged over by λ, α, comprise internal move τ, and visible ones: input ($a(b)$), output ($\overline{a}b$) and bound output ($\overline{a}(c)$). We note that actions occur only on name constants, and a communicated name is also a constant. We denote by $\equiv$ the standard structural congruence [17] [22], which is the smallest relation satisfying the monoid laws for parallel composition, commutative laws for both composition and restriction, and a distributive law $(c)(P \mid Q) \equiv (c)P \mid Q$ (if $c \notin fn(Q)$). We use $\Longrightarrow$ for the reflexive transitive closure of $\xrightarrow{\tau}$, $\stackrel{\lambda}{\Longrightarrow}$ for $\Longrightarrow \xrightarrow{\lambda} \Longrightarrow$, and $\stackrel{\widehat{\lambda}}{\Longrightarrow}$ for $\stackrel{\lambda}{\Longrightarrow}$ if λ is not τ, and $\Longrightarrow$ otherwise. A process P is divergent, denoted $P^{\Uparrow}$, if it has an infinite sequence of τ actions.

Throughout the paper, we use the following standard notion of ground bisimulation [4, 17, 22].

Definition 1. A ground bisimulation is a symmetric relation $\mathscr{R}$ on π processes s.t. whenever $P \mathscr{R} Q$ the following property holds: If $P \xrightarrow{\alpha} P'$ where α is $a(b)$, $\overline{a}b$, $\overline{a}(b)$, or τ, then $Q \stackrel{\widehat{\alpha}}{\Longrightarrow} Q'$ for some Q' and $P' \mathscr{R} Q'$. Ground bisimilarity, $\approx_g$, is the largest ground bisimulation.

We denote by $\sim_g$ the strong ground bisimilarity (i.e., replacing $\stackrel{\widehat{\alpha}}{\Longrightarrow}$ with $\xrightarrow{\alpha}$ in the definition). It is well-known that $\approx_g$ is a congruence [4, 22], and coincides with the so-called local bisimilarity as defined below [5, 25].

Definition 2. Local bisimilarity $\approx_l$ is the largest symmetric local bisimulation relation $\mathscr{R}$ on π processes such that: (1) if $P \xrightarrow{\lambda} P'$, λ is not bound output, then $Q \stackrel{\widehat{\lambda}}{\Longrightarrow} Q'$ and $P' \mathscr{R} Q'$; (2) if $P \xrightarrow{\overline{a}(b)} P'$, then $Q \stackrel{\overline{a}(b)}{\Longrightarrow} Q'$, and for every R, $(b)(P' \mid R) \mathscr{R} (b)(Q' \mid R)$.

2.2 Calculus $\Pi^{D,d}$

For the sake of conciseness, we first define the basic higher-order calculus and then the extension with parameterizations.

2.2.1 Calculus Π

The basic higher-order (process-passing) calculus, Π, is defined by the following grammar in which the operators have their standard meaning. We denote by X, Y, Z process variables.

$$T, T' ::= 0 \mid X \mid u(X).T \mid \overline{u}T'.T \mid T \mid T' \mid (c)T \mid !u(X).T \mid !\overline{u}T'.T$$

We use $a.0$ for $a(X).0$, $\overline{a}.0$ for $\overline{a}0.0$, $\tau.P$ for $(a)(a.P \mid \overline{a}.0)$, and sometimes $\overline{a}[A].T$ for $\overline{a}A.T$. Like π, a tilde represents a tuple. We reuse the notations for names in π and additionally use fpv$(\cdot)$, bpv$(\cdot)$, pv$(\cdot)$ respectively to denote free process variables, bound process variables and process variables in a set of processes. Closed processes are those having no free variables. A higher-order substitution $T\{A/X\}$ replaces variable X with A and can be extended to tuples in the usual way. $E[\widetilde{X}]$ denotes E with (possibly) variables $\widetilde{X}$, and $E[\widetilde{A}]$ stands for $E\{\widetilde{A}/\widetilde{X}\}$. The guarded replications used in the grammar can actually be derived [13, 24], and we make them primitive for convenience. The semantics of Π is as below.

$$\frac{}{a(X).T \xrightarrow{a(A)} T\{A/X\}} \quad \frac{}{\overline{a}A.T \xrightarrow{\overline{a}A} T} \quad \frac{T \xrightarrow{\lambda} T'}{(c)T \xrightarrow{\lambda} (c)T'}\, c \notin n(\lambda) \quad \frac{T \xrightarrow{\lambda} T'}{T \mid T_1 \xrightarrow{\lambda} T' \mid T_1} \quad \frac{}{!\overline{a}A.T \xrightarrow{\overline{a}A} T \mid !\overline{a}A.T}$$

$$\frac{T \xrightarrow{(\widetilde{c})\overline{a}[A]} T'}{(d)T \xrightarrow{(d)(\widetilde{c})\overline{a}[A]} T'}\, d \in fn(A) - \{\widetilde{c}, a\} \quad \frac{T_1 \xrightarrow{a(A)} T_1', T_2 \xrightarrow{(\widetilde{c})\overline{a}[A]} T_2'}{T_1 \mid T_2 \xrightarrow{\tau} (\widetilde{c})(T_1' \mid T_2')} \quad \frac{}{!a(X).T \xrightarrow{a(A)} T\{A/X\} \mid !a(X).T}$$

We denote by α, λ the actions: internal move (τ), input ($a(A)$), output ($(\widetilde{c})\overline{a}A$) in which $\widetilde{c}$ is some local names carried by A during the output. We always assume no name capture with resort to α-conversion. The notations $\Longrightarrow$, $\stackrel{\lambda}{\Longrightarrow}$ and $\stackrel{\widehat{\lambda}}{\Longrightarrow}$ are similar to those in π. We also reuse $\equiv$ for the structural congruence in Π (and also $\Pi^{D,d}$ to be defined shortly) [22], and this shall not raise confusion under specific context.

2.2.2 Calculus $\Pi^{D,d}$

Parameterization extends Π with the syntax and semantics below. Symbol U_i (respectively, K_i) ($i = 1, ..., n$) is used as a meta-parameter of an abstraction (respectively, meta-instance of an application), and stands for a process variable or name variable (respectively, a process or a name).

Extension of syntax: $\quad \langle U_1, U_2, ..., U_n \rangle T \;\Big|\; T'\langle K_1, K_2, ..., K_n \rangle$

Extension of semantics: $\quad \dfrac{Q \equiv P \quad P \xrightarrow{\lambda} P' \quad P' \equiv Q'}{Q \xrightarrow{\lambda} Q'}$

Extension of structural congruence ($\equiv$): $\quad F\langle \widetilde{K} \rangle \equiv T\{\widetilde{K}/\widetilde{U}\}$ $\quad$ where $F \stackrel{\text{def}}{=} \langle \widetilde{U} \rangle T$ and $|\widetilde{U}| = |\widetilde{K}|$

We denote by $\langle U_1, U_2, ..., U_n \rangle T$ an n-ary abstraction in which $U_1, U_2, ..., U_n$ are the parameters to be instantiated during the application $T'\langle K_1, K_2, ..., K_n \rangle$ in which the parameters are replaced by instances $K_1, K_2, ..., K_n$. This application is modelled by an extensional rule for structural congruence as above, in combination with the usual LTS rule for structural congruence as well, so as to make the process engaged in application evolve effectively. The condition $|\widetilde{U}| = |\widetilde{K}|$ requires that the parameters and the instantiating objects should be equal in length.

Now parameterization on process is obtained by taking $\widetilde{U}, \widetilde{K}$ as $\widetilde{X}, \widetilde{T'}$ respectively, and parameterization on names is obtained by taking $\widetilde{U}, \widetilde{K}$ as $\widetilde{x}, \widetilde{u}$ respectively. The corresponding abstractions are sometimes called process abstraction and name abstraction respectively. For convenience, names are handled in the same way as that in π (so are the related notations). We denote by $\Pi^{D,d}$ the calculus Π extended with both kinds of parameterizations. Calculus $\Pi^{D,d}$ can be made more precise with the help of a type system [19] which however is not important for this work and not presented. We note that in $\langle U_1, U_2, ..., U_n \rangle T$, variables $U_1, U_2, ..., U_n$ are bound.

Throughout the paper, we reply on the following notion of context bisimulation [19, 20].

Definition 3. A symmetric relation $\mathscr{R}$ on $\Pi^{D,d}$ processes is a context bisimulation, if $P \mathscr{R} Q$ implies the following properties: (1) if $P \xrightarrow{\alpha} P'$ and α is $a(A)$ or τ, then $Q \stackrel{\widehat{\alpha}}{\Longrightarrow} Q'$ for some Q' and $P' \mathscr{R} Q'$;
(2) if $P \xrightarrow{(\widetilde{c})\overline{a}A} P'$ and A is a process abstraction or name abstraction or not an abstraction, then $Q \stackrel{(\widetilde{d})\overline{a}B}{\Longrightarrow} Q'$

for some B that is accordingly a process abstraction or name abstraction or not an abstraction, and moreover for every $E[X]$ s.t. $\{\widetilde{c},\widetilde{d}\}\cap fn(E)=\emptyset$ it holds that $(\widetilde{c})(E[A]\,|\,P')\;\mathscr{R}\;(\widetilde{d})(E[B]\,|\,Q')$. Context bisimilarity, written $\approx_{ct}$, is the largest context bisimulation.

We note that the matching for output in context bisimulation is required to bear the same kind of communicated process as compared to the simulated action. Relation $\sim_{ct}$ denotes the strong context bisimilarity. As is well-known, $\approx_{ct}$ is a congruence [19, 20].

2.3 A notion of encoding

We define a notion of encoding in this section. We assume a process model $\mathscr{L}$ is a triplet $(\mathscr{P},\rightarrow,\approx)$, where $\mathscr{P}$ is the set of processes, $\rightarrow$ is the LTS with a set $\mathscr{A}$ of actions, and $\approx$ is a behavioral equivalence. Given $\mathscr{L}_i \stackrel{\text{def}}{=} (\mathscr{P}_i,\rightarrow_i,\approx_i)$ (i=1,2), an encoding from $\mathscr{L}_1$ to $\mathscr{L}_2$ is a function $[\![\cdot]\!]:\mathscr{P}_1\longrightarrow\mathscr{P}_2$ that satisfies some set of criteria. Notation $[\![\mathscr{P}_1]\!]$ stands for the image of the $\mathscr{L}_1$-processes inside $\mathscr{L}_2$ under the encoding. It should be clear that $[\![\mathscr{P}_1]\!]\subseteq\mathscr{P}_2$. We use $\dot{\approx}_2$ to denote the behavioural equivalence $\approx_2$ restricted to $[\![\mathscr{P}_1]\!]$ [10]. The following criteria set (Definition 4) used in this paper stems from [12] (the variant [12] provides is based on [8]). As is known, encodability enjoys transitivity [12]. We will show that the encoding in Section 3 satisfies all the criteria in Definition 4 except adequacy (1a).

Definition 4 (Criteria for encodings)**. Static criteria**: (1) Compositionality. *For any k-ary operator op of $\mathscr{L}_1$, and all $P_1,...,P_k\in\mathscr{P}_1$, $[\![op(P_1,...,P_k)]\!]=C_{op}[[\![P_1]\!],...,[\![P_k]\!]]$ for some (multihole) context $C_{op}[\cdots]\in\mathscr{P}_2$;*

Dynamic criteria: (1a) Adequacy. *$P\approx_1 P'$ implies $[\![P]\!]\approx_2[\![P']\!]$. This is also known as* soundness*. The converse is known as* completeness*;* (1b) Weak adequacy (or weak soundness). *$P\approx_1 P'$ implies $[\![P]\!]\dot{\approx}_2[\![P']\!]$;* (2) Divergence-reflecting. *If $[\![P]\!]$ diverges, so does P.*

Adequacy (1a) obviously entails weak adequacy (1b), since $\approx_2$ allows more processes in the target model $\mathscr{L}_2$ (thus more variety of contexts). Yet weak adequacy is still useful because it may be too strong if one requires the encoding process to be compatible with all kinds of contexts in the target model. For instance, in order to achieve first-order interactions in a higher-order target model, it appears quite demanding to require equivalence under all kinds of input because the target higher-order model may have more powerful computation ability (so it can feed a much involved input). So sometimes using limited contexts in the target model may be sufficient to meet the goal of the encoding.

It is worthwhile to note that the criteria are short of those for operational correspondence. Although generally soundness and completeness appear not very informative in absence of operational correspondence (and the others) [9, 18], arguably we make this choice in this work out of the following consideration. The criteria for operational correspondence used in [8], though proven useful in many models, appear not quite convenient when discussing encodings into higher-order models [12], since (for example) the case of input can be hard to comply with the criteria due to the increased complexity in the environment (namely in the context of the target higher-order model). After all, here it seems more important to have the soundness and completeness properties eventually (w.r.t. the canonical bisimulation equivalences in the source and target models), likely in a different manner of operational correspondence. Notwithstanding, we will discuss the operational correspondence of the encoding in Section 3. Moreover, as will be seen, the concrete operational correspondence in there somehow strengthens the criteria of operational correspondence (and related concepts) used in [8, 12] (in [8] the criteria are not action-labelled and thus the notion of success sensitiveness is contrived; in [12] a labelled variant criteria is posited to its purpose). Beyond the scope of this paper, it would be intriguing to examine the possibility of formally pinning down some variant criteria of operational correspondence having vantage for higher-order (process) models.

3 Encoding π into $\Pi^{D,d}$

We show that π can be encoded in $\Pi^{D,d}$.

3.1 The encoding

We have the encoding defined as below (being homomorphism on the other operators, except that the encoding of input guarded replication is defined as $[\![!m(x).P]\!] \stackrel{\text{def}}{=} ![\![m(x).P]\!]$).

$$\begin{array}{rcl} [\![m(x).P]\!] & \stackrel{\text{def}}{=} & m(Y).Y\langle\langle x\rangle [\![P]\!]\rangle \\ [\![\overline{m}n.Q]\!] & \stackrel{\text{def}}{=} & \overline{m}[\langle Z\rangle (Z\langle n\rangle)].[\![Q]\!] \end{array}$$

The encoding above uses both name parameterization and process parameterization. Typically one can assume that Y and Z are fresh for simplicity, but this is not essential, because these variables are bound and can be α-converted whenever necessary, and moreover the encoded first-order process does not have higher-order variables. Specifically, the encoding of an output 'transmits' the name to be sent (i.e., n) in terms of a process parameterization (i.e., $\langle Z\rangle (Z\langle n\rangle)$) that, once being received by the encoding of an input, is instantiated by a name-parameterized term (i.e., $\langle x\rangle [\![P]\!]$), which then can apply n on x in the encoding of P, thus fulfilling 'name-passing'. Below we give an example. Suppose $P \stackrel{\text{def}}{=} (c)(a(x).\overline{x}c.P_1)$ and $Q \stackrel{\text{def}}{=} (d)(\overline{a}d.d(y).Q_1)$. So

$$\begin{array}{rcl} P\,|\,Q & \stackrel{\tau}{\to} & (d)((c)(\overline{d}c.P_1\{d/x\})\,|\,d(y).Q_1) \\ & \stackrel{\tau}{\to} & (dc)(P_1\{d/x\}\,|\,Q_1\{c/y\}) \end{array}$$

The encoding and interactions of $[\![P\,|\,Q]\!]$ are as below. For clarity, we use **bold font** to indicate the evolving part during a communication.

$$\begin{array}{rcl} [\![P\,|\,Q]\!] & \equiv & (c)(a(Y).Y\langle\langle x\rangle [\![\overline{x}c.P_1]\!]\rangle) \mid (d)(\overline{a}[\langle Z\rangle (Z\langle d\rangle)].[\![d(y).Q_1]\!]) \\ & \stackrel{\tau}{\to} & (d)\big((c)((\langle \boldsymbol{Z}\rangle (\boldsymbol{Z}\langle \boldsymbol{d}\rangle))\langle\langle \boldsymbol{x}\rangle [\![\overline{x}\boldsymbol{c}.\boldsymbol{P_1}]\!]\rangle) \mid [\![d(y).Q_1]\!]\big) \\ & \equiv & (d)\big((c)([\![\overline{x}\boldsymbol{c}.\boldsymbol{P_1}]\!]\{\boldsymbol{d/x}\}) \mid [\![d(y).Q_1]\!]\big) \\ & \equiv & (d)\big((c)((\overline{x}[\langle \boldsymbol{Z}\rangle (\boldsymbol{Z}\langle \boldsymbol{c}\rangle)].[\![\boldsymbol{P_1}]\!])\{\boldsymbol{d/x}\}) \mid d(Y).Y\langle\langle y\rangle [\![Q_1]\!]\rangle\big) \\ & \equiv & (d)\big((c)((\overline{d}[\langle \boldsymbol{Z}\rangle (\boldsymbol{Z}\langle \boldsymbol{c}\rangle)].[\![\boldsymbol{P_1}]\!]\{\boldsymbol{d/x}\})) \mid d(Y).Y\langle\langle y\rangle [\![Q_1]\!]\rangle\big) \\ & \stackrel{\tau}{\to} & (dc)\big([\![P_1]\!]\{d/x\} \mid (\langle \boldsymbol{Z}\rangle (\boldsymbol{Z}\langle \boldsymbol{c}\rangle))(\langle\langle \boldsymbol{y}\rangle [\![\boldsymbol{Q_1}]\!]\rangle)\big) \\ & \equiv & (dc)\big([\![P_1]\!]\{d/x\} \mid [\![Q_1]\!]\{c/y\}\big) \\ & \equiv & (dc)\big([\![P_1\{d/x\}]\!] \mid [\![Q_1\{c/y\}]\!]\big) \end{array}$$

Apparently the encoding is compositional, preserves the (free) names, and moreover divergence-reflecting (since the encoding does not introduce any extra internal action), as stated in the follow-up lemma whose proof is a standard induction.

Lemma 5. *Assume P is a π process. The encoding above from π to $\Pi^{D,d}$ is compositional and divergence-reflecting; moreover $[\![P]\!]\{n/m\} \equiv [\![P\{n/m\}]\!]$.*

3.2 Operational correspondence

We have the following properties clarifying the correspondence of actions before and after the encoding. To delineate some case of the operational correspondence in terms of certain special input, i.e., a

trigger, we define $Tr_m^D \stackrel{\text{def}}{=} \langle Z\rangle\overline{m}Z$ in which m is assumed to be fresh (it will also be used in Section 4, but here simply allows for more flexible characterization of the operational correspondence). We note that sometimes existential quantification is omitted when it is clear from context.

Lemma 6. *Suppose P is a π process. (1) If $P \xrightarrow{a(b)} P'$, then $[\![P]\!] \xrightarrow{a(\langle Z\rangle(Z\langle b\rangle))} T$ and $T \sim_{ct} [\![P']\!]$; (2) If $P \xrightarrow{a(b)} P'$, then $[\![P]\!] \xrightarrow{a(Tr_m^D)} T$ and $(m)(T \mid !m(Y).Y\langle b\rangle) \approx_{ct} [\![P']\!]$; (3) If $P \xrightarrow{\overline{a}b} P'$, then $[\![P]\!] \xrightarrow{\overline{a}[\langle Z\rangle(Z\langle b\rangle)]} T$ and $T \sim_{ct} [\![P']\!]$; (4) If $P \xrightarrow{\overline{a}(b)} P'$, then $[\![P]\!] \xrightarrow{(b)\overline{a}[\langle Z\rangle(Z\langle b\rangle)]} T$ and $T \sim_{ct} [\![P']\!]$; (5) If $P \xrightarrow{\tau} P'$, then $[\![P]\!] \xrightarrow{\tau} T$ and $T \sim_{ct} [\![P']\!]$.*

The converse is as below.

Lemma 7. *Suppose P is a π process. (1) If $[\![P]\!] \xrightarrow{a(\langle Z\rangle(Z\langle b\rangle))} T$, then $P \xrightarrow{a(b)} P'$ and $T \sim_{ct} [\![P']\!]$; (2) If $[\![P]\!] \xrightarrow{a(Tr_m^D)} T$, then $P \xrightarrow{a(b)} P'$ and $(m)(T \mid !m(Y).Y\langle b\rangle) \approx_{ct} [\![P']\!]$; (3) If $[\![P]\!] \xrightarrow{\overline{a}[\langle Z\rangle(Z\langle b\rangle)]} T$, then $P \xrightarrow{\overline{a}b} P'$ and $T \sim_{ct} [\![P']\!]$; (4) If $[\![P]\!] \xrightarrow{(b)\overline{a}[\langle Z\rangle(Z\langle b\rangle)]} T$, then $P \xrightarrow{\overline{a}(b)} P'$ and $T \sim_{ct} [\![P']\!]$; (5) If $[\![P]\!] \xrightarrow{\tau} T$, then $P \xrightarrow{\tau} P'$ and $T \sim_{ct} [\![P']\!]$.*

Lemma 6 and Lemma 7 can be proven in a similar fashion (details can be found in [27]), and moreover be lifted to the weak situation. That is, if one replaces strong transitions (single arrows) with weak transitions (double arrows), the results still hold ($\sim_{ct}$ retains because the encoding does not bring any extra internal action); see [19,22] for a reference. We will however simply refer to these two lemmas in related discussions.

3.3 Soundness

In this section, we discuss the soundness of the encoding. First of all, it is unfortunate that the soundness of the encoding is not true. To see this, take the processes R_1 and R_2 below. We recall that the CCS-like prefixes are defined as usual, i.e., $a.P \stackrel{\text{def}}{=} a(x).P$ $(x \notin \text{n}(P))$, $\overline{a}.P \stackrel{\text{def}}{=} (c)\overline{a}c.P$ $(c \notin \text{n}(P))$; sometimes we trim the trailing 0, e.g., a stands for $a.0$ and $\overline{a}$ for $\overline{a}.0$.

$$R_1 \stackrel{\text{def}}{=} (b)(a.\overline{b} \mid b.\overline{c}) \qquad R_2 \stackrel{\text{def}}{=} (b)(a.\overline{b} \mid b.\overline{c} \mid b.\overline{c})$$

Obviously, R_1 and R_2 are ground bisimilar. Now we examine their encodings.

$$\begin{array}{rcl} [\![R_1]\!] & \equiv & (b)(a(Y).Y\langle\langle x\rangle[\![\overline{b}]\!]\rangle \mid b(Y).Y\langle\langle x\rangle[\![\overline{c}]\!]\rangle) \\ [\![R_2]\!] & \equiv & (b)(a(Y).Y\langle\langle x\rangle[\![\overline{b}]\!]\rangle \mid b(Y).Y\langle\langle x\rangle[\![\overline{c}]\!]\rangle \mid b(Y).Y\langle\langle x\rangle[\![\overline{c}]\!]\rangle) \end{array}$$

We show that $[\![R_1]\!]$ and $[\![R_2]\!]$ are not context bisimilar. Define $T \stackrel{\text{def}}{=} (m)(\overline{a}[\langle Z\rangle\overline{m}Z] \mid m(X).(X\langle d\rangle \mid X\langle d\rangle)$. Then $(a)([\![R_1]\!] \mid T)$ and $(a)([\![R_2]\!] \mid T)$ can be distinguished. The latter can fire two output on c, whereas the former cannot, as shown below.

$$\begin{array}{rcl} (a)([\![R_1]\!] \mid T) & \xrightarrow{\tau} \sim_{ct} & (m)((b)(\overline{m}[\langle x\rangle[\![\overline{b}]\!]] \mid b(Y).Y\langle\langle x\rangle[\![\overline{c}]\!]\rangle) \mid m(X).(X\langle d\rangle \mid X\langle d\rangle)) \\ & \xrightarrow{\tau} \sim_{ct} & (b)(b(Y).Y\langle\langle x\rangle[\![\overline{c}]\!]\rangle \mid [\![\overline{b}]\!] \mid [\![\overline{b}]\!]) \\ & \equiv & (b)(b(Y).Y\langle\langle x\rangle[\![\overline{c}]\!]\rangle \mid (e)\overline{b}[\langle Z\rangle(Z\langle e\rangle)] \mid [\![\overline{b}]\!]) \\ & \xrightarrow{\tau} \sim_{ct} & (b)([\![\overline{c}]\!] \mid [\![\overline{b}]\!]) \\ & \equiv & (b)((f)\overline{c}[\langle Z\rangle(Z\langle f\rangle)] \mid [\![\overline{b}]\!]) \\ & \xrightarrow{(f)\overline{c}[\langle Z\rangle(Z\langle f\rangle)]} \sim_{ct} & 0 \end{array}$$

$$\begin{array}{rcl}
(a)([\![R_2]\!]\,|\,T) \xrightarrow{\tau} \sim_{ct} & (m)((b)(\overline{m}[\langle x\rangle[\![\overline{b}]\!]]\,|\,b(Y).Y\langle\langle x\rangle[\![\overline{c}]\!]\rangle\,|\,b(Y).Y\langle\langle x\rangle[\![\overline{c}]\!]\rangle)\,|\,m(X).(X\langle d\rangle\,|\,X\langle d\rangle)) \\
\xrightarrow{\tau} \sim_{ct} & (b)(b(Y).Y\langle\langle x\rangle[\![\overline{c}]\!]\rangle\,|\,b(Y).Y\langle\langle x\rangle[\![\overline{c}]\!]\rangle\,|\,[\![\overline{b}]\!]\,|\,[\![\overline{b}]\!]) \\
\equiv & (b)(b(Y).Y\langle\langle x\rangle[\![\overline{c}]\!]\rangle\,|\,b(Y).Y\langle\langle x\rangle[\![\overline{c}]\!]\rangle\,|\,(e)\overline{b}[\langle Z\rangle(Z\langle e\rangle)]\,|\,(e)\overline{b}[\langle Z\rangle(Z\langle e\rangle)]) \\
\xrightarrow{\tau}\xrightarrow{\tau} \sim_{ct} & [\![\overline{c}]\!]\,|\,[\![\overline{c}]\!] \\
\equiv & (f)\overline{c}[\langle Z\rangle(Z\langle f\rangle)]\,|\,(f)\overline{c}[\langle Z\rangle(Z\langle f\rangle)] \\
\xrightarrow{(f)\overline{c}[\langle Z\rangle(Z\langle f\rangle)]} \sim_{ct} & (f)\overline{c}[\langle Z\rangle(Z\langle f\rangle)] \\
\xrightarrow{(f)\overline{c}[\langle Z\rangle(Z\langle f\rangle)]} \sim_{ct} & 0
\end{array}$$

Intuitively, the reason general soundness does not hold is that context bisimulation is somewhat more discriminating in the target higher-order calculus, which can have more flexibility when dealing with blocks of processes in presence of parameterization (e.g., some subprocess can be sent as needed). This is however beyond the capability of a first-order process.

In spite of the falsity of soundness in general, we can have a somewhat weaker yet still sensible soundness. Remember that our main goal is to achieve first-order concurrency in the higher-order model, so maybe we do not need to be so demanding when coping with the encodings of first-order processes, that is, when testing an encoding process with an input, one can focus on those representing a name instead of a general one. Then it is expected that soundness will hold under this assumption. Fortunately, this is indeed true.

We have the following lemma stating the weak soundness of the encoding. Recall that $\dot{\approx}_{ct}$ is the $\approx_{ct}$ restricted to the image of the encoding (i.e., the processes in the target model that have reverse-image w.r.t. the encoding).

Lemma 8. *Suppose P is a π process. Then $P \approx_g Q$ implies $[\![P]\!] \dot{\approx}_{ct} [\![Q]\!]$.*

Proof. We show that $\mathscr{R} \stackrel{\text{def}}{=} \{([\![P]\!],[\![Q]\!])\,|\,P \approx_g Q\} \cup \dot{\approx}_{ct}$ is a context bisimulation up-to context and $\sim_{ct}$ (we refer the reader to, for example, [1, 22] and the references therein for the up-to proof technique for establishing bisimulations; we note that using $\sim_{ct}$ here is sufficient since it is stronger than $\dot{\sim}_{ct}$, i.e., $\sim_{ct}$ restricted to the image of the encoding).

Suppose $[\![P]\!]\,\mathscr{R}\,[\![Q]\!]$. There are several cases, where Lemma 6 and Lemma 7 play an important part.

- $[\![P]\!] \xrightarrow{a(\langle Z\rangle(Z\langle b\rangle))} T$. By Lemma 7, $P \xrightarrow{a(b)} P'$ and $T \sim_{ct} [\![P']\!]$. Because $P \approx_g Q$, we know that $Q \xrightarrow{a(b)} Q'$ $\approx_g P'$ and thus $[\![P']\!]\,\mathscr{R}\,[\![Q']\!]$. Then by Lemma 6, $[\![Q]\!] \xrightarrow{a(\langle Z\rangle(Z\langle b\rangle))} T'$ and $T' \sim_{ct} [\![Q']\!]$. So we have $T \sim_{ct} [\![P']\!]\,\mathscr{R}\,[\![Q']\!] \sim_{ct} T'$.
- $[\![P]\!] \xrightarrow{\overline{a}[(b)\langle Z\rangle(Z\langle b\rangle)]} T$. By Lemma 7, $P \xrightarrow{\overline{a}(b)} P'$ and $T \sim_{ct} [\![P']\!]$. Because $P \approx_g Q$, we know that $Q \xrightarrow{\overline{a}(b)} Q' \approx_g P'$ and thus $[\![P']\!]\,\mathscr{R}\,[\![Q']\!]$. Then by Lemma 6, $[\![Q]\!] \xrightarrow{(b)\overline{a}[\langle Z\rangle(Z\langle b\rangle)]} T'$ and $T' \sim_{ct} [\![Q']\!]$. Consider the following pair

  $$(b)(T\,|\,E[A]) \quad , \quad (b)(T'\,|\,E[A])$$

  in which $b \notin \text{fn}(E[X])$ and $A \stackrel{\text{def}}{=} \langle Z\rangle(Z\langle b\rangle)$. So

  $$(b)(T\,|\,E[A]) \sim_{ct} (b)([\![P']\!]\,|\,E[A]) \quad , \quad (b)([\![Q']\!]\,|\,E[A]) \sim_{ct} (b)(T'\,|\,E[A])$$

  By setting a context $C \stackrel{\text{def}}{=} (b)([\cdot]\,|\,E[A])$, we have the following pair in which $[\![P']\!]\,\mathscr{R}\,[\![Q']\!]$.

  $$C[[\![P']\!]] \quad , \quad C[[\![Q']\!]]$$

  This suffices to close this case in terms of the up-to context requirement.

- $[\![P]\!] \xrightarrow{\overline{a}[\langle Z\rangle(Z\langle b\rangle)]} T$. This case is similar to the last case.
- $[\![P]\!] \xRightarrow{\tau} T$. By Lemma 7, $P \xRightarrow{\tau} P'$ and $T \sim_{ct} [\![P']\!]$. From $P \approx_g Q$, we know $Q \Longrightarrow Q' \approx_g P'$ and thus $[\![P']\!] \mathscr{R} [\![Q']\!]$. Then by Lemma 6, $[\![Q]\!] \Longrightarrow T'$ and $T' \sim_{ct} [\![Q']\!]$. So we have $T \sim_{ct} [\![P']\!] \mathscr{R} [\![Q']\!] \sim_{ct} T'$.

□

3.4 Completeness

The completeness of the encoding is stated in the lemma below. We note that completeness is true even if we do not constrain the domain to be the image of the encoded π processes.

Lemma 9. *Suppose P is a π process. Then $[\![P]\!] \approx_{ct} [\![Q]\!]$ implies $P \approx_g Q$.*

Proof. We show that $\mathscr{R} \stackrel{\text{def}}{=} \{(P,Q) \mid [\![P]\!] \approx_{ct} [\![Q]\!]\} \cup \approx_g$ is a local bisimulation. Suppose $P \mathscr{R} Q$. There are several cases.

- $P \xRightarrow{a(b)} P'$. By Lemma 6, $[\![P]\!] \xRightarrow{a(\langle Z\rangle(Z\langle b\rangle))} T$ and $T \sim_{ct} [\![P']\!]$. Because $[\![P]\!] \approx_{ct} [\![Q]\!]$, we know that $[\![Q]\!] \xRightarrow{a(\langle Z\rangle(Z\langle b\rangle))} T' \approx_{ct} T$. By Lemma 7, $Q \xRightarrow{a(b)} Q'$ and $T' \sim_{ct} [\![Q']\!]$. Thus we have $[\![P']\!] \sim_{ct} T \approx_{ct} T' \sim_{ct} [\![Q']\!]$, so $P' \mathscr{R} Q'$, which fulfills this case.
- $P \xRightarrow{\overline{a}b} P'$. By Lemma 6, $[\![P]\!] \xRightarrow{\overline{a}[\langle Z\rangle(Z\langle b\rangle)]} T$ and $T \sim_{ct} [\![P']\!]$. Since $[\![P]\!] \approx_{ct} [\![Q]\!]$, we know that $[\![P]\!]$ must be able to be matched by $[\![Q]\!] \xRightarrow{\overline{a}[\langle Z\rangle(Z\langle b\rangle)]} T'$, because $[\![Q]\!]$ can only output such shape of processes, and if the matching is, e.g., $[\![Q]\!] \xRightarrow{\overline{a}[\langle Z\rangle(Z\langle c\rangle)]} T''$ then a context can be designed to distinguish between $[\![P]\!]$ and $[\![Q]\!]$. So for every $E[X]$, we have $T \mid E[\langle Z\rangle(Z\langle b\rangle)] \approx_{ct} T' \mid E[\langle Z\rangle(Z\langle b\rangle)]$. By Lemma 7, $Q \xRightarrow{\overline{a}b} Q'$ and $T' \sim_{ct} [\![Q']\!]$. So we know

$$[\![P']\!] \mid E[\langle Z\rangle(Z\langle b\rangle)) \approx_{ct} [\![Q']\!] \mid E[\langle Z\rangle(Z\langle b\rangle)] \tag{1}$$

  We want to show

$$P' \mathscr{R} Q' \quad \text{that is,} \quad [\![P']\!] \approx_{ct} [\![Q']\!] \tag{2}$$

  By setting E to be 0 in (1), we obtain (2), and thus close this case.
- $P \xRightarrow{\overline{a}(b)} P'$. By Lemma 6, $[\![P]\!] \xRightarrow{(b)\overline{a}[\langle Z\rangle(Z\langle b\rangle)]} T$ and $T \sim_{ct} [\![P']\!]$. Since $[\![P]\!] \approx_{ct} [\![Q]\!]$, we know that $[\![P]\!]$ must be able to be matched by $[\![Q]\!] \xRightarrow{(b)\overline{a}[\langle Z\rangle(Z\langle b\rangle)]} T'$ (apply α-conversion if needed). This is because $[\![Q]\!]$ can only emit such form of processes, and moreover if the matching does not have a bound name (e.g., $[\![Q]\!] \xRightarrow{\overline{a}[\langle Z\rangle(Z\langle c\rangle)]} T''$) then one can design a context to distinguish $[\![P]\!]$ and $[\![Q]\!]$. So for every $E[X]$ s.t. $b \notin \text{fn}(E)$, we have $(b)(T \mid E[\langle Z\rangle(Z\langle b\rangle)]) \approx_{ct} (b)(T' \mid E[\langle Z\rangle(Z\langle b\rangle)])$. By Lemma 7, $Q \xRightarrow{\overline{a}(b)} Q'$ and $T' \sim_{ct} [\![Q']\!]$. So we know

$$(b)([\![P']\!] \mid E[\langle Z\rangle(Z\langle b\rangle)]) \approx_{ct} (b)([\![Q']\!] \mid E[\langle Z\rangle(Z\langle b\rangle)]) \tag{3}$$

  In terms of local bisimulation [5, 25], for every π process R, we need to show

$$(b)(P' \mid R) \mathscr{R} (b)(Q' \mid R) \quad \text{i.e.,} \quad (b)([\![P']\!] \mid [\![R]\!]) \approx_{ct} (b)([\![Q']\!] \mid [\![R]\!]) \tag{4}$$

  Comparing equations (3) and (4), one can see that the different part is $E[\langle Z\rangle(Z\langle b\rangle)]$ and $[\![R]\!]$. Since the inverse of the encoding is a surjection, if all possible forms of E is itinerated, $[\![R]\!]$ must be hit somewhere (i.e., some choice of E makes $E[\langle Z\rangle(Z\langle b\rangle)]$ and $[\![R]\!]$ equal). Therefore we infer that (4) is true and thus complete this case.

- $P \stackrel{\tau}{\Longrightarrow} P'$. By Lemma 6, $[\![P]\!] \stackrel{\tau}{\Longrightarrow} T$ and $T \sim_{ct} [\![P']\!]$. Because $[\![P]\!] \approx_{ct} [\![Q]\!]$, we know $[\![Q]\!] \Longrightarrow T' \approx_{ct} T$. Then by Lemma 7, $Q \Longrightarrow Q'$ and $T' \sim_{ct} [\![Q']\!]$. So we have $P' \mathscr{R} Q'$ because $[\![P']\!] \sim_{ct} T \approx_{ct} T' \sim_{ct} [\![Q']\!]$.

□

4 Normal bisimulation for $\Pi^{D,d}$

In this section, we show that context bisimulation in $\Pi^{D,d}$ can be characterized by the much simpler normal bisimulation.

The factorization theorem

Below is the factorization theorem in presence of parameterization on names (and on processes as well). We recall that $\equiv$ is the structural congruence. As explained in Section 1, the upshot of establishing the factorization theorem is to find the right small processes so-called triggers. Here we have three kinds of triggers, to tackle different kinds of parameterizations. In particular, we stipulate that the triggers are as follows: $Tr^d_m \stackrel{\text{def}}{=} \langle z\rangle\overline{m}[\langle Y\rangle(Y\langle z\rangle)]$, $Tr^D_m \stackrel{\text{def}}{=} \langle Z\rangle\overline{m}Z$, and $Tr_m \stackrel{\text{def}}{=} \overline{m}$. These triggers are of somewhat a similar flavor but quite different in shape, with the aim at factorizing out respectively a name abstraction, a process abstraction and a non-abstraction process in certain context. The first trigger, i.e., Tr^d_m, is the main contribution of this work, whereas the other two are inherited from [26] and [19] respectively.

Theorem 10 (Factorization)**.** *Given $E[X]$ of $\Pi^{D,d}$, it holds for every A, fresh m (i.e., $m \notin fn(E,A)$) that*

(1) if $E[X]$ is not an abstraction, then

- *(i) if A is not an abstraction, then $E[A] \approx_{ct} (m)(E[Tr_m] \mid !m.A)$;*
- *(ii) if A is an abstraction on process, then $E[A] \approx_{ct} (m)(E[Tr^D_m] \mid !m(Z).A\langle Z\rangle)$;*
- *(iii) if A is an abstraction on name, then $E[A] \approx_{ct} (m)(E[Tr^d_m] \mid !m(Z).Z\langle A\rangle)$.*

(2) else if $E[X]$ is an abstraction, i.e., $E[X] \equiv \widetilde{\langle U\rangle}E'$ for some non-abstraction E' (here $\widetilde{\langle U\rangle}$ denotes the abstractions prefixing E'), then

- *(i) if A is not an abstraction, then $E[A] \approx_{ct} \widetilde{\langle U\rangle}((m)(E'[Tr_m] \mid !m.A))$;*
- *(ii) if A is an abstraction on process, then $E[A] \approx_{ct} \widetilde{\langle U\rangle}((m)(E'[Tr^D_m] \mid !m(Z).A\langle Z\rangle))$;*
- *(iii) if A is an abstraction on name, then $E[A] \approx_{ct} \widetilde{\langle U\rangle}((m)(E'[Tr^d_m] \mid !m(Z).Z\langle A\rangle))$.*

In Theorem 10, the clause (i) of (1) and (2) is actually Sangiorgi's seminal work [19]. The clause (ii) of (1) and (2) is analyzed in [26]. The clause (iii) of (1) and (2), which depicts the factorization for abstraction on names, can be discussed through a technical routine almost the same as (ii). With regard to more details we refer the reader to [19, 22, 26].

The method of *trigger* (including the technical approach) is well-developed in the field, due to the fundamental framework by Sangiorgi [22]. So the key to establishing factorization for processes allowing abstraction on names is the *trigger*, which is not known for a long time in contrast to the cases of abstraction on processes and that without abstractions. Once a right trigger is found, the rest of discussion is then almost standard. Below we give an example of the factorization concerning abstraction on names.

Example The basic idea of factorization concerning abstraction on names can be illustrated in the following example in which m is fresh (i.e., not in $A\langle d\rangle$).

$$A\langle d\rangle \quad \approx_{ct} \quad (m)(\ (\langle z\rangle\overline{m}[\langle Y\rangle(Y\langle z\rangle)])\langle d\rangle \mid m(Z).Z\langle A\rangle\ ) \ \equiv\ (m)(\ \overline{m}[\langle Y\rangle(Y\langle d\rangle)] \mid m(Z).Z\langle A\rangle\ )$$

For example, if A is $\langle x\rangle\bar{x}b$, then $A\langle d\rangle \equiv \bar{d}b$, and

$$A\langle d\rangle \;\approx_{ct}\; (m)(\,\overline{m}[\langle Y\rangle(Y\langle d\rangle)] \mid m(Z).Z\langle A\rangle\,) \;\approx_{ct}\; (m)(\,(\langle Y\rangle(Y\langle d\rangle))\langle A\rangle\,) \equiv A\langle d\rangle \equiv \bar{d}b$$

Normal bisimulation for $\Pi^{D,d}$

Below is the definition of normal bisimulation whose clauses are designed with regard to the factorization theorem. We recall that $Tr_m \stackrel{\text{def}}{=} \overline{m}$, $Tr_m^D \stackrel{\text{def}}{=} \langle Z\rangle\overline{m}Z$, and $Tr_m^d \stackrel{\text{def}}{=} \langle z\rangle\overline{m}[\langle Y\rangle(Y\langle z\rangle)]$.

Definition 11. A symmetric binary relation $\mathscr{R}$ on closed processes of $\Pi^{D,d}$ is a normal bisimulation, if whenever $P\,\mathscr{R}\,Q$ the following properties hold:

1. If $P \xrightarrow{a(Tr_m)} P'$ (m is fresh w.r.t. P and Q), then $Q \xRightarrow{a(Tr_m)} Q'$ for some Q' s.t. $P'\,\mathscr{R}\,Q'$;
2. If $P \xrightarrow{a(Tr_m^D)} P'$ (m is fresh w.r.t. P and Q), then $Q \xRightarrow{a(Tr_m^D)} Q'$ for some Q' s.t. $P'\,\mathscr{R}\,Q'$;
3. If $P \xrightarrow{a(Tr_m^d)} P'$ (m is fresh w.r.t. P and Q), then $Q \xRightarrow{a(Tr_m^d)} Q'$ for some Q' s.t. $P'\,\mathscr{R}\,Q'$;
4. If $P \xrightarrow{(\widetilde{c})\bar{a}A} P'$ and A is not an abstraction, then $Q \xRightarrow{(\widetilde{d})\bar{a}B} Q'$ for some $\widetilde{d}, Q'$ and B that is not an abstraction, and it holds that (m is fresh) $(\widetilde{c})(P' \mid !m.A)\;\mathscr{R}\;(\widetilde{d})(Q' \mid !m.B)$.
5. If $P \xrightarrow{(\widetilde{c})\bar{a}A} P'$ and A is an abstraction on process, then $Q \xRightarrow{(\widetilde{d})\bar{a}B} Q'$ for some $\widetilde{d}, Q'$ and B that is an abstraction on process, and it holds that (m is fresh) $(\widetilde{c})(P' \mid !m(Z).A\langle Z\rangle)\;\mathscr{R}\;(\widetilde{d})(Q' \mid !m(Z).B\langle Z\rangle)$.
6. If $P \xrightarrow{(\widetilde{c})\bar{a}A} P'$ and A is an abstraction on name, then $Q \xRightarrow{(\widetilde{d})\bar{a}B} Q'$ for some $\widetilde{d}, Q'$ and B that is an abstraction on name, and it holds that (m is fresh) $(\widetilde{c})(P' \mid !m(Z).Z\langle A\rangle)\;\mathscr{R}\;(\widetilde{d})(Q' \mid !m(Z).Z\langle B\rangle)$.
7. If $P \xrightarrow{\tau} P'$, then $Q \Longrightarrow Q'$ for some Q' s.t. $P'\,\mathscr{R}\,Q'$;

Process P is normal bisimilar to Q, written $P \approx_{nr} Q$, if $P\,\mathscr{R}\,Q$ for some normal bisimulation $\mathscr{R}$. Relation $\approx_{nr}$ is called normal bisimilarity, and is a congruence (see [19] for a reference). The strong version of $\approx_{nr}$ is denoted by $\sim_{nr}$.

Coincidence between normal bisimilarity and context bisimilarity in $\Pi^{D,d}$

Now we have the following theorem. The detailed proof is referred to [27].

Theorem 12. *In $\Pi^{D,d}$, normal bisimilarity coincides with context bisimilarity; that is, $\approx_{nr} = \approx_{ct}$.*

5 Conclusion

In this paper, we have exhibited a new encoding of name-passing in the higher-order paradigm that allows parameterization, and a normal bisimulation in that setting as well. In the former, we demonstrate the conformance of the encoding to the well-established criteria in the literature. In the latter, we prove the coincidence between normal and context bisimulation by pinpointing how to factorize an abstraction on some name. The encoding of this work is inspired by the one proposed by Alan Schmitt during the communication concerning another work. That encoding, as given below, somewhat swaps the roles of input and output and treats $a(x).P$ somehow as $a.\langle x\rangle P$ (like those calculi admitting abstractions and concretions [19]).

$$\begin{array}{rcl} [\![a(x).P]\!] & \stackrel{\text{def}}{=} & \bar{a}[\langle x\rangle[\![P]\!]] \\ [\![\bar{a}b.Q]\!] & \stackrel{\text{def}}{=} & a(Y).(Y\langle b\rangle \mid [\![Q]\!]) \end{array}$$

From the angle of achieving first-order interaction, the encoding strategy above is truly interesting. However, it appears not to satisfy some usual operational correspondence (say, in [8] or [12]), and full abstraction is not quite clear. Based on the results in this paper, it is tempting to expect that this encoding have some (nearly) same properties, and this is worthwhile for more investigation.

The results of this paper can be dedicated to facilitate further study on the expressiveness of higher-order processes. The following questions, among others, are still open: whether π can be encoded in a higher-order setting only allowing parameterization on processes; whether there is a better encoding of π than the one in [28], using higher-order processes only capable of parameterization on names; whether Π^d afford a normal-like characterization of context bisimulation.

Acknowledgements We thank the anonymous referees for their useful comments on this article.

References

[1] F. Bonchi, D. Petrisan, D. Pous & J. Rot (2015): *Lax Bialgebras and Up-To Techniques for Weak Bisimulations*. In: *Proceedings of the 26th International Conference on Concurrency Theory (CONCUR 2015)*, *Leibniz International Proceedings in Informatics (LIPICS)* 42, pp. 240–253, doi:10.4230/LIPIcs.CONCUR.2015.240.

[2] M. Bundgaard, T. Hildebrandt & J. C. Godskesen (2006): *A CPS Encoding of Name-passing in Higher-order Mobile Embedded Resources*. *Theoretical Computer Science* 356(3), pp. 422–439, doi:10.1016/j.tcs.2006.02.006.

[3] U. H. Engberg & M. Nielsen (1986): *A Calculus of Communicating Systems with Label Passing*. Technical Report DAIMI PB-208, Computer Science Department, University of Aarhus. Available at `http://www.daimi.au.dk/PB/208/`.

[4] U. H. Engberg & M. Nielsen (2000): *A Calculus of Communicating Systems with Label Passing - Ten Years After*. In: *Proof, Language, and Interaction: Essays in Honour of Robin Milner*, MIT Press Cambridge, pp. 599–622.

[5] Yuxi Fu (2005): *On Quasi Open Bisimulation*. *Theoretical Computer Science* 338(1-3), pp. 96–126, doi:10.1016/j.tcs.2004.10.041.

[6] Yuxi Fu (2015): *Theory of interaction*. *Theoretical Computer Science*, doi:10.1016/j.tcs.2015.07.043.

[7] Yuxi Fu & Hao Lu (2010): *On the Expressiveness of Interaction*. *Theoretical Computer Science* 411, pp. 1387–1451, doi:10.1016/j.tcs.2009.11.011.

[8] D. Gorla (2008): *Towards a Unified Approach to Encodability and Separation Results for Process Calculi*. In: *Proceedings of the 19th International Conference on Concurrency Theory (CONCUR 2008)*, *LNCS* 5201, Springer Verlag, pp. 492–507, doi:10.1007/978-3-540-85361-9_38.

[9] D. Gorla & U. Nestmann (2016): *Full Abstraction for Expressiveness: History, Myths and Facts*. *Mathematical Structures in Computer Scinece* 26, pp. 639–654, doi:10.1017/S0960129514000279.

[10] Daniele Gorla (2009): *On the Relative Expressive Power of Calculi for Mobility*. *Electronic Notes in Theoretical Computer Science* 249, pp. 269–286, doi:10.1016/j.entcs.2009.07.094.

[11] D. Kouzapas, J. A. Pérez & Nobuko Yoshida (2016): *On the Relative Expressiveness of Higher-Order Session Processes*. In: *Proceedings of the 25th European Symposium on Programming (ESOP 2016)*, LNCS, pp. 446–475, doi:10.1007/978-3-662-49498-1_18.

[12] I. Lanese, J. A. Pérez, D. Sangiorgi & A. Schmitt (2010): *On the Expressiveness of Polyadic and Synchronous Communication in Higher-Order Process Calculi*. In: *Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP 2010)*, LNCS, Springer Verlag, pp. 442–453, doi:10.1007/978-3-642-14162-1_37.

[13] I. Lanese, J.A. Pérez, D. Sangiorgi & A. Schmitt (2008): *On the Expressiveness and Decidability of Higher-Order Process Calculi*. In: *Proceedings of the 23rd Annual IEEE Symposium on Logic in Computer Science (LICS 2008)*, IEEE Computer Society, pp. 145–155, doi:10.1109/LICS.2008.8. Journal version in [**?**].

[14] S. Lenglet, A. Schmitt & J.-B. Stefani (2009): *Normal Bisimulations in Calculi with Passivation*. In: *Proceedings of the 12th International Conference on Foundations of Software Science and Computational Structures (FOSSACS 2009)*, *LNCS* 5504, Springer Verlag, pp. 257–271, doi:10.1007/978-3-642-00596-1_19.

[15] S. Lenglet, A. Schmitt & J.-B. Stefani (2011): *Characterizing Contextual Equivalence in Calculi with Passivation*. *Information and Computation* 209, pp. 1390–1433, doi:10.1016/j.ic.2011.08.002.

[16] R. Milner (1989): *Communication and Concurrency*. Prentice Hall.

[17] R. Milner, J. Parrow & D. Walker (1992): *A Calculus of Mobile Processes (Parts I and II)*. *Information and Computation* 100(1), pp. 1–77, doi:10.1016/0890-5401(92)90008-4, 10.1016/0890-5401(92)90009-5.

[18] J. Parrow (2016): *General Conditions for Full Abstraction*. *Mathematical Structures in Computer Science* 26, pp. 655–657, doi:10.1017/s0960129514000280.

[19] D. Sangiorgi (1992): *Expressing Mobility in Process Algebras: First-order and Higher-order Paradigms*. Phd thesis, University of Edinburgh.

[20] D. Sangiorgi (1996): *Bisimulation for Higher-order Process Calculi*. *Information and Computation* 131(2), pp. 141–178, doi:10.1006/inco.1996.0096.

[21] D. Sangiorgi (1998): *On the Bisimulation Proof Method*. *Mathematical Structures in Computer Science* 8(6), pp. 447–479, doi:10.1017/S0960129598002527.

[22] D. Sangiorgi & D. Walker (2001): *The Pi-calculus: a Theory of Mobile Processes*. Cambridge Universtity Press.

[23] B. Thomsen (1990): *Calculi for Higher Order Communicating Systems*. Phd thesis, Department of Computing, Imperial College.

[24] B. Thomsen (1993): *Plain CHOCS, a Second Generation Calculus for Higher-Order Processes*. *Acta Informatica* 30(1), pp. 1–59, doi:10.1007/BF01200262.

[25] Xian Xu (2012): *Distinguishing and Relating Higher-order and First-order Processes by Expressiveness*. *Acta Informatica* 49(7-8), pp. 445–484, doi:10.1007/s00236-012-0168-9.

[26] Xian Xu (2013): *On Context Bisimulation for Parameterized Higher-order Processes*. In: *Proceedings of the 6th Interaction and Concurrency Experience (ICE 2013)*, *EPTCS* 131, pp. 37–51, doi:10.4204/EPTCS.131.5.

[27] Xian Xu (2016): *Higher-order Processes with Parameterization over Names and Processes (with appendices)*. Available at `http://basics.sjtu.edu.cn/~xuxian/express2016withappendices.pdf`.

[28] Xian Xu, Qiang Yin & Huan Long (2015): *On the Computation Power of Name Parameterization in Higher-order Processes*. In: *Proceedings of 8th Interaction and Concurrency Experience (ICE 2015)*, *EPTCS* 189, pp. 114–127, doi:10.4204/EPTCS.189.10.

Self-Similarity Breeds Resilience

Sanjiva Prasad
IIT Delhi
sanjiva@cse.iitd.ac.in

Lenore D. Zuck *
University of Illinois at Chicago
lenore@cs.uic.edu

Self-similarity is the property of a system being similar to a part of itself. We posit that a special class of behaviourally self-similar systems exhibits a degree of resilience to adversarial behaviour. We formalise the notions of system, adversary and resilience in operational terms, based on transition systems and observations. While the general problem of proving systems to be behaviourally self-similar is undecidable, we show, by casting them in the framework of well-structured transition systems, that there is an interesting class of systems for which the problem is decidable. We illustrate our prescriptive framework for resilience with some small examples, e.g., systems robust to failures in a fail-stop model, and those avoiding side-channel attacks.

Keywords: Behavioural self-similarity, resilience, adversary, context, barbs, bisimulation, well-structured transition systems, fault-tolerance.

1 Introduction

Building systems that are resilient to actions of adversarial environments is an important software engineering concern. In this paper, we propose a class of systems whose resilience arises from a notion of *self-similarity*. An object is said to be "structurally self-similar" if it is similar to a proper part of itself. An important quality of the class of self-similar structures is that they are *scale-invariant*. In analogy, we consider a class of systems that are *behaviourally self-similar* — the behaviour of the system as a whole is "equivalent" to that of a part of the system — and develop a framework for showing how systems in this class are *resilient to adversarial actions*. The intuition behind our thesis is that if a part of a system is sufficient to exhibit the behaviour of the system as a whole, then the *rest* of the system provides *redundancy*, which in turn may provide resilience against a hostile environment. The notion of resilience is with respect to that of an *adversary*, a general concept pervading computing science, i.e., any way of choosing inputs or an environment that can thwart a program from achieving its desired behaviour.

A trivial example of a behaviourally self-similar system is a constant signal. Its behaviour during *any interval* of time is equivalent to its behaviour during an initial (arbitrarily small) interval, which is repeated *ad infinitum*. The signal can be considered to be resilient to an adversary that determines *when* to sample the output value, in that it is able to map the adversary's sampling interval to a more convenient input (the initial interval) for which its behaviour has been defined. Such "delay-tolerance" may also be seen in other time-independent functions.

The notion of behavioural self-similarity finds common currency in formal languages, in concurrency theory, as well as in programming. In formal languages, we have a ready example of a construction that supports behavioural self-similarity, namely the Kleene star, as $(e^*)^* \approx e^*$. Note that the construct $(\ )^*$ is semantically idempotent, a property that is often associated with fault-tolerance. The replication operation in process calculi such as the π-calculus [15] is another example of self-similarity: $!p \approx !p \| !p$, where $\approx$ denotes (behavioural/semantic) equivalence. It is also idempotent since $!!p \approx !p$, for any process

*The second author was supported in part by NSF grant CNS-1228697

D. Gebler and K. Peters (Eds.): Combined Workshop on Expressiveness in Concurrency and Structural Operational Semantics (EXPRESS/SOS 2016).
EPTCS 222, 2016, pp. 30–44, doi:10.4204/EPTCS.222.3

p. Indeed, in any recursive program scheme, we can find a syntactic part that behaves in a manner similar to the entire system. Consider, e.g., a recursive equation $X \approx F[X]$, for some non-trivial context $F[\,]$. By construction, X is *a fortiori* operationally equivalent to the expression $F[X]$. The semantics attached to such recursive equations involves finding an appropriate fixed point, usually the least fixed point, preferably by a (finite) iterative process. Observe that placing any solution to this equation in the context $F[\,]$ is a (behaviourally) idempotent operation[1].

From these examples, a naïve idea arises linking system structure, behavioural self-similarity and resilience: If, assuming no adversarial action, a part q of a system p can behave as the whole system does, then this part can be considered to provide the core functionality of the system; the rest of the system (the "context" $C[\,]$, where $p \equiv C[q]$) serves to neutralise adversarial actions or transform the interactions of the adversarial environment A with the system into a form which this "core" q can digest, and thereby make the system behave as though adversarial action by A were absent.

Resilience. We say that a system p is *resilient to an adversary* A if its observable behaviour in the presence of adversarial action is equivalent to its behaviour in the absence of the adversary: $p \circ A \approx p$, where $\circ$ represents coupling the system p with the adversary A[2]. A somewhat similar formulation has been explored earlier by Liu, Joseph, Peled, Janowski and others [14, 17, 11], but we believe our formulation is more natural (discussed in §1.1). Now, if p can be expressed by $C[q]$ as above, and in the absence of an adversary, $C[q] \approx q$, we have, by transitivity, the desired resilience to the adversary A arising from self-similarity. Resilience in this sense should not be equated with a notion of correctness; a system may be resilient even if it is not correct with respect to a given specification. Note that if $\approx$ is a congruence, $C[C[q]] \approx C[q] \approx q$, showing the expected idempotence of $C[\,]$ in countering adversary A.

Adversary model. An adversary can be viewed as a way of forcing the program to face an unfavourable environment. The class of adversaries may be expressed in any of a variety of ways: as processes in a language, as automata or transition systems, as logical constraints on behaviour, etc. All that our framework requires is that the program coupled with the adversary is a transition system on which a reasonable notion of observational equality can be defined. We include in the class of adversaries a completely benign adversary, denoted 1_A, who behaves as if there were no adversary present when coupled with any system. If the adversarial model is specified as a transition system, we require that it be *well-structured*, with $1_A \preceq A$ for any A in the class of adversaries.

We identify here some constraints on what an adversary can and cannot do. (i) Adversaries may act in ways completely unrestricted by the system. (ii) Adversarial moves, except for announcement of error, are not directly observable. This is justifiable in that most adversaries are sneaky, not bruiting their actions, until and unless they wish to announce that they have defeated the system, i.e., a denouement *err*. (iii) An adversary cannot directly prevent the system from making any *observable* move by removing an enabled action. (iv) An adversary can, however, interact with the system, and make joint moves. These interactions too may not be directly observable, but may cause the system to (eventually) produce different observable effects from its normal behaviour.

Structure of the paper. In the sequel, we develop this idea by formalising the notions of a system's behaviour, adversarial action, resilience and *a methodology for proving a system to be resilient*, by fac-

[1] Finding the minimal fixed point context helps us avoid "needless redundancy".

[2] Note that A need not be specified in the syntax of the language in which p is expressed, and that the notion of coupling of the adversary to the system may be more general than the usual notion of parallel composition.

toring a system into its "core" and fault-tolerant context. This is followed by a discussion of related work (§1.1). Formalisation (§2) of both system and adversarial behaviour is done in the very general setting of *transition systems*. We employ a process calculus notation for expressing processes, and the associated structural operational semantics helps in relating structure to behavioural self-similarity. In this paper, we use suitable notions of *observation* and *equivalence*, namely, those of *barbs* and bisimulation [16], since we consider systems *closed* when coupled with an adversary. While other behavioural equivalences may be considered, we chose bisimulation since it is the finest extensional notion of equivalence of observable behaviour. Proposition 1 expresses the soundness of our proposed methodology.

In general, proving bisimilarity of transition systems, and consequently showing a system to be behaviourally self-similar, is *undecidable*. However, we can show that this problem is decidable for an interesting class of systems (Proposition 2) by using the framework of *Well-Structured Transition Systems* (WSTS) [6]. The WSTS conditions required for establishing decidability seem to arise very *naturally* from the constraints placed on the context and adversary.

We then illustrate our prescriptive framework for resilience with small examples (§3), such as systems robust to failures in a fail-stop model, and defeating side-channel attacks. In the examples in this paper, which progress from finite to finite-control to infinite state systems, we do not propose any new mechanisms for building resilient systems. We deploy the familiar armoury of devices — redundancy, replication, retry, and repetition — for countering the arsenal at the adversary's disposal. However, our framework may be seen as providing a formal (methodological) justification of these constructions. While fault-resistance has earlier been shown using rigorous mathematical techniques, we believe that our use of the WSTS framework provides the basis for automated techniques for proving resilience, especially in the case of infinite-state systems. We illustrate our approach by conveying only the intuition for the different examples, and omit the tedious details of the proofs. In §4, we conclude with a discussion of our approach, its limitations, alternative frameworks for specifying and verifying resilience to adversarial behaviour, as well as some directions for future work.

Methodology. Our proposed methodology is:

1. We identify a class of adversaries, with a *least* adversary 1_A. Adversarial moves are not normally observable, except perhaps for a final barb *err*.
2. We decompose the system process p into a core q which provides the basic functionality and a (fault-resilient) context $C[\,]$. Thus $p \equiv C[q]$. In general, the context may be multi-holed. The context $C[\,]$ should not alter the core functionality of q. In particular, it should satisfy the following conditions:
   (a) $C[\,]$ should permit q to make any of its possible moves, i.e., $q \longrightarrow q'$ implies $C[q] \longrightarrow C[q']$ and $q \Downarrow o$ implies $C[q] \Downarrow o$;
   (b) If $C[\,]$ and q jointly make a move, then all of q's possible barbs are preserved, i.e., if $C[q] \Downarrow o$ and $C[q] \longrightarrow C'[q']$ then $C'[q'] \Downarrow o$;
   (c) The context $C[\,]$ (and its derivatives) should by itself contribute no observable barbs, i.e., $C[\,] \not\Downarrow o$ for any o;
   (d) No transition arising purely due to $C[\,]$ disables the execution of the process q, i.e., if $C[\,] \longrightarrow C'[\,]$, then (i) $q \Downarrow o$ implies $C'[q] \Downarrow o$ (since $C[q] \Downarrow o$), and (ii) $q \longrightarrow q'$ implies $C'[q] \longrightarrow C'[q']$.
3. We then specify the coupling $p \circ A$ of a process and an adversary as a transition system. Formally, we will require that this transition system be a WSTS. In particular, this composite transition system should exhibit the *upward simulation property* (defined in §2).

4. To show resilience of p with respect to the adversary A, we show that $q \circ 1_A \approx p \circ A$. This problem is decidable for WSTSs with *certain additional properties*.

We dub the conditions on the context and the adversary listed above the *self-similarity constraints*.

Beyond finite-state systems. While our examples in this paper are small, our framework is not confined to dealing with finite (or finite-state) systems, for which it may be easy to show the required bisimilarity. Accordingly, we explore systems that provide sufficient structural properties to ensure that *bisimilarity is decidable*. We find that Well-Structured Transition Systems (WSTSs) [6] provide a framework in which we can formulate and *verify* resilience by virtue of self-similarity.

Consider first a *simple version* of the framework: Structural inclusion of q in $C[q]$ for a context satisfying the self-similarity conditions is an obvious candidate when defining an ordering relation ($q \| r \preceq C[q] \| r$). A simple way to obtain the conditions on context $C[\,]$ mentioned above is to constrain the hole(s) $[\,]$ in context $C[\,]$ to appear only at "head" or "enabled" positions. This allows $C[q]$ to simulate q, and if $C[\,]$ has no observable actions, then every barb of q is a barb of $C[q]$ and vice versa.

Often the adversary itself can be formulated as a WSTS with a least element 1_A representing the absence of an adversary. The composite transition system is obtained from those of the system and the adversary, and a *pointwise combination* of the system and adversary orderings yields the desired ordering relation for the WSTS. We say that the composition with adversary A is *monotone* if $p \longrightarrow p'$ implies $p \circ A \longrightarrow p' \circ A$, and $p \Downarrow o$ implies $p \circ A \Downarrow o$. This is usually the case with parallel composition in most process calculi.

The self-similarity constraints on the context and adversary imply the following properties:

- (Upward simulation) $q \longrightarrow q'$ implies $C[q] \longrightarrow C[q']$. For monotone compositions with adversaries, this further implies $C[q] \circ A \longrightarrow C[q'] \circ A$.
- For an observable o, $q \Downarrow o$ if and only if $C[q] \Downarrow o$.
- if $A \longrightarrow A'$ then for any p: $p \circ A \longrightarrow p \circ A'$.

What remains is to place reasonable effectiveness constraints on the WSTSs in order to ensure that bisimilarity is decidable. We require that the states of the system and the class of adversaries are *recursive sets* and that the ordering $\preceq$ is decidable. We also assume that the transition systems are *finitely-branching*. In order to ensure decidability, we require that the transition systems satisfy a technical condition of having an *effective pred-basis*, and exhibit downward reflexive simulation (these definitions are recalled in §2).

1.1 Contributions and Related Work

We are unaware of any previous work where the notion of self-similarity has been wedded to that of behavioural equivalence to formulate a notion of a system being resilient to actions by an adversarial environment. Furthermore, we believe that the methodology we enunciate — construing a system as being constructed of a core behaviour and a context for handling the actions of a formally defined adversarial environment — is novel, as also casting them in the framework of Well-Structured Transition Systems [6]. The structural decomposition of the system into core and fault-absorbing context seems natural and dovetails nicely with the WSTS conditions. As a consequence, the bisimulation proofs become much easier. Moreover, the effectiveness conditions provide decidability, and thus in principle at least, support automated techniques for showing resilience that can work even for infinite-state systems. We also believe that our third example, which deals with a building block for converting communication

over a non-FIFO channel with omission failures to a FIFO-channel with omission failures, has not been presented earlier in its essential form.

The idea of formulating fault tolerance in terms of behavioural equivalence is not new [14, 17, 11]. The idea of a fault preorder, capturing the relative severity of faults, can be found in the works of Janowski, Krishnan and others [11, 12, 14]. Janowski, e.g., studies the problem of monotonicity of fault tolerance — a system tolerant of faults higher in the preorder should tolerate faults lower in the preorder, but finds that this requirement does not square well with the standard notion of bisimilarity. A similar observation is made by Krishnan [12], where he considers replicated systems to model systems with synchronous majority voting. Accordingly, a notion of bisimilarity parameterised by the fault model is proposed [11, 12]. In contrast, we believe that the notion of behaviour should be *uniform* and therefore formulate the notion of resilience to an adversary using a *standard* notion of equivalence, e.g., weak (asynchronous) barbed bisimulation [16, 2].

Another major difference with these approaches [10, 11] is that they formulate the faulty versions of the system by incorporating the anticipated faulty behaviour into definitions of the system. We see this is as unsatisfactory in that the adversarial behaviour has to be expressed concretely and within the syntax of the system (e.g., in the CCS formulation), thereby severely restricting the expressive power given to the adversary. It is also not a very satisfactory way of composing adversaries. Janowski uses the technique of incorporating the faulty version with the original system by providing a *redefinition of the original system* taking a non-deterministic choice of the two behaviours. This approach does not work well, e.g., with modelling a fail-stop model, because "revenant" processes become possible — a system which is supposed to have failed, rises Lazarus-like and exhibits some active behaviour. In contrast, our formulation uses a very general framework of transition systems, which may be specified and combined in any convenient manner. Indeed, the syntactic formulation used for describing the example systems is only a convenient way for specifying a transition system and the constructive nature of fault-resilient transformations.

There is also some similarity between our work and that of Liu, Joseph, Peled and others, in e.g., [14, 17], where they present frameworks in which fault-tolerance is expressed by transforming a process with respect to the specification of a fault model: q is a ψ-tolerant implementation of p if $p \approx T(q,\psi)$, for a transformation $T(\_,\_)$. The juxtaposition of the recovery algorithm is viewed as a transformation that makes a process fault-tolerant. The connections between these logic-based ideas and our preliminary operational formulation deserves further study.

2 The Framework

2.1 Transition Systems

A *transition system* $\mathscr{T} = \langle S, \longrightarrow \rangle$ consists of a set of states S and a transition relation $\longrightarrow \subseteq S \times S$. A finite *trace* σ with respect to $\mathscr{T}$ is a sequence of states $\langle s_0, s_1, \ldots, s_n \rangle$ such that $s_i \longrightarrow s_{i+1}$ for all $0 \leq i < n$. An infinite trace with respect to $\mathscr{T}$ can be considered to be a function $\sigma : \mathbb{N} \to S$ such that for all $i \in \mathbb{N}$: $\sigma(i) \longrightarrow \sigma(i+1)$. We write $\sigma_1\sigma_2$ to denote the concatenation of traces, which in the case of σ_1 being an infinite trace, results in σ_1.

The *successors* and *predecessors* of a state $s \in S$, are defined as $Succ(s) = \{s' \mid s \longrightarrow s'\}$ and $Pred(s) = \{s' \mid s' \longrightarrow s\}$ respectively. The notations $\longrightarrow^n$, $\longrightarrow^=$, $\longrightarrow^+$ and $\longrightarrow^*$ are used for the n-step iteration, reflexive closure, transitive closure and reflexive-transitive closure of the transition relation $\longrightarrow$. We use a similar notation for the n-step iterations, and closures of *Succ* and *Pred*. $\mathscr{T}$ is *finitely-branching* if $Succ(s)$ is finite for each s.

Assuming a notion of *observable actions*, we define a *barb* as an observable action that a process has the potential to perform, written $p \downarrow o$ (where o is observable) [16]. We will use "weak barbs" which depict the potential of a process to perform an observable action after making some "silent" transitions, i.e., $p \Downarrow o$ if for some p', $p \longrightarrow^* p'$ and $p' \downarrow o$.

A weak *barbed simulation* is a binary relation R on processes such that if $(p_1, p_2) \in R$, then (1) if $p_1 \Downarrow o$ then $p_2 \Downarrow o$; and (2) whenever $p_1 \longrightarrow^* p_1'$ then there exists p_2' such that $p_2 \longrightarrow^* p_2'$ and $(p_1', p_2') \in R$. R is a *weak barbed bisimulation* if both R and its symmetric inverse R^{-1} are weak barbed simulations. Processes p_1 and p_2 are weakly barbed bisimilar, written $p_1 \approx_b p_2$, if they are related by some weak barbed bisimulation. Weak barbed bisimulation is not fine enough to distinguish processes that differ after the first communication (barb), when they interact with other processes, so the equivalence relation usually desired is a refinement that is preserved under parallel composition.

$$p_1 \approx p_2 \textit{ if } \forall\, q :\ (p_1 \| q) \approx_b (p_2 \| q)$$

However, for closed systems, it is reasonable to use weak barbed bisimulation as the notion of equivalence.

Process notation. In our examples, we employ a process calculus notation akin to a value-passing CCS. Let a represent a channel, x a variable, and v a value drawn from some set of values. We assume without further specification that the language includes a syntactic category of expressions e, which contains in particular variables and integer-valued expressions. In our examples, expressions may also include tuples, and we assume a matching operation. Terms in the language of communicating processes, typically p, p_1, p_2 are specified inductively by the following abstract syntax:

$$p ::= 0 \mid \overline{a}e.p \mid ax.p_1 \mid p_1 \| p_2 \mid (a)p_1 \mid p_1 + p_2 \mid [e_1 = e_2]p \mid\ !p \mid P(e_1, \ldots e_n)$$

The process 0 is inert. The prefix $\overline{a}e$ stands for the output of the value of e on channel a, whereas $ax.p$ stands for input of a value over channel a with the value bound to x in the continuation p. $p_1 \| p_2$ represents parallel composition whereas $p_1 + p_2$ stands for non-deterministic choice between p_1 and p_2. The notation $(a)p_1$ describes the *restriction* operation on channels, i.e., channel a is local to scope p_1. Communication on restricted channels is *not* observable. $[e_1 = e_2]p$ is a conditional matching operation. $!p$ represents the replication of process p, yielding as many copies of p as desired, running in parallel. For convenience, we include parameterized (recursive) processes of the form $P(e_1, \ldots e_n)$.

In a distributed system, we associate processes with *locations*, written for instance as $\ell[\![p]\!]$, where ℓ is a location constant. A context $C[\,]$ is a process term with a hole $[\,]$ in the place of a process term. We may also have multi-hole contexts. We do not present here the formal rules for the operational semantics of this language. Indeed, these constructs can be encoded in a core asynchronous calculus with replication and choice restricted to *guarded processes*. We refer the reader to any standard presentation of such asynchronous calculi [9, 2] for the encodings and the operational semantics rules.

2.2 Well-structured Transition Systems

We now summarise some results about WSTSs [6].

A *quasi-order* or *pre-order* $\langle S, \preceq \rangle$ consists of a set S with any reflexive and transitive relation $\preceq \subseteq S \times S$. $\langle S, \preceq \rangle$ is *well-ordered* (henceforth a *wqo*) if for any *infinite* sequence $s_0, s_1, \ldots$ there exist indices i and j with $i < j$ such that $s_i \preceq s_j$. Consequently, a wqo has no infinite strictly decreasing sequence, nor any

infinite sequence of unrelated elements. It also follows that in a wqo, any infinite sequence $s_0, s_1, \ldots$ has an *infinite non-decreasing subsequence* $s_{i_0} \preceq s_{i_1} \preceq s_{i_2} \ldots$ where $i_0 < i_1 < i_2 \ldots$.
In quasi-order $\langle S, \preceq \rangle$, *an upward-closed set* is any set $I \subseteq S$ such that if $x \in I$ and $x \preceq y$, then $y \in I$. Let $\uparrow x = \{y \mid x \preceq y\}$, called the upward closed set induced by x. A *basis* for an upward closed set I is a set $I_b \subseteq I$ such that $I = \bigcup_{x \in I_b} (\uparrow x)$.
Higman's Lemma states that if $\langle S, \preceq \rangle$ is a wqo, then any upward-closed set I has a *finite basis*. The minimal elements of I form a basis, and these must be finite, since otherwise, they would include an infinite sequence of unrelated elements. Further, any chain of upward-closed sets $I_0 \subseteq I_1 \subseteq I_2 \subseteq \ldots$ stabilises, i.e., for some k, $I_j = I_k$ for every $j \geq k$.
A *Well-Structured Transition System* (WSTS) $\mathscr{T} = \langle S, \longrightarrow, \preceq \rangle$ consists of a transition system $\langle S, \longrightarrow \rangle$ equipped with a well-ordered quasi-order $\langle S, \preceq \rangle$ that satisfies *weak upward simulation*: For every s, s', and $t \in S$, if $s \longrightarrow s'$ and $s \preceq t$, then there exists $t' \in S$ such that $t \longrightarrow^* t'$ and $s' \preceq t'$.
A WSTS exhibits *downward reflexive simulation* if for each s, if $s \longrightarrow s'$ and $t \preceq s$, then there exists t' such that $t \longrightarrow^= t'$ and $t' \preceq s'$, i.e., either $t \equiv t'$ (0 steps) or $t \longrightarrow t'$ (1 step).
A WSTS $\mathscr{T} = \langle S, \longrightarrow, \preceq \rangle$ has an *effective pred-basis* [6] if there exists an algorithm that, given any state $s \in S$, computes *pb*(s), a finite basis of $\uparrow Pred(\uparrow s)$, i.e., minimal elements of the upward-closed set induced by the predecessors of states in the upward-closed set induced by s.
Backward reachability analysis involves computing $Pred^*(J)$ as the limit of a chain J_i, where $J_i \subseteq J_{i+1}$. If J is upward-closed, then this process converges, and $Pred^*(J)$ is upward-closed. If a WSTS $\mathscr{T} = \langle S, \longrightarrow, \preceq \rangle$ has an *effective pred-basis* and $\preceq$ is decidable, then if any upward-closed J is given via its finite basis, one can compute a finite basis of $Pred^*(J)$.
The *covering problem* is, given states s and t, to decide whether there exists a t' such that $s \longrightarrow^* t'$ and $t \preceq t'$. The covering problem is decidable for $\mathscr{T} = \langle S, \longrightarrow, \preceq \rangle$ with an *effective pred-basis* and decidable $\preceq$.
If $\mathscr{T} = \langle S, \longrightarrow, \preceq \rangle$ exhibits downward reflexive simulation, and if *Succ* is computable and $\preceq$ decidable, then one can compute for any s a finite basis of $\uparrow Succ^*(s)$.
The *sub-covering* problem is to decide, given s and t, whether there exists t' such that $s \longrightarrow^* t'$ and $t' \preceq t$. *Subcovering* is decidable for any WSTS which shows downward reflexive simulation, if *Succ* is computable and $\preceq$ decidable.

Putting the pieces together, we get the following method for proving whether a putatively fault-tolerant process $p \equiv C[q]$ is indeed resilient (or not) to an adversary A, where $C[\,]$ denotes the fault-digesting context and q its core functionality:

Proposition 1 *A process p is resilient to adversary A while providing behaviour q if p can be expressed as $C[q]$ such that $C[\,]$ and A satisfy the self-similarity constraints, and $C[q] \circ A \not\Downarrow err$.*

Proof: Consider $C[q] \circ A$. We wish to show $q \circ 1_A \approx C[q] \circ A$.

- *Upward Simulation:* The adversary does not directly restrict any action of the system. Since 1_A represents the non-existence of an adversary, $q \circ 1_A \longrightarrow t$ implies $t \equiv q' \circ 1_A$ for some q'. If $q \Downarrow o$, then clearly $q \circ 1_A \Downarrow o$ and also $C[q] \Downarrow o$. If composition with the adversary is monotone, then $C[q] \circ A \Downarrow o$. In general, this may not be the case (as will be seen in the second example of §3). If for a context $C[\,]$, upward simulation is not satisfied, then that context *fails to provide resilience* to the adversary A[3].

[3]The above-mentioned second example satisfies the WSTS condition only for certain contexts amongst contexts that satisfy the other self-similarity conditions 2(a)-2(d).

- *Downward reflexive simulation:* The adversary or its interaction with the system does not cause anything that the core cannot do. Since the context $C[\ ]$ and the adversary do not contribute observables (except the "denouement"), $C[q] \circ A \Downarrow o$ implies either $q \Downarrow o$ or that the process is *not* resilient to adversary A (if o is the barb *err*). (The first example in §3 involves an adversary that attempts to make observable the barb *err*.) Now consider the moves that $C[q] \circ A$ can make. These may be
  1. a move due to q: $C[q] \longrightarrow C[q']$. This is downward simulated by $q \longrightarrow q'$ (and hence $q \circ 1_A \longrightarrow q' \circ 1_A$).
  2. a move due to only $C[\ ]$: $C[q] \longrightarrow C'[q]$. This is downward simulated by q making no move. Moreover, $C[q] \Downarrow o$ iff $C'[q] \Downarrow o$ iff $q \Downarrow o$.
  3. a move due to A: $C[q] \circ A \longrightarrow C[q] \circ A'$. Since the adversary's moves are unobservable, this is downward simulated by $q \circ 1_A$ making no move, since $1_A \preceq_2 A'$.
  4. a move involving both $C[\ ]$ and q: $C[q] \longrightarrow C'[q']$: this is simulated by $q \longrightarrow q'$, and $C'[q'] \Downarrow o$ iff $q' \Downarrow o$, since $q' \preceq C'[q']$ and $C'[\ ]$ does not contribute any observables and does not inhibit any observables of q'.
  5. A move involving the adversary *and* the fault-handling context $C[\ ]$: $C[q] \circ A \longrightarrow C'[q] \circ A'$. Again, since this move is unobservable, this is downward simulated by $q \circ 1_A$ making no move, since $q \circ 1_A \preceq C'[q] \circ A'$ for $C'[\ ], A'$.

⊡

Thus, our prescriptive framework of crafting self-similar processes and formalising adversarial behaviour operationally yields the desired resilience. If the two systems are not barbed bisimilar, then the process is not fault tolerant, and a counter-example may be found. Another way of looking at the definition of resilience is that adversarial moves keep the composite system within the same behavioural equivalence class.

Dealing with infinite state systems. Since the state spaces may potentially be infinite, there may not be an effective method to *prove* that $C[q] \circ A$ and $q \circ 1_A$ are bisimilar. In order for this question to be decidable, we place the restrictions mentioned earlier, namely that the WSTS has an effective pred-basis, that the successor relation is effectively computable, and that the conditions on adversarial moves ensure reflexive downward simulation. The "self-similar subterm" orderings are clearly decidable.

Proposition 2 *The problem of deciding whether a process is resilient to an adversary is decidable if in addition to satisfying the self-similarity constraints, the coupled transition system has an effective pred-basis, and the successor relationship is effectively computable.*

Proof: Suppose $q \circ 1_A \longrightarrow t$. We need to show that there is a t' such that $C[q] \circ A \longrightarrow^* t'$ and $t \preceq t'$. This is an instance of the *covering problem* for $C[q] \circ A$ and t, which is decidable under the assumptions. *Proof technique:* Compute K_b, the finite basis of $Pred^*(\uparrow t)$, and check if $C[q] \circ A \in \uparrow K_b$.
Now suppose $C[q] \circ A \longrightarrow t$. We need to show that there is a t' such that $q \longrightarrow^* t'$ and $t' \preceq t$. This is an instance of the sub-covering problem for q and t, which is decidable when context/adversarial/joint moves are simulated downwards in 0 steps, and successors are effectively computable. *Proof technique:* Compute K_b, the finite basis for $Succ^*(q \circ 1_A)$. Check if $t \in \uparrow K_b$. ⊡

3 Examples

In this section we put our proposed methodology to test by applying it in situations that demand resilience of a system to an adversary. Our examples proceed from finite to finite-control to infinite state systems.

In each case, we apply the methodology of identifying the core computation that would have sufficed in the absence of an adversary. We then identify an adversary and show how by constructing an absorptive context satisfying the self-similarity conditions presented earlier, one can defeat the adversary. The wqo notion used is usually simple, though the later examples, involving resilience in distributed systems, motivate the need for more flexible notions than simple embedding of a process into a context. However, they retain the essential semantic requirement that the context preserves the ability of a process to perform its actions, and that the context neither contributes any observable actions directly, nor does it take away the observables of the core process. It can at best interact with other parts of the context and/or with the adversary.

3.1 White Noise to defeat Side-channel Attacks

Let c be a deterministic finite computation which generates an observable result m in n steps: $c \longrightarrow^n c' \downarrow m$. Normally, observers cannot see the number of steps taken by c. Let us now consider an adversary that can see in addition to this outcome, the "side property" of how many steps were taken to termination[4]. If it observes that c terminates within n steps, it flags *err*. A class of step-counting adversaries can be coded as $A(i,k)$ for $k \geq 0$, with the behaviour of the composite transition system $c \circ A(i,n)$ given by the following rules:

$$\frac{p \longrightarrow p'}{p \circ A(i,n) \longrightarrow p' \circ A(i+1,n)} \quad \frac{p \downarrow m}{p \circ A(i,n) \downarrow m} \quad \frac{p \downarrow m}{p \circ A(i,n) \downarrow err} \quad (i \leq n)$$

Note that the adversary does not suppress any observable of p, and makes no observable moves except signalling *err*. In the absence of an adversary monitoring the side-channel, we have $c \circ 1_A \Downarrow m$, but in the presence of such an adversary, $c \circ A(0,n) \Downarrow err$ as well. Thus if c_1 is a program behaviourally equivalent to c but which takes more than n steps to terminate, this adversary can distinguish between c and c_1 as $c_1 \circ A(0,n) \not\Downarrow err$.

We now justify the correctness of the method of interleaving a computation with an indeterminate number of NOPs to defeat such side-channel attacks. Let *WhiteNoise* be a computation that consumes cycles, but does *not* generate any observables, e.g., $WhiteNoise \longrightarrow WhiteNoise$. *WhiteNoise* does not suppress or alter any normal observables of computations running in parallel/interleaved with it. Consider the context $C[\,] = WhiteNoise \|_f [\,]$, and suppose $\|_f$ is a weakly fair (nondeterministic[5]) interleaved implementation of parallel composition that executes at least one step of *WhiteNoise*. Now note that while $C[c] \circ A(0,n) \Downarrow m$, it is no longer the case that $C[c] \circ A(0,n) \Downarrow err$. Thus $A(0,n)$ cannot distinguish between $C[c]$ and $C[c_1]$.

It is straightforward to show that $C[\,]$ and $A(0,n)$ satisfy the self-similarity conditions. We cast the composite system as a WSTS, using the "self-similar subterm" ordering on processes. In particular, we consider as minimal elements (which form a finite basis) all processes c' such that $c \longrightarrow^* c'$. The *effective pred-basis* is then easy to compute. It is therefore easy to prove that $C[c] \circ A(0,n) \approx c$. Note also that this context $C[\,]$ is idempotent; reiterating it, as in $C[C[c]]$, does not provide further security against this side-channel attack.

[4] Other side properties such as heat generated, or power consumed could also be monitored.

[5] If the interleaving performs a deterministic number of *Whitenoise* steps, then by a small extension to the adversary class, one can mount a side-channel attack.

3.2 Replicated server

The next example involves finite-control, and justifies the use of repetition to address multiplicity of requests and spatial replication to counter failure. Consider a one-time provider of a file v: $OTP = \overline{a}v$, where a is the channel on which v is sent. Similarly, a basic client is rendered as $BC_i = ax.\overline{d_i}x$, which receives some file x on the channel a, and delivers it to the client's application layer, written as $\overline{d_i}x$. The single-request client-provider system is written as $Sys_1 = (a)(BC_1 \| OTP)$. The only observable barb of Sys_1 is the unrestricted send action $\overline{d_1}v$.

If the file provider has to deal with more than one client, or if the client repeatedly requests the file, we need an ever-obliging *responsive* server, represented as a process that can *repeatedly* send file v on the channel a.

$$Server = \ell_1[\![!\ [\overline{a}v]]\!].$$

Typically, the server is located at some site ℓ_1, written in a distributed calculus (e.g., [18]) as $\ell_1[\![\ldots]\!]$, which may be different from the client's site. For simplicity we have the *Server* located at ℓ_1 repeatedly sending the file over a to whoever wishes to receive it[6]. In the distributed setting, note that $p \longrightarrow p'$ implies $\ell_1[\![p]\!] \longrightarrow \ell_1[\![p']\!]$, and $p \downarrow o$ implies $\ell_1[\![p]\!] \downarrow o$ only when site ℓ_1 is "up" (locations do not figure in the observable barbs). Observe that the context in which *OTP* is placed contributes no barbs, and has a hole in a position that is enabled. Consider the new system:

$$Sys_2 = (a)(BC_1 \| \ldots BC_n \| Server)$$

The observable barbs are $\overline{d_i}v$, for $i \in \{1,\ldots,n\}$. Since a is restricted, the send actions on a do not contribute any observable barb. This construction also handles the case when the clients repeatedly request a value, i.e., when $BC_i = !(ax.\overline{d_i}x)$.

Now consider an adversary A that can cause location ℓ_1 to fail, taking the server down permanently in the fail-stop model (see e.g., [3, 18]) after which no client can receive any file from the *Server*. Observe that the requirement that ℓ_1 needs to be "up" for a barb to be observable means that the coupling with the adversary is *not* monotone. The adversary can be modelled as a transition system with states that represent the set of locations that are "up" and the transition $\{\ell_1\} \longrightarrow \{\}$ to model the failure of ℓ_1. It is now possible to have trace suffixes in which $Sys_2 \circ A \not\Downarrow \overline{d_i}v$ whereas $Sys_2 \circ 1_A \Downarrow \overline{d_i}v$, for some value(s) of i. Thus, Sys_2 is not resilient to an adversary that can cause a single location to fail.

We build a server resilient to a *single node failure* by *replicating* the responsive file provision on two sites, ℓ_1 and ℓ_2. Fault tolerance is provided by using the two hole context

$$C_{rep}[\,] = \ell_1[\![![\,]_1]\!] \parallel \ell_2[\![![\,]_2]\!]$$

that places a process in two holes both at head positions, and satisfying all the requisite self-similarity conditions for the context, yielding a *replicated responsive server*:

$$RepServer = C_{rep}[OTP] \equiv \ell_1[\![!\overline{a}v]\!] \| \ell_2[\![!\overline{a}v]\!].$$

The server process located at ℓ_i can execute only if that location has not failed (fail-stop model of failure) — it provides the value v on channel a while its site is up. Let

$$Sys_3 = (a)(BC_1 \| \ldots BC_n \| RepServer)$$

[6]Typically this is coded (in a π-calculus) as a client sending a request to a server, sending a private channel over which it wishes to receive the file.

Client BC_i can read a value on channel a either from the server at ℓ_1 or ℓ_2, whichever is up; if both are up, it obtains the value from either one (the location of the server is not observable). So if either or both ℓ_i are up, then $\ldots BC_i \ldots \| \mathit{RepServer} \Downarrow \overline{d_i} v$.

The adversary A, which is able to cause *at most one node* to fail, can be modelled by a finite state machine, which *may* make a transition from a state in which both $\{\ell_1, \ell_2\}$ are up, to states where ℓ_1 (respectively ℓ_2) is down, and then remains in that state (modelling fail-stop of at most one of the two sites). The benign adversary 1_A can be modelled as a single-state FSM (with a self-loop). It is easy to formulate the composite system as a WSTS, by using the self-similar subterm ordering on processes, and the obvious trivial ordering on the adversary FSMs. It is also easy to exhibit the minimal elements and the effectiveness conditions for this system. Thus we can prove that $\mathit{Sys}_3 \circ A \approx \mathit{Sys}_2 \circ 1_A$. This being a finite-control system, the proof is quite easy. Our WSTS-based framework is also able to handle extensions to the system to permit persistent clients, which repeatedly request the file and deliver it *ad nauseum*.

Replicating *RepServer* again by placing it in $C_{rep}[\,]$ serves no purpose with respect to an adversary that can cause only one of the ℓ_i to fail[7]. However, if the adversary can cause $k > 1$, to fail, then a replication context should place the server at least $k+1$ locations.

3.3 Reliable transmission

We now address resilience to adversaries that make communication channels unreliable. Instead of presenting yet another verification of the ABP protocol [4] and its bisimulation proofs, which have been published several times before (e.g., [13]), we describe a small protocol for communication between a client R and a server Sr over a channel c that may arbitrarily reorder messages and omit messages. This protocol can be used as a basic building block within a larger protocol that builds a FIFO channel over a non-FIFO layer [1]. To our knowledge, this construction has not been earlier presented in as simple a formulation. Its core is similar to the "probe" construct of Afek and Gafni [7].

The server is extremely simple: it receives a request on channel b, and sends a message on channel c. At any point of time, it may segue to sending another value v'.

$$Sr(v) = (b.\overline{c}v.Sr(v)) + Sr(v'),$$

where $v' \neq v$ and $v, v' \in D$, some set of values (which we assume for convenience is of cardinality k). The server is representable as a (parameterised) non-deterministic finite-control machine.

A simple client R_s can be expressed as: $R_s = \overline{b}.cx.\overline{d}x.R_s$, where R_s sends requests on b and, on receiving a value on channel c, delivers it over channel d and repeats. If channel c does not omit or reorder values, R_s will produce all the values sent by the server in FIFO order. If, however, b, c are *lossy* channels, then some requests (or responses) may be lost in transmission. R_s may therefore get stuck waiting for messages. We make the client more persistent, modifying R_s to R_o,

$$R_o = !\,[\overline{b}]_1 \parallel [cx]_2.[\overline{d}x]_3.R_o$$

which decouples the (repeated) request sending on b from the receipt of messages on c. Since it works with lossy channels, R_o may omit delivering some messages, but all delivered messages appear in the order in which they were sent. Note that we have placed the three communication actions in three distinct "holes" ($[\,]_1, [\,]_2, [\,]_3$).

[7] This is true even when the language permits nested sublocations.

If, however, the channel c can reorder messages, it is possible to confuse messages corresponding to earlier requests with those for later requests. While this problem can be addressed by placing serial numbers on the messages, an interesting question is whether it can be solved without that mechanism. Accordingly, we modify the client:

$$\begin{array}{rcll} R_{ro}(p,n_1,\ldots,n_k) & = & [\overline{b}]_1.R_{ro}(p+1,n_1,\ldots,n_k) & \\ & & +R'(p,n_1,\ldots,n_k) & \\ & & & \\ R'(p,n_1,\ldots,n_k) & = & [cx]_2.\textbf{case } x \textbf{ of} & \\ & & \vdots & \\ & & v_i: & \textbf{if } (n_i>0) \textbf{ then} \\ & & & R_{ro}(p-1,n_1,\ldots,n_i-1,\ldots,n_k) \\ & & & \textbf{else } [\overline{d}x]_3.R_{ro}(0,p-1,\ldots,p-1) \\ & & \vdots & \end{array}$$

The client initiates a *round* of the protocol by sending a request to the server on channel b. Since the responses from the server may be reordered or dropped by the response channel c, the client has no way of knowing whether a response it receives is acknowledging its current request (a "fresh" message), or whether it is a response to a previous request. However, by pigeon-holing, if it receives more responses than pending unanswered requests from the past, it knows that the server has acknowledged its most recent request. For this it keeps a variable p, which is the number of requests sent in this round, and bounds the number of responses it requires for each value so that when it receives a response on c, it can determine that it has got enough responses to safely conclude that the value is a fresh one. Note that in our soution, *the server remains unchanged*, and in the client, all the parameters take *non-negative integral* values.

Let A_{ro} be an adversary that may reorder and omit messages (the interesting ability of the adversary lies in its being able to reorder responses on channel c), and let A_o be an adversary that may only omit messages on channel c. The equivalence to be established is:

$$(b)(c)(Sr\|R_{ro}(0,0,\ldots,0))\circ A_{ro} \approx (b)(c)(Sr\|R_o)\circ A_o$$

One may notice that the process R_o is not exactly syntactically embedded within a context in R_{ro}. However, the essence of the self-similarity constraints is met as every communication action comes into an enabled position at exactly corresponding points marked by the holes $[\,]_1,[\,]_2,[\,]_3$. Thus, the resulting ordering on processes $\preceq$ would (while still being decidable) be more complex, but nonetheless adheres to the self-similarity requirements and those of being a WSTS. In defining the states of the coupled transition system we consider the states of the server Sr, and that of the client R_o and R_{ro}. In identifying the latter, we demarcate as significant the three marked holes as points where to pin control. The start of a "round" is a significant point where the control of the R_o process is matched with that of the R_{ro} process. We take into account the parameters of the R_{ro} process in framing the $\preceq$ relation. Finally, we consider the channel states, adapting the subword ordering that has been used for lossy channels in order to deal with lossy-reordering channels. The details are omitted here, but WSTS technique provides a novel way of proving *operationally* the correctness of a protocol that has an unbounded state space.

By orienting this protocol in both directions, one can, at the price of counters (for the pending and new messages) obtain an implementation of a FIFO channel over a lossy reordering communication channel. The only way we know to avoid the counters is using sequence numbers on data items, as

in [19], but this implies an infinite message alphabet. For practical purposes, one usually assumes that sequence numbers can be recycled (in a "sliding window" fashion) under the belief that the channel does not deliver an "ancient" message. On similar lines, if one assumes a bound on the number of messages the channel may delay, then our counters can be limited by that bound. This may be a reasonable assumption since, as shown in [1], in any implementation of a FIFO channel over a lossy reordering one with a finite message alphabet, the more the messages that the channel delays, the more the messages that need be transmitted.

4 Conclusion

The question of whether a specified program behaviour can be achieved versus an arbitrary adversary is, of course, undecidable in general, with several famous impossibility results. Even the question of whether a given program is resilient to some particular adversary is in general not decidable.

What we have sought to do is to formalise an intuitive connection between resilience to an adversary and the self-similarity in the structure and behaviour of a program. It is a prescriptive framework, and we recognise that there are several other ways of correctly constructing fault-tolerant systems that do not fit this methodology. The program is constructed in terms of its core functionality (that generates the observables) and an absorptive context that soaks up the adversarial actions, but otherwise neither contributes nor detracts from the observable behaviour of a program[8]. While in our framework, we require that the adversary's moves are not directly observable (except in a denouement), the interaction between the adversary and the program may result in different observables from the normal execution. However, a fault-resilient program exhibits no difference.

Another way of thinking about the framework we have proposed is in terms of *abstractions* that operate as follows: consider the traces of the system by itself, and also those of the system composed with the adversary. Consider the equivalence relation that arises from an abstraction function which has the property of "stuttering" over moves by the adversary. A system may be considered tolerant of an adversary if every adversarial transition is within an induced equivalence class. While there is a certain simplicity to such an account, our proposed framework is richer in at least some respects: First, it is able to account for moves made in conjunction between the adversary and the system (the interactive moves). Second, it is able to relate the structure of the system with its behavioural self-similarity, and capture the intuition that the seemingly redundant parts of a program provide resilience against adversaries.

We are unaware of any published work on relating self-similarity with resilience. While there has been a body of work in which adversaries have been classified according to a partial order based on the severity of their disruptive capabilities, and relativising the notion of behavioural equivalence based on the adversary model, we believe this is less elegant or compelling than an account where a *standard* notion of behavioural equivalence is used. While we have used barbed bisimulation, other notions of equivalence may be appropriate in certain settings, and may provide easier methods for proving resilience. Another shortcoming of the previous approaches is that the adversarial model was incorporated into the syntax of the processes, whereas our approach supports more eclectic styles of specifying the transition system of processes in the context of an adversary.

Further, while the previous approaches seem to be confined to finite (or at best finite state processes) – where it is possible to find appropriate bisimulation relations, we are able to deal with a class of infinite-state systems by couching the problem in a WSTS framework. Although there has been substantial work on WSTSs, our use in proving systems to be fault-resilient seems to be novel. We believe that one can

[8] This factoring of a program into core and context cannot in general be automated.

in this framework also deal with composite adversarial models by combining the fault-resilient contexts in novel ways (e.g., embedding one kind of resilient context inside another), although this is beyond the scope of the current paper.

As we have presented very simple examples and introduced no new mechanisms for resilience, it may seem that the results are unsurprising. However, it is satisfying to not only confirm the correctness of intuitive constructions and folklore but also find an effective basis for demonstrating their correctness. It would be interesting to study whether our framework also provides a way of discovering *minimal* constructions for resilience against particular adversaries.

Of course, our proposed methodology should be put to further tests. For instance, it would be interesting to see whether techniques for building resilience in storage systems such as RAID [5] would be amenable to these techniques. A major area which needs addressing concerns cryptographic protocols and resilience in the face of cryptanalytic adversaries. The challenge there is to find the right notion of structural orderings, and recognising the contexts that provide resilience. Another domain that deserves greater study is coding theory and the use of redundant bits to provide error-correction. It is not obvious in such techniques whether convergence (reaching fixed-points) can be obtained.

In the future, we would also like to explore conditions under which, given a specification of the operational behaviour of the adversary, and the core functionality, it may be possible (if at all) to *synthesise* the fault-resilient context. We would like to develop a framework analogous to those of Liu, Joseph et al. where fault-tolerant versions of systems are developed using a stepwise refinement methodology, and transformations dealing with combined fault models.

In the current paper, we have used an operational framework; in the future, we would like to explore a logic-based formulation, using e.g., knowledge-based analysis techniques to reason about resilience in the same way that correctness of distributed systems/protocols has been studied [8].

Acknowledgements. We would like to thank the anonymous referees for their helpful suggestions in improving the paper. In particular, we would like to thank them for being generous in their assessment while being eagle-eyed in spotting mistakes in the submission.

References

[1] Yehuda Afek, Hagit Attiya, Alan Fekete, Michael J. Fischer, Nancy A. Lynch, Yishay Mansour, Da-Wei Wang & Lenore D. Zuck (1994): *Reliable Communication Over Unreliable Channels*. *Journal of the ACM* 41(6), pp. 1267–1297, doi:10.1145/195613.195651.

[2] Roberto M. Amadio, Ilaria Castellani & Davide Sangiorgi (1998): *On Bisimulations for the Asynchronous pi-Calculus*. *Theoretical Computer Science* 195(2), pp. 291–324, doi:10.1016/S0304-3975(97)00223-5.

[3] Roberto M. Amadio & Sanjiva Prasad (1994): *Localities and Failures (Extended Abstract)*. In: *Proceedings of the 14th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 1994)*, pp. 205–216, doi:10.1007/3-540-58715-2_126.

[4] K. A. Bartlett, R. A. Scantebury & P.T. Wilkinson (1969): *A note on reliable full-duplex transmission over half-duplex links*. *CACM* 12(5), pp. 260–261, doi:10.1145/362946.362970.

[5] Peter M. Chen, Edward K. Lee, Garth A. Gibson, Randy H. Katz & David A. Patterson (1994): *RAID: High-Performance, Reliable Secondary Storage*. *ACM Computing Surveys* 26, pp. 145–185, doi:10.1145/176979.176981.

[6] A. Finkel & Ph. Schnoebelen (2001): *Well-structured Transition Systems Everywhere! Theoretical Computer Science* 256(1-2), pp. 63–92, doi:10.1016/S0304-3975(00)00102-X.

[7] Eli Gafni & Yehuda Afek (1988): *End-to-End Communication in Unreliable Networks*. In: *Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing*, pp. 131–148, doi:10.1145/62546.62570.

[8] Joseph Y. Halpern & Lenore D. Zuck (1992): *A Little Knowledge Goes a Long Way: Knowledge-Based Derivations and Correctness Proofs for a Family of Protocols*. *Journal of the ACM* 39(3), pp. 449–478, doi:10.1145/146637.146638.

[9] Kohei Honda & Nobuko Yoshida (1993): *On Reduction-Based Semantics*. In: *Proceedings of the 13th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 1993)*, pp. 373–387, doi:10.1007/3-540-57529-4_70.

[10] Tomasz Janowski (1994): *Stepwise transformations for fault-tolerant design of CCS processes*. In: *Formal Description Techniques VII, Proceedings of the 7th IFIP WG6.1 International Conference on Formal Description Techniques*, pp. 505–520.

[11] Tomasz Janowski (1997): *On Bisimulation, Fault-Monotonicity and Provable Fault-Tolerance*. In: *Proceedings of the 6th International Conference on Algebraic Methodology and Software Technology (AMAST '97)*, pp. 292–306, doi:10.1007/BFb0000478.

[12] Padmanabhan Krishnan (1994): *A Semantic Characterisation for Faults in Replicated Systems*. *Theoretical Compututer Science* 128(1-2), pp. 159–177, doi:10.1016/0304-3975(94)90168-6.

[13] Kim Guldstrand Larsen & Robin Milner (1992): *A Compositional Protocol Verification Using Relativized Bisimulation*. *Information and Computation* 99(1), pp. 80–108, doi:10.1016/0890-5401(92)90025-B.

[14] Zhiming Liu & Mathai Joseph (1992): *Transformation of Programs for Fault-Tolerance*. *Formal Aspects of Computing* 4(5), pp. 442–469, doi:10.1007/BF01211393.

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

[16] Robin Milner & Davide Sangiorgi (1992): *Barbed Bisimulation*. In: *Proceedings of the 19th International Colloquium on Automata, Languages and Programming (ICALP '92)*, Springer-Verlag, pp. 685–695, doi:10.1007/3-540-55719-9_114.

[17] Doron Peled & Mathai Joseph (1994): *A Compositional Framework for Fault Tolerance by Specification Transformation*. *Theoretical Computer Science* 128(1&2), pp. 99–125, doi:10.1016/0304-3975(94)90166-X.

[18] James Riely & Matthew Hennessy (1997): *Distributed Processes and Location Failures (Extended Abstract)*. In: *Proceedings of the 24th International Colloquium on Automata, Languages and Programming, (ICALP'97)*, pp. 471–481, doi:10.1007/3-540-63165-8_203.

[19] N. V. Stenning (1976): *A data transfer protocol*. *Computer Networks* 1, pp. 99–110, doi:10.1016/0376-5075(76)90015-5.

Unique Parallel Decomposition for the π-calculus *

Matias David Lee
Univ. Lyon, ENS de Lyon, CNRS, UCB Lyon 1, LIP, France.
matias-david.lee@ens-lyon.fr

Bas Luttik
Eindhoven University of Technology, The Netherlands.
s.p.luttik@tue.nl

A (fragment of a) process algebra satisfies unique parallel decomposition if the definable behaviours admit a unique decomposition into indecomposable parallel components. In this paper we prove that finite processes of the π-calculus, i.e. processes that perform no infinite executions, satisfy this property modulo strong bisimilarity and weak bisimilarity. Our results are obtained by an application of a general technique for establishing unique parallel decomposition using decomposition orders.

1 Introduction

A (fragment of a) process algebra has *unique parallel decomposition* (UPD) if all definable behaviours admit a unique decomposition into indecomposable parallel components. In this paper we prove that finite processes definable in the π-calculus satisfy this property modulo strong bisimilarity and modulo weak bisimilarity.

From a theoretical point of view, this property is interesting because it can be used to prove other theoretical properties about process calculi. For instance, relying on unique parallel decomposition, Moller proves in [18, 19] that PA and CCS cannot be finitely axiomatized without auxiliary operations, and Hirshfeld and Jerrum prove in [12] that bisimilarity is decidable for normed PA. Unique parallel decomposition can be also used to define a notion of normal form. Such a notion of normal form is useful in completeness proofs for equational axiomatizations in settings in which an elimination theorem for parallel composition is lacking (see, e.g., [1, 2, 3, 9, 11]). In [13], UPD is used to prove complete axiomatisation and decidability results in the context of a higher-order process calculus.

From a practical point of view, unique parallel decomposition can be used to devise methods for finding the maximally parallel implementation of a behaviour [6], or for improving verification methods [10]. In [8], a unique parallel decomposition result is used as a tool in the comparison of different security notions in the context of electronic voting.

The UPD property has been widely studied for different process calculi and variants of the parallel operator. Milner and Moller were the first to establish a unique parallel decomposition theorem; they proved the property for a simple process calculus that allows the specification of all finite behaviours up to strong bisimilarity and includes parallel composition in the form of pure interleaving without interaction between its components [16]. Moller, in his dissertation [17], extended this result replacing interleaving parallel composition by CCS parallel composition, and then also considering weak bisimilarity. Christensen, also in his dissertation [5], proved unique decomposition for normed behaviours recursively definable modulo strong bisimilarity, and for all behaviours recursively definable modulo

*M.D. Lee has been supported by the project ANR 12IS02001 PACE

D. Gebler and K. Peters (Eds.): Combined Workshop on Expressiveness in
Concurrency and Structural Operational Semantics (EXPRESS/SOS 2016).
EPTCS 222, 2016, pp. 45–59, doi:10.4204/EPTCS.222.4

distributed bisimilarity; the proof of the latter result relies on a cancellation law for parallel composition up to distributed bisimilarity, first established by Castellani as [4, Lemma 4.14].

Most of the aforementioned unique parallel decomposition results were established with subsequent refinements of an ingenious proof technique attributed to Milner. In [15], the notion of *decomposition order* is introduced in order to formulate a sufficient condition on commutative monoids that facilitates an abstract version of Milner's proof technique. It is then proved that if a partial commutative monoid can be endowed with a decomposition order, then it has unique decomposition. Thus, an algebraic tool is obtained that allows one to prove UPD for a process calculus by finding a decomposition order.

The tool can deal with most of the settings aforementioned. In this paper, we show how the tool can also be applied to obtain unique parallel decomposition results for finite processes of the π-calculus w.r.t. strong bisimilarity and w.r.t. weak bisimilarity. But, to this end, we do face two complications: The first complication, in the context of the π-calculus is that, as opposed to previous settings, the decomposition order is not directly induced on the commutative monoid of processes by the transition relation. The culprit is that, in general, two parallel components may fuse into a single indecomposable process as a result of scope extrusion. To define the decomposition order we consider a fragment of the transition relation that avoids this phenomenon. The second complication, which arises only in the case of weak bisimilarity, is that certain transitions are deemed unobservable, and that, as a consequence, there are transitions that do not change state (are between weakly bisimilar processes). We demonstrate that a decomposition order can, nevertheless, be obtained by ignoring such *stuttering transitions*.

The paper [7] studies unique parallel decomposition w.r.t. both strong bisimilarity and weak bisimilarity for the applied π-calculus. The applied π-calculus is a variant of the π-calculus that was designed for the verification of cryptographic protocols. Its main feature is that channels can only transmit variables and the values of the variables are set using *active substitutions*. Roughly, active substitution is an extension of the grammar of the π-calculus that works as a 'memory' that save the value of a variable. Because the variables in a transition are observable but the 'memories' are not, it is possible to mask sensitive information. The proof of the result for the strong case in [7] relies on induction over the *norm* of a process and the fact that the norm of the arguments of a parallel composition is less than the norm of the parallel composition. Unfortunately, this property is not true because of the restriction operator (see Section 4 for a counter example). This is the reason why we restrict ourselves to finite processes in the strong setting. The proof of the weak case in [7] follows the proof technique attributed to Milner. The general techniques from [15] cannot be applied directly in the setting of the applied π-calculus due to the active substitutions.

In [14], the second author presented an adaptation of the general result of [15] in order to make it suitable for establishing unique parallel decomposition in settings with a notion of unobservable behaviour. The ensued technique amounts to showing that the transition relation induces a so-called *weak* decomposition order satisfying a property that is called *power cancellation*. In the present paper, we show how, instead of using the adapted technique from [14], the original technique from [15] may be applied in settings with a notion of unobservable behaviour, considering a stutter-free fragment of the transition relation. This method appears to be simpler than the method suggested by the result in [14].

The paper is organized as follows. In Section 2, we briefly recall the abstract framework introduced in [15] to prove UPD results. In Section 3 we recall the syntax and different semantics of the π-calculus. Section 4 is composed of two subsections. In Section 4.1 we introduce the notion of *depth* of a process and we prove some properties of this notion. In Section 4.2 we use these results and the result in Section 2 to prove that finite processes of the π-calculus satisfy unique parallel decomposition w.r.t. strong bisimilarity. Section 5 follows a similar structure. In Section 5.1 we introduce the notion of *processes without stuttering transitions* and we prove some properties of this kind of processes. These properties

and the result in Section 2 are used in Section 5.2 to prove that finite processes of the π-calculus satisfy unique parallel decomposition w.r.t. weak bisimilarity. In Section 6 we present some final remarks.

2 Decomposition orders

In this section, we briefly review the theory of unique decomposition for commutative monoids that we shall apply in the remainder of the paper to prove UPD results in the context of the π-calculus.

Definition 1. *A* commutative monoid *is a set M with a distinguished element e and a binary operation on M denoted by $\cdot$ such that for all $x,y,z \in M$:*

- $x \cdot (y \cdot z) = (x \cdot y) \cdot z$ *(associativity);*
- $x \cdot y = y \cdot x$ *(commutativity);*
- $x \cdot e = e \cdot x = x$ *(identity).*

In the remainder of the paper we often suppress the symbol $\cdot$ or use $\|$.

Definition 2. *An element p of a commutative monoid M is called* indecomposable *if $p \neq e$ and $p = xy$ implies $x = e$ or $y = e$.*

Definition 3. *Let M be a commutative monoid. A* decomposition *in M is a finite multi-set $\wr p_1, \ldots, p_k \int$ of indecomposable elements of M such that $p_1 \cdot p_2 \cdots p_k$ is defined. The element $p_1 \cdot p_2 \cdots p_k$ in M will be called the* composition associated *with the decomposition $\wr p_1, \ldots, p_k \int$, and, conversely, we say that $\wr p_1, \ldots, p_k \int$ is a* decomposition *of the element $p_1 \cdot p_2 \cdots p_k$ of M. Decompositions $d = \wr p_1, \ldots, p_k \int$ and $d' = \wr p'_1, \ldots, p'_l \int$ are equivalent in M (notation $d \equiv d'$) if they have the same composition, i.e. $p_1 \cdot p_2 \cdots p_k = p'_1 \cdots p'_l$. A decomposition d in M is* unique *if $d \equiv d'$ implies $d = d'$ for all decompositions d' in M. We say that an element x of M has a* unique decomposition *if it has a decomposition and this decomposition is unique. If every element of M has a unique decomposition, then we say that M has* unique decomposition.

Theorem 1 below gives a sufficient condition to ensure that a commutative monoid M has unique decomposition. It requires the existence of a *decomposition order* for M.

Definition 4. *Let M be a commutative monoid; a partial order $\preccurlyeq$ on M is a* decomposition order *if*

1. *it is* well-founded*, i.e., for every non-empty subset $\hat{M} \subseteq M$ there is $m \in \hat{M}$ such that for all $m' \in M$, $m' \preccurlyeq m$ implies $m' = m$. In this case, we say that m is a $\preccurlyeq$-minimal element of $\hat{M}$;*
2. *the identity element e of M is the least element of M with respect to $\preccurlyeq$, i.e., $e \preccurlyeq x$ for all x in M;*
3. *$\preccurlyeq$ is* strictly compatible*, i.e., for all $x,y,z \in M$ if $x \prec y$ (i.e. $x \preccurlyeq y$ and $x \neq y$) and yz is defined, $xz \prec yz$;*
4. *it is* precompositional*, i.e., for all $x,y,z \in M$ $x \preccurlyeq yz$ implies $x = y'z'$ for some $y' \preccurlyeq y$ and $z' \preccurlyeq z$; and*
5. *it is* Archimedean*, i.e., for all $x,y \in M$ $x^n \preccurlyeq y$ for all $n \in \mathbb{N}_0$ implies that $x = e$.*

Theorem 1 ([15])**.** *Every commutative monoid M with a decomposition order has unique decomposition.*

3 The π-calculus

We recall the syntax of the π-calculus and the rules to define the transition relation [20]. We assume a set of *names* or *channels* $\mathcal{V}$. We use a,b,c,x,y,z to range over $\mathcal{V}$.

$$\frac{}{\overline{x}y.P \xrightarrow{\overline{x}y} P}\ \text{(Out)} \qquad \frac{}{x(z).P \xrightarrow{xy} P\{y/z\}}\ \text{(Inp)} \qquad \frac{}{\tau.P \xrightarrow{\tau} P}\ \text{(Tau)} \qquad \frac{\pi.P \xrightarrow{\alpha} P}{[x=x]\pi.P \xrightarrow{\alpha} P}\ \text{(Mat)}$$

$$\frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'}\ \text{(Sum-L)} \qquad \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}\ bn(\alpha) \cap fn(Q) = \varnothing\ \text{(Par-L)}$$

$$\frac{P \xrightarrow{\overline{x}y} P' \quad Q \xrightarrow{xy} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}\ \text{(Comm-L)} \qquad \frac{P \xrightarrow{\overline{x}(z)} P' \quad Q \xrightarrow{xz} Q'}{P \mid Q \xrightarrow{\tau} \nu z(P' \mid Q')}\ z \notin fn(Q)\ \text{(Close-L)}$$

$$\frac{P \xrightarrow{\alpha} P'}{\nu z(P) \xrightarrow{\alpha} \nu z(P')}\ z \notin n(\alpha)\ \text{(Res)} \qquad \frac{P \xrightarrow{\overline{x}z} P'}{\nu z(P) \xrightarrow{\overline{x}(z)} P'}\ z \neq x\ \text{(Open)} \qquad \frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P' \mid !P}\ \text{(Rep-Act)}$$

$$\frac{P \xrightarrow{\overline{x}y} P' \quad P \xrightarrow{xy} P''}{!P \xrightarrow{\tau} (P' \mid P'') \mid !P}\ \text{(Rep-Comm)} \qquad \frac{P \xrightarrow{\overline{x}(z)} P' \quad P \xrightarrow{xz} P''}{!P \xrightarrow{\tau} (\nu z(P' \mid P'')) \mid !P}\ z \notin fn(P)\ \text{(Rep-Close-L)}$$

Table 1: Transition rules for the π-calculus

Definition 5. *The* processes, summations *and* prefixes *of the π-calculus are given respectively by*

$$\begin{array}{rcl} P ::= & M \mid P \mid P' \mid \nu z.P \mid !P \\ M ::= & \mathbf{0} \mid \pi.P \mid M + M' \\ \pi ::= & \overline{x}y \mid x(z) \mid \tau \mid [x=y]\pi \end{array}$$

We denote with Π the set of processes of the π-calculus.

An occurrence of a name $z \in \mathcal{V}$ is *bound* in a process P if it is in the scope of a restriction νz or of an input $a(z)$. A name $a \in \mathcal{V}$ is *free* in a process P if there is at least one occurrence of a that is not bound. We write $bn(P)$ and $fn(P)$ to denote, respectively, the set of bound names and free names of a process P. The *set of names* of a process P is defined by $n(P) = bn(P) \cup fn(P)$. We employ the following convention of the π-calculus w.r.t. names.

Convention 1. *[20, P.47] In any discussion, we assume that the bound names of any processes or actions under consideration are chosen to be different from the names free in any other entities under consideration, such as processes, actions, substitutions, and sets of names. This convention is subject to the limitation that in considering a transition $P \xrightarrow{\overline{x}(z)} Q$, the name z that is bound in $\overline{x}(z)$ and in P may occur free in Q. This limitation is necessary for expressing scope extrusion.*

The transition relation associated to a term is defined by the rules in Table 1, where we have omitted the symmetric version of the rules (Sum-L), (Par-L), (Comm-L) and (Close-L). We denote with A the set of *visible actions* that can be executed by a process $P \in \Pi$, i.e. $A = \{xy \mid x,y \in \mathcal{V}\} \cup \{\overline{x}y \mid x,y \in \mathcal{V}\} \cup \{\overline{x}(z) \mid x,z \in \mathcal{V}\}$. The action τ is the *internal action*. We define $A_\tau = A \cup \{\tau\}$. For $P,P' \in \Pi$ and $\alpha \in A_\tau$, we write $P \xrightarrow{\alpha} P'$ if there is a derivation of $P \xrightarrow{\alpha} P'$ with rules in Table 1.

Definition 6. Strong bisimilarity *is the largest symmetric relation over Π, notation $\sim$, such that whenever $P \sim Q$, if $P \xrightarrow{\alpha} P'$ then there is Q' s.t. $Q \xrightarrow{\alpha} Q'$ and $P' \sim Q'$.*

The relation $\sim$ is not compatible with input prefix: We have that

$$\overline{z}x \mid a(y) \sim \overline{z}x.a(y) + a(y).\overline{z}x \ ,$$

whereas

$$b(a).(\overline{z}x \mid a(y)) \not\sim b(a).(\overline{z}x.a(y) + a(y).\overline{z}),$$

because when z is received over the channel a, we have

$$(\overline{z}x \mid a(y))\{a/z\} \not\sim (\overline{z}x.a(y) + a(y).\overline{z})\{a/z\} \ .$$

Hence, $\sim$ is not a congruence for the full syntax of the π-calculus. It is, however, a so-called *non-input congruence* (see [20, Theorem 2.2.8]): it is compatible with all the other constructs in the syntax. In the present paper we shall only use the fact that $\sim$ is compatible with parallel composition, i.e., if $P_1 \sim Q_1$ and $P_2 \sim Q_2$, then $P_1 \mid P_2 \sim Q_1 \mid Q_2$.

We recall now the weak variant of bisimilarity. We write $P \Longrightarrow P'$ if $P = P'$ or if there are $P_0, \ldots, P_n$ with $n > 0$ s.t. $P = P_0 \xrightarrow{\tau} \ldots \xrightarrow{\tau} P_n = P'$. We write $P \overset{\alpha}{\Longrightarrow} Q$ with $\alpha \in A_\tau$ if there are P', Q' s.t. $P \Longrightarrow P' \xrightarrow{\alpha} Q' \Longrightarrow Q$. Notice the difference between $P \Longrightarrow P'$ and $P \overset{\tau}{\Longrightarrow} P'$, in the second case, at least one τ-transition is executed.

Definition 7. Weak bisimilarity *is the largest symmetric relation over* Π*, notation* $\approx$*, such that whenever* $P \approx Q$*, (i) if* $P \xrightarrow{\alpha} P'$ *with* $\alpha \in A$ *then there is* Q' *s.t.* $Q \overset{\alpha}{\Longrightarrow} Q'$ *and* $P' \approx Q'$ *and (ii) if* $P \xrightarrow{\tau} P'$ *then there is* Q' *s.t.* $Q \Longrightarrow Q'$ *and* $P' \approx Q'$*.*

Like strong bisimilarity, it is only possible to prove that $\approx$ is a congruence for non-input contexts (see [20, Theorem 2.4.22]).

4 Unique decomposition with respect to strong bisimilarity

In this section, we shall use the result presented in Section 2 to prove that every *finite* π-calculus process has a unique parallel decomposition w.r.t. strong bisimilarity. In Section 4.1 we introduce the definition of depth of a process and some of its properties. We also explain why we restrict our development to finite processes. In Section 4.2 we present the unique decomposition result.

4.1 The depth of a process

Given a set X, we denote with X^* the set of finite sequences over X, where $\varepsilon \in X^*$ is the empty sequence. For $\omega = \alpha_1\alpha_2\cdots\alpha_n \in A_\tau^*$ with $n > 0$, we write $P \xrightarrow{\omega} P'$ if there are processes $P_0, P_1, \ldots, P_n$ s.t. $P = P_0 \xrightarrow{\alpha_1} P_1 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_n} P_n = P'$. If $\omega = \varepsilon$, then $P \xrightarrow{\omega} P'$ implies $P' = P$. If we are not interested in P', we write $P \xrightarrow{\omega}$. In addition, we write $P \downarrow$ if for all $\alpha \in A_\tau$, $P \not\xrightarrow{\alpha}$.

Definition 8. *Let* $length : A_\tau^* \to \mathbb{N}_0$ *be the function defined by*

$$length(\omega) = \begin{cases} 0 & \text{if } \omega = \varepsilon, \\ length(\omega') + 1 & \text{if } \omega = \alpha\omega' \text{ and } \alpha \neq \tau \\ length(\omega') + 2 & \text{if } \omega = \alpha\omega' \text{ and } \alpha = \tau \end{cases}$$

Definition 9. *A process $P \in \Pi$ is* normed *if there is $\omega \in A_\tau^*$ such that $P \xrightarrow{\omega} P' \downarrow$. We denote with Π_n the set of normed processes. The depth $: \Pi_n \to \mathbb{N}_0 \cup \{\infty\}$ and the norm $: \Pi_n \to \mathbb{N}_0$ of a normed process $P \in \Pi$ are defined, respectively, by*

$$depth(P) = \sup(\{length(\omega) \mid P \xrightarrow{\omega} P' \text{ and } P' \downarrow\})$$
$$norm(P) = \inf(\{length(\omega) \mid P \xrightarrow{\omega} P' \text{ and } P' \downarrow\})$$

Where $\sup(X) = \infty$ *whenever X is an infinite set, and* $\inf(\varnothing) = \infty$.

We remark that we have assigned a higher weight to occurrences of the label τ in the definition of the length of a sequence $\omega \in A_\tau^*$. This is to ensure that depth is additive w.r.t. parallel composition (i.e., the depth of a parallel composition is the sum of the depths of its components), as we shall prove in Lemma 6) below. As opposed to other process calculi for which unique decomposition has been established (see, e.g. [15]), due to scope extrusion, norm is not additive for the π-calculus: Consider

$$P = P_0 \mid P_1 = \nu z(\overline{a}z) \mid a(x).!\overline{x}a$$

P is normed because $P \xrightarrow{\tau} \nu z(\mathbf{0} \mid !\overline{z}a) \downarrow$ but P_1 is not because it only performs an execution of infinite length. Then, to ensure this kind of properties, one approach could be just consider normed processes. Unfortunately this is not enough. Consider

$$Q = Q_0 \mid Q_1 = \nu z(\overline{a}z) \mid a(x).\overline{x}a$$

Processes Q, Q_0 and Q_1 are normed and, moreover, they perform no infinite execution. Despite this, we have that $norm(Q) = 2$, because $Q \xrightarrow{\tau} \nu z(\mathbf{0} \mid \overline{z}a) \downarrow$, and $norm(Q_0) + norm(Q_1) = 1 + 2 = 3$. Moreover, notice that the norm of the arguments of a parallel composition is not less than the norm of the parallel composition, i.e. $norm(Q) = 2$ and $norm(Q_1) = 2$. In particular, these examples show that item 4 in Lemma 3 of [7] (norm is additive) is false, and, as a consequence, some proofs in [7] are flawed. The authors of [7] proposed a solution to this problem that we discuss in the conclusion of this paper. So, to facilitate inductive reasoning, we will consider *finite processes* and depth.

Definition 10. *A process $P \in \Pi$ is* finite *if there is $n \in \mathbb{N}_0$ s.t. there is no $\omega = \alpha_1\alpha_2\cdots\alpha_{n+1} \in A_\tau^*$ such that $P \xrightarrow{\omega}$. We denote with Π_f the set of finite processes of* Π.

Following the last example, we have that $Q, Q_0, Q_1 \in \Pi_f$ and $depth(Q) = 3 = depth(Q_0) + depth(Q_1)$.

To conclude this section we present a collection of results including lemmas and theorems. Most of the lemmas are only needed to prove the theorems. The theorems and only few lemmas will be used in the next section. Theorem 2 states that bisimilar processes have the same depth. Theorem 3 states that the depth of a parallel composition of two processes not bisimilar to $\mathbf{0}$ is greater than the depth of each process. Thanks to these results, we will able to extend the notion of depth to equivalence classes and apply inductive reasoning.

Lemma 1. *For all $P \in \Pi_f$, $P \not\sim \mathbf{0}$ implies $depth(P) > 0$.*

Lemma 2. *If $P \in \Pi_f$ and $P \xrightarrow{\alpha} P'$, $\alpha \in A_\tau$, then $P' \in \Pi_f$ and $depth(P) > depth(P')$.*

Theorem 2. *If $P \sim Q$ then $P \in \Pi_f$ iff $Q \in \Pi_f$; moreover, $depth(P) = depth(Q)$.*

Proof. Suppose that $P \sim Q$. Then, clearly, $P \xrightarrow{\omega}$ iff $Q \xrightarrow{\omega}$, and hence $P \in \Pi_f$ iff $Q \in \Pi_f$.

To prove that $depth(P) = depth(Q)$, first note that if $P, Q \notin \Pi_f$, then $depth(P) = \infty = depth(Q)$. In the case that remains, both $depth(P)$ and $depth(Q)$ are natural numbers; we proceed by induction over $depth(P)$. If $depth(P) = 0$ then $P \sim \mathbf{0}$ by Lemma 1 , so $Q \sim \mathbf{0}$ and therefore $depth(Q) = 0$. Suppose now $depth(P) = n > 0$. Assume that the statement holds for processes with depth less than n. Suppose $depth(Q) = m > n$ then there is Q' s.t. $Q \xrightarrow{\alpha} Q'$ and $m = length(\alpha) + depth(Q')$. By definition of $\sim$, we get $P \xrightarrow{\alpha} P'$, $P' \sim Q'$. By Lemma 2, $depth(P') < depth(P)$, therefore $depth(P') = depth(Q')$ and $depth(P) \geq depth(P') + length(\alpha) = m > n = depth(P)$, i.e. we get a contradiction. Similarly, for the case $depth(Q) = m < n$, we can reach a contradiction by considering a transition $P \xrightarrow{\alpha} P'$ with $n = length(\alpha) + depth(P')$. Then we can conclude $depth(P) = depth(Q)$. □

Lemma 3. *Let $P, P' \in \Pi_f$, $\omega = \alpha\omega' \in A_\tau^*$ be such that $P \xrightarrow{\alpha} P' \xrightarrow{\omega'}$ and $depth(P) = length(\omega)$. Then $depth(P') = length(\omega')$ and therefore $depth(P) = depth(P') + length(\alpha)$.*

Lemma 4. *For all $P \in \Pi_f$, $depth(P) \geq depth(\nu z(P))$ for all $z \in \mathcal{V}$.*

Lemma 5. *Let $P, Q \in \Pi_f$ and $\omega \in A_\tau^*$ be such that $P \mid Q \xrightarrow{\omega}$ and $length(\omega) = depth(P \mid Q)$. Then, there are $\omega_1, \omega_2 \in A_\tau^*$ such that $P \xrightarrow{\omega_1}$, $Q \xrightarrow{\omega_2}$ and $length(\omega_1) + length(\omega_2) = length(\omega)$.*

Proof. We proceed by complete induction on $n = depth(P) + depth(Q)$. Suppose that the property holds for parallel compositions of finite processes such that the sum of the depths is smaller than $n > 0$. Let $\omega = \alpha\omega'$ and R be such that $length(\omega) = n$ and $P \mid Q \xrightarrow{\alpha} R \xrightarrow{\omega'}$. We analyse the different ways of deriving the first transition (we omit the symmetric cases).

- Case (Par-L). Then $P \xrightarrow{\alpha} P'$ and $R = P' \mid Q$. By Lemma 3 and induction $depth(P' \mid Q) = length(\omega') < n$ and then there are ω_1 and ω_2 s.t. $P' \xrightarrow{\omega_1}$, $Q \xrightarrow{\omega_2}$ and $length(\omega_1) + length(\omega_2) = length(\omega')$. Then $P \xrightarrow{\alpha\omega_1}$ and $length(\alpha\omega_1) + length(\omega_2) = length(\omega)$.
- Case (Comm-L). Then $P \xrightarrow{\overline{x}y} P'$, $Q \xrightarrow{xy} Q'$ and $R = P' \mid Q'$ and $\alpha = \tau$. By Lemma 3 and induction $depth(P' \mid Q') = length(\omega') < length(\tau) + length(\omega') = n$ and then there are ω_1 and ω_2 s.t. $P' \xrightarrow{\omega_1}$, $Q' \xrightarrow{\omega_2}$ and $length(\omega_1) + length(\omega_2) = length(\omega')$. Then $P \xrightarrow{\overline{x}y\omega_1}$ and $Q \xrightarrow{xy\omega_2}$ and $length(\overline{x}y\omega_1) + length(xy\omega_2) = length(\tau) + length(\omega') = length(\omega)$.
- Case (Close-L). Then $P \xrightarrow{\overline{x}(z)} P', Q \xrightarrow{xz} Q'$, $\alpha = \tau$ and $R = \nu z(P' \mid Q')$. The side condition of (Close-L) allows us to use the rules (Par-L) and its symmetric version, then $P \mid Q \xrightarrow{\overline{x}(z)xz} P' \mid Q'$. On one hand, by Lemma 4, $depth(P' \mid Q') \geq depth(\nu z(P' \mid Q'))$. On the other hand, $depth(P' \mid Q') \leq depth(\nu z(P' \mid Q'))$ because $\omega = \tau\omega'$ is a maximal execution, $length(\tau) = length(\overline{x}zxz)$, $\nu z(P' \mid Q') \xrightarrow{\omega'}$ and by Lemma 3. Then $depth(P' \mid Q') = depth(\nu z(P' \mid Q')) < n$. Moreover, there is ω'' such that $P' \mid Q' \xrightarrow{\omega''}$ and $length(\overline{x}zxz\omega'') = depth(P \mid Q)$. Then $P \mid Q \xrightarrow{\overline{x}(z)} P' \mid Q \xrightarrow{xz\omega''}$ with $length(xz\omega'') < n$. From this point we can repeat the proof of the first case.

□

Lemma 6. *For all processes $P, Q \in \Pi_f$, $depth(P \mid Q) = depth(P) + depth(Q)$.*

Proof. By Lemma 5 we can ensure $depth(P \mid Q) \leq depth(P) + depth(Q)$. On the other hand, by Convention 1, we have that $P \xrightarrow{\omega}$ or $Q \xrightarrow{\omega}$ implies $P \mid Q \xrightarrow{\omega}$. This allows us to conclude $depth(P \mid Q) \geq depth(P) + depth(Q)$ and therefore $depth(P \mid Q) = depth(P) + depth(Q)$. □

Lemma 7. *For all $P,Q \in \Pi$, $P,Q \in \Pi_f$ iff $P \mid Q \in \Pi_f$.*

Theorem 3. *If $P,Q,R \in \Pi_f$, $P \not\sim \mathbf{0}$, $Q \not\sim \mathbf{0}$ and $P \mid Q \sim R$ then $depth(P) < depth(R)$ and $depth(Q) < depth(R)$.*

Proof. By Lemma 1, $depth(P) > 0$, $depth(Q) > 0$. By Theorem 2, $depth(R) = depth(P \mid Q)$. By Lemma 6, $depth(R) = depth(P) + depth(Q)$ and we conclude $depth(P) < depth(R)$ and $depth(Q) < depth(R)$. □

4.2 Unique decomposition

The commutative monoid associated with Π_f modulo $\sim$ is defined by

- $\mathbf{P}_\sim = \{[P]_\sim : P \in \Pi_f\}$ where $[P]_\sim = \{P' : P' \sim P\}$
- $e = [\mathbf{0}]_\sim \in \mathbf{P}_\sim$.
- $\| : \mathbf{P}_\sim \times \mathbf{P}_\sim \to \mathbf{P}_\sim$ is such that $[P]_\sim \| [Q]_\sim = [P \mid Q]_\sim$

By Lemma 7 we have that the definition of $\|$ is sound. By Theorem 2 we have that all $P' \in [P]_\sim$ have the same depth. Then we can lift function *depth* to $\mathbf{P}_\sim$, i.e. $depth([P]_\sim) = depth(P)$.

Lemma 8. *$\mathbf{P}_\sim$ with neutral element $[\mathbf{0}]_\sim$ and binary operation $\|$ is a commutative monoid. I.e., $\| \subseteq \mathbf{P}_\sim \times \mathbf{P}_\sim$ satisfies the associativity, commutativity and identity properties.*

In order to use the Theorem 1 we need to define on $\mathbf{P}_\sim$ a decomposition order. In [15, 14], it is shown that the transition relation directly induces a decomposition order on a commutative monoid of processes. In the case of the π-calculus, however, the order induced by the transition relation cannot be directly used, as is illustrated by the following example: Define a binary relation $\rightsquigarrow \subseteq \mathbf{P}_\sim \times \mathbf{P}_\sim$ by $[R]_\sim \rightsquigarrow [S]_\sim$ if there is $R' \in [R]_\sim$ and $S' \in [S]_\sim$ such that $R' \xrightarrow{\alpha} S'$. We denote the inverse of the reflexive-transitive closure of $\rightsquigarrow$ by $\preccurlyeq_\rightsquigarrow$, i.e., $\preccurlyeq_\rightsquigarrow = (\rightsquigarrow^*)^{-1}$. The order $\preccurlyeq_\rightsquigarrow$ is not precompositional. Consider the processes $P = \nu z.(\overline{a}z.\overline{z}c.\overline{c}a)$ and $Q = a(x).x(y).\overline{y}b$. Then

- $P \mid Q = \nu z.(\overline{a}z.\overline{z}c.\overline{c}a) \mid a(x).x(y).\overline{y}b \xrightarrow{\tau} \nu z.(\overline{z}c.\overline{c}a \mid z(y).\overline{y}b) = R$ and therefore
- $[P]_\sim \| [Q]_\sim = [P \| Q]_\sim \rightsquigarrow [R]_\sim = [\nu z.(\overline{z}c.\overline{c}a \mid z(y).\overline{y}b)]_\sim$.
- Note that R executes only one transition, i.e. $\nu z.(\overline{z}c.\overline{c}a \mid z(y).\overline{y}b) \xrightarrow{\tau} \nu z(\overline{c}a \mid \overline{c}b)$, then it is clear that there are no processes P' and Q' s.t.

$$[\nu z.(\overline{a}z.\overline{z}c.\overline{c}a)]_\sim \rightsquigarrow^* [P']_\sim \qquad [a(x).x(y).\overline{y}b]_\sim \rightsquigarrow^* [Q']_\sim \qquad [P']_\sim \| [Q']_\sim = [\nu z(\overline{z}c.\overline{c}a \mid z(y).\overline{y}b)]_\sim$$

The particularity of this example is the *scope extrusion*. We need to define an order based on a fragment of the transition relation that avoids this phenomenon. We shall define the partial order $\preccurlyeq$ over $\mathbf{P}_\sim$ as the reflexive-transitive closure of the relation $\to \subseteq \mathbf{P}_\sim \times \mathbf{P}_\sim$, which is, in turn, defined as follows:

$$\to_0 = \{([P]_\sim, [Q]_\sim) : P \xrightarrow{\alpha} Q, \alpha \in A_\tau \text{ and } \nexists P_0, P_1 \in \Pi_f \text{ s.t. } P_0 \not\sim \mathbf{0}, P_1 \not\sim \mathbf{0}, P_0 \mid P_1 \sim P\}$$

$$\begin{aligned}\to_{k+1} = {} & \{([P_0 \mid P_1]_\sim, [Q_0 \mid P_1]_\sim) : [P_0]_\sim \to_k [Q_0]_\sim, P_1 \in \Pi_f\} \\ & \cup \{([P_0 \mid P_1]_\sim, [P_0 \mid Q_1]_\sim) : [P_1]_\sim \to_k [Q_1]_\sim, P_0 \in \Pi_f\}\end{aligned}$$

$$\to = \bigcup_{k=0}^{\infty} \to_n \quad .$$

The partial order $\preccurlyeq$ is defined as the inverse of the reflexive-transitive closure of $\to$ i.e., $\preccurlyeq = (\to^*)^{-1}$. We write $[P]_\sim \prec [Q]_\sim$ if $[P]_\sim \preccurlyeq [Q]_\sim$ and $[P]_\sim \neq [Q]_\sim$. Notice that the definition of $\to$ avoids any kind of communications between the arguments of the parallel operator, this ensures that the scope extrusion is also avoided.

Lemma 9. *If* $[P]_\sim \to [Q]_\sim$ *then* $depth([Q]_\sim) < depth([P]_\sim)$.

Lemma 10. $\preccurlyeq$ *is a partial order.*

Proof. We have to prove that $\preccurlyeq$ is reflexive, antisymmetric, and transitive. $\preccurlyeq$ is reflexive and transitive because it is the reflexive-transitive closure of $\to$. To prove that $\preccurlyeq$ is antisymmetric notice that $[P]_\sim \prec [Q]_\sim$ implies $[Q]_\sim = [P_n]_\sim \to \ldots \to [P_1]_\sim \to [P_0]_\sim = [P]_\sim$ for $n > 0$ and then, by Lemma 9, $depth([P]_\sim) < depth([Q]_\sim)$. Therefore $[P]_\sim \preccurlyeq [Q]_\sim$ and $[Q]_\sim \preccurlyeq [P]_\sim$ implies $[P]_\sim = [Q]_\sim$. □

In Lemma 12, we prove that $\preccurlyeq$ is a decomposition order. To prove this result we need to add a last auxiliary result, Lemma 11.

Lemma 11. *If* $P \in \Pi_f$ *and* $depth(P) > 0$ *then there is* Q *s.t.* $[P]_\sim \to [Q]_\sim$.

Proof. We proceed by complete induction over $n = depth(P)$. Assume that the hypothesis holds for values less than $n \geq 1$. Suppose there are no $P_0, P_1 \in \Pi_f$ such that $P \sim P_0 \mid P_1$, $P_0 \not\sim \mathbf{0}$ and $P_1 \not\sim \mathbf{0}$. Given that $n \geq 1$ then there is $\alpha \in A_\tau$ s.t. $P \xrightarrow{\alpha} P'$. Then $[P]_\sim \to_0 [P']_\sim$ and therefore $[P]_\sim \to [P']_\sim$. Finally, we can define $[Q]_\sim = [P']_\sim$.

Suppose there are $P_0, P_1 \in \Pi_f$ such that $P \sim P_0 \mid P_1$, $P_0 \not\sim \mathbf{0}$ and $P_1 \not\sim \mathbf{0}$, then $[P]_\sim = [P_0 \mid P_1]_\sim$. By Theorem 3 $depth(P_0) < depth(P)$ and by Lemma 1 $depth(P_0) > 0$. By induction there is Q_0 s.t. $[P_0]_\sim \to [Q_0]_\sim$ and therefore there is k s.t. $[P_0]_\sim \to_k [Q_0]_\sim$ By definitions of $\to_{k+1}$ and $\to$, $[P_0 \mid P_1]_\sim \to_{k+1} [Q_0 \mid P_1]_\sim$ and $[P_0 \mid P_1]_\sim \to [Q_0 \mid P_1]_\sim$. Therefore if we define $[Q]_\sim = [Q_0 \parallel P_1]_\sim$ the proof is complete. □

Lemma 12. $\preccurlyeq \subseteq \mathbf{P}_\sim \times \mathbf{P}_\sim$ *is a decomposition order.*

Proof.

1. $\preccurlyeq$ is well-founded. We have to prove that every non-empty subset of $\mathbf{P}_\sim$ has a $\preccurlyeq$-minimal element. Let $X \subseteq \mathbf{P}_\sim$ with $X \neq \emptyset$. Let $[P]_\sim$ be s.t. $[P]_\sim \in X$ and $depth([P]_\sim) = \min\{depth([Q]_\sim) \mid [Q]_\sim \in X\}$, then $[P]_\sim$ is a minimal element of X by Lemma 9 and definition of $\preccurlyeq$.
2. $[\mathbf{0}]_\sim$ is the least element of $\mathbf{P}_\sim$ w.r.t. $\preccurlyeq$. We consider $[P]_\sim$ and proceed by induction on $depth(P)$. If $depth(P) = 0$, then $P \sim \mathbf{0}$ and therefore $[P]_\sim = [\mathbf{0}]_\sim$. Suppose that $depth(P) = n > 0$. By Lemma 11 there is Q s.t. $[P]_\sim \to [Q]_\sim$. By Lemma 9, $depth(Q) < depth(P)$. By induction and definition of $\preccurlyeq$, $[\mathbf{0}]_\sim \preccurlyeq [Q]_\sim \preccurlyeq [P]_\sim$.
3. $\preccurlyeq$ is strictly compatible. Suppose $[Q]_\sim \prec [P]_\sim$ and consider $[P]_\sim \parallel [S]_\sim$. By definition of $\prec$ there are $P_0, \ldots, P_n \in \Pi_f$, with $n > 0$, s.t. $[P]_\sim = [P_0]_\sim \to [P_1]_\sim \to \ldots \to [P_n]_\sim = [Q]_\sim$. By definition of $\to$, for each $i = 0, \ldots n-1$ there is k_i s.t. $[P_i]_\sim \to_{k_i} [Q_i]_\sim$. Define $k = \max\{k_i : i = 0, \ldots n-1\}$. Then

$$[P]_\sim \parallel [S]_\sim = [P_0]_\sim \parallel [S]_\sim = [P_0 \mid S]_\sim \to_{k+1} \ldots \to_{k+1} [P_n \mid S]_\sim = [P_n]_\sim \parallel [S]_\sim = [Q]_\sim \parallel [S]_\sim$$

   By definition of $\to$,

$$[P]_\sim \parallel [S]_\sim = [P_0]_\sim \parallel [S]_\sim = [P_0 \mid S]_\sim \to \ldots \to [P_n \mid S]_\sim = [P_n]_\sim \parallel [S]_\sim = [Q]_\sim \parallel [S]_\sim$$

   By Lemma 9 and $n > 0$, $depth([P]_\sim \parallel [S]_\sim) > depth([Q]_\sim \parallel [S]_\sim)$. Then $[Q]_\sim \parallel [S]_\sim \prec [P]_\sim \parallel [S]_\sim$.
4. $\preccurlyeq$ is precompositional. Suppose $[P]_\sim \preccurlyeq [Q]_\sim \parallel [R]_\sim$, we have to prove there are $[Q']_\sim \preccurlyeq [Q]_\sim$ and $[R']_\sim \preccurlyeq [R]_\sim$ s.t. $[P]_\sim = [Q']_\sim \parallel [R']_\sim$. If $Q \sim R \sim \mathbf{0}$ then $Q' \sim R' \sim \mathbf{0}$ and the conditions are satisfied. Suppose that only one of both processes is bisimilar to $\mathbf{0}$. W.l.o.g. suppose $Q \not\sim \mathbf{0}$ and $R \sim \mathbf{0}$. In this case, $[P]_\sim \preccurlyeq [Q]_\sim \parallel [R]_\sim = [Q]_\sim$, then if we define $[Q']_\sim = [P]_\sim$ and $[R']_\sim = [\mathbf{0}]_\sim$, the conditions are also satisfied. Suppose now that $Q \not\sim \mathbf{0}$ and $R \not\sim \mathbf{0}$. By definition of $\preccurlyeq$ there are $n \geq 0$ and

processes $S_n,\ldots,S_0$ s.t. $[Q]_\sim \parallel [R]_\sim = [Q \mid R]_\sim = [S_n]_\sim \to \ldots \to [S_0]_\sim = [P]_\sim$. The proof proceed by induction on n. Suppose that the hypothesis holds for n, we prove the case $n+1$. Given that $[S_{n+1}]_\sim = [Q \mid R]_\sim = [Q]_\sim \parallel [R]_\sim \to [S_n]_\sim$, by definition of $\to$, there is T s.t. either $[Q]_\sim \to [T]_\sim$ and $[S_n]_\sim = [T \mid R]_\sim$ or, $[R]_\sim \to [T]_\sim$ and $[S_n]_\sim = [Q \mid T]_\sim$. (We have omitted the sub-index of $\to$ because it does not play any role.) W.l.o.g. suppose that $[Q]_\sim \to [T]_\sim$ and $[S_n]_\sim = [T \mid R]_\sim$. Then $[P]_\sim \preccurlyeq [T \mid R]_\sim = [T]_\sim \parallel [R]_\sim$. By induction there are $[T']_\sim$ and $[R']_\sim$ s.t. $[T']_\sim \preccurlyeq [T]_\sim$, $[R']_\sim \preccurlyeq [R]_\sim$ and $[P]_\sim = [T']_\sim \parallel [R']_\sim$. Because $[T]_\sim \preccurlyeq [Q]_\sim$ and $\preccurlyeq$ is a partial order, we have that $[T']_\sim \preccurlyeq [Q]_\sim$ and we can conclude the proof.

5. $\preccurlyeq$ is Archimedean. Suppose that $[P]_\sim, [Q]_\sim \in \mathbf{P}_\sim$ are s.t. $[P]_\sim^n \preccurlyeq [Q]_\sim$ for all $n \in \mathbb{N}_0$. By Lemma 6, $depth(P^n) = n \cdot depth(P)$. Given that $depth(Q) \in \mathbb{N}_0$ we can conclude that $depth(P) = 0$ and therefore $[P]_\sim = [\mathbf{0}]_\sim$.

□

By Theorem 1, it follows that $\mathbf{P}_\sim$ has unique decomposition.

Corollary 1. *The commutative monoid $\mathbf{P}_\sim$ has unique decomposition.*

5 Unique parallel decomposition with respect to weak bisimilarity

To prove the result of unique parallel decomposition w.r.t. strong bisimilarity, we relied on the definition of depth and on the properties that are satisfied when we take into account strong bisimilarity. In particular, we proved that all strongly bisimilar processes have the same depth. For the weak bisimilarity we do not have the same property. Consider the following processes

$$P = \overline{x}y.\mathbf{0} \qquad P' = \tau.\overline{x}y.\mathbf{0} \qquad P'' = \tau.\tau.\overline{x}y.\mathbf{0}$$

Notice that $P \approx P' \approx P''$, despite this, $depth(P) < depth(P') < depth(P'')$. To avoid this problem and to adapt the ideas behind results in the previous section, we will consider processes without *stuttering transitions*. A transition $P \xrightarrow{\alpha} P'$ is a stuttering transition if $\alpha = \tau$ and $P \approx P'$.

We could not establish UPD for normed processes in the strong setting, because the norm of the arguments of a parallel composition is not necessarily less than the norm of the parallel composition. In the weak setting, it is known that normed processes of Π do not satisfy UPD w.r.t. bisimilarity. Consider the following counter example [7]: define $P = \nu z(\overline{z}c \mid z(x).!\overline{a}b \mid z(y))$. P is normed because $P \xrightarrow{\tau} \nu z(\mathbf{0} \mid z(x).!\overline{a}b \mid \mathbf{0}) \downarrow$ but there is no a unique parallel decomposition of P because $P \approx P \mid P$.

We study processes without stuttering transitions in Section 5.1. Using the results developed in that section and Theorem 1, in Section 5.2 we prove that for finite processes there is a unique parallel decomposition w.r.t. weak bisimilarity.

5.1 Processes without stuttering steps

For $\omega = a_1 a_2 \cdots a_n \in A_\tau^*$ with $n > 0$, we write $P \overset{\omega}{\Longrightarrow} P'$ if there are processes $P_0, P_1, \ldots, P_n$ s.t. $P = P_0 \overset{a_1}{\Longrightarrow} P_1 \overset{a_2}{\Longrightarrow} \ldots \overset{a_n}{\Longrightarrow} P_n = P'$. If $\omega = \varepsilon$, then $P \overset{\omega}{\Longrightarrow} P'$ implies $P \Longrightarrow P'$.

Definition 11. *A process $P \in \Pi_f$ is a process* without stuttering transitions *if there are no $\omega \in A^*$ and $P', P'' \in \Pi_f$ s.t. $P \overset{\omega}{\Longrightarrow} P' \xrightarrow{\tau} P''$ and $P' \approx P''$. We denote with $\Pi_{\not\circlearrowleft}$ the set of processes of Π_f without stuttering transitions.*

In Section 4.1 we discussed why we do not consider infinite processes, this discussion also applies for weak bisimilarity. By definition, $\Pi_{\not\oslash} \subseteq \Pi_f$. This fact and Lemma 14 ensure that we can use processes in $\Pi_{\not\oslash}$ to define properties over equivalence classes of processes in Π_f w.r.t. weak bisimilarity.

To prove Lemma 14 we need to introduce some notation and Lemma 13. We write $\overline{x}(z).P$ to denote $\nu z.\overline{x}z.P$. We call $\overline{x}(z)$ a *bound-output prefix*. We use λ to range over prefixes, including bound-outputs.

Lemma 13. *For all $P \in \Pi_f$ there are prefixes $\lambda_1,\ldots,\lambda_n$ and processes $P_1,\ldots,P_n$ such that $P \sim \sum_{i=1}^{n} \lambda_i.P_i$.*

Proof. The proof proceeds by structural induction on P. Cases $P = \mathbf{0}$, $P = \pi.P'$ are straightforward by definition. For the case $P = P_1 + P_2$, by induction hypothesis there are processes $Q_1 = \sum_{i\in I} \lambda_i.P_i$ and $Q_2 = \sum_{j\in J} \lambda_j.P_j$ and bisimulations R_1 and R_2 s.t. $P_k\, R_k\, Q_k$ for $k = 1,2$. It is easy to see that $\{(P, Q_1+Q_2)\} \cup R_1 \cup R_2$ is a bisimulation. Case $P = P_1 \mid P_2$ is straightforward by induction and the Expansion Lemma for $\sim$ in the π-calculus [20, Lemma 2.2.14]. Thanks this lemma we can state that for all $P = \sum_{i\in I} \lambda_i P_i$ and $Q = \sum_{j\in J} \lambda_j Q_j$ there is $R = \sum_{k\in K} \lambda_k R_k$ s.t. $P \mid Q \sim R$. Case $P = \nu z.P'$ proceeds by structural induction on P'. If $P' = \mathbf{0}$ then $\nu z.P' \approx \mathbf{0} = \sum_{i=1}^{0} \lambda_i.P_i$. If $P' = \pi.P''$ then there are three cases to analyse: (i) $z \notin n(\pi)$ then $P \sim \pi.(\nu z.P'')$, (ii) $\pi = \overline{x}z$ then $\nu z.P'$ can be denoted by $\overline{x}(z)P''$, (iii) $z \in n(\pi)$ and $\pi \neq \overline{x}z$ then $\nu z.\pi.P'' \sim \mathbf{0}$. If $P' = P_0 + P_1$ then notice that $\nu z(P_0+P_1) \sim \nu z.P_0 + \nu z.P_1$; by induction there are processes $\sum_{i\in I} \lambda_i P_i$ and $\sum_{j\in J} \lambda_j P_j$ s.t. $\nu z.P_0 \sim \sum_{i\in I} \lambda_i P_i$ and $\nu z.P_1 \sim \sum_{j\in J} \lambda_j P_j$. From this point, the proof follows as in the case $P = P_1 + P_2$. Finally, case $P' = P_0 \mid P_1$ can be reduced to the previous case using the Expansion Lemma for $\sim$ ([20, Lemma 2.2.14]). □

Lemma 14. *For every process $P \in \Pi_f$ there is $Q \in \Pi_{\not\oslash}$ s.t. $P \approx Q$.*

Proof. The proof of the result follows by complete induction on $n = depth(P)$. By Lemma 13 for $P \in \Pi_f$ there are prefixes $\lambda_1,\ldots,\lambda_n$ and processes $P_1,\ldots,P_n$ such that $P \sim \sum_{i=1}^{n} \lambda_i.P_i$. By induction and Lemma 2, for each i there is $Q_i \in \Pi_{\not\oslash}$ s.t. $P_i \approx Q_i$. Then if we define

$$Q = \textstyle\sum_{i\in\{1,\ldots n\} \text{ and } P\not\approx P_i} \lambda_i.Q_i$$

we get Q s.t. $Q \in \Pi_{\not\oslash}$ and $P \approx Q$. □

We cannot restrict our attention only to processes in $\Pi_{\not\oslash}$ because the property of not executing stuttering transitions is not preserved by parallel composition. Consider the processes $P_0 = \nu z(\overline{a}z)$ and $P_1 = a(x).(\overline{x}b + \tau.\overline{c}b)$. Both $P_0, P_1 \in \Pi_{\not\oslash}$ but $P_0 \parallel P_1 \notin \Pi_{\not\oslash}$ because

$$P_0 \parallel P_1 = \nu z(\overline{a}z) \parallel a(x).(\overline{x}b + \tau.\overline{c}b) \xrightarrow{\tau} \nu z(\mathbf{0} \parallel (\overline{z}b + \tau.\overline{c}b)) \sim \tau.\overline{c}b \approx \overline{c}b$$

If we compare this fact with the strong setting, we can say that it is not possible to prove a lemma similar to Lemma 7 for processes in $\Pi_{\not\oslash}$.

As in Section 4.1, we conclude with a collection of theorems and lemmas. Theorems 4 and 5 are equivalent, respectively, to Theorems 2 and 3 but w.r.t. processes in $\Pi_{\not\oslash}$ and weak bisimilarity. Most of the lemmas are needed to prove these results and only a few of them are used in the next section.

Lemma 15. *If $P \in \Pi_{\not\oslash}$ and $P \stackrel{\omega}{\Longrightarrow} Q$ for $\omega \in A_\tau^*$ then $Q \in \Pi_{\not\oslash}$.*

Proof. Suppose $Q \notin \Pi_{\not\oslash}$, then there are $\omega' \in A^*$ and $Q', Q'' \in \Pi_f$ s.t. $Q \stackrel{\omega'}{\Longrightarrow} Q' \xrightarrow{\tau} Q''$ with $Q' \approx Q''$. Let $\tilde{\omega}$ obtained from ω by removing τ's actions. Then $P \stackrel{\tilde{\omega}\omega'}{\Longrightarrow} Q' \xrightarrow{\tau} Q''$ and therefore $P \notin \Pi_{\not\oslash}$, which is a contradiction. □

Lemma 16. *If $P,Q \in \Pi_{\not\emptyset}$ are s.t. $P \approx Q$ and $P \xrightarrow{\alpha} P'$ with $\alpha \in A_\tau$, then $Q \stackrel{\alpha}{\Longrightarrow} Q'$, i.e. Q executes at least a transition, and $P' \approx Q'$*

Proof. If $\alpha \neq \tau$ the result is straightforward by Def. 7. If $\alpha = \tau$ and there is no transition $Q_0 \xrightarrow{\tau} Q_1$ s.t. $Q \Longrightarrow Q_0 \xrightarrow{\tau} Q_1 \Longrightarrow Q'$ and $P' \approx Q'$ then $P' \approx Q$ since $P \approx Q$. This implies that $P \approx Q \approx P'$, i.e. $P \xrightarrow{\alpha} P'$ is a stuttering transition. This contradicts $P \in \Pi_{\not\emptyset}$. □

Theorem 4. *If $P,Q \in \Pi_{\not\emptyset}$ and $P \approx Q$ then $depth(P) = depth(Q)$.*

Proof. We proceed by complete induction over $n = depth(P)$. If $depth(P) = 0$, then $P \sim \mathbf{0}$ and moreover $P \approx \mathbf{0}$. Because $P \approx Q$ and $P \sim \mathbf{0}$, there is no $\alpha \in A$ s.t. $Q \stackrel{\alpha}{\Longrightarrow}$. Taking into account this fact, if there is Q' s.t. $Q \xrightarrow{\tau} Q'$, Q' is such that $Q' \approx \mathbf{0}$. This creates a contradiction because $Q \xrightarrow{\tau} Q'$ is a stuttering transition and $Q \in \Pi_{\not\emptyset}$. Then $Q \sim \mathbf{0}$ and therefore $depth(Q) = 0 = depth(P)$. Suppose $depth(P) = n+1$. Let $\omega = \alpha\omega' \in A_\tau^*$ and P' be s.t. $length(\omega) = n+1$ and $P \xrightarrow{\alpha} P' \xrightarrow{\omega'}$. Because $P \approx Q$ and Lemma 16 there are Q_0, Q_1, Q' s.t. $Q \Longrightarrow Q_0 \xrightarrow{\alpha} Q_1 \Longrightarrow Q'$ and $P' \approx Q'$. By induction $depth(P') = depth(Q')$ and therefore $depth(Q) \geq depth(P) = n+1$. We prove now that when we assume $depth(Q) > n+1$ we reach a contradiction; it then follows that $depth(Q) = n+1$. Assume $depth(Q) > n+1$ and let $\omega = \alpha\omega' \in A_\tau^*$ be such that $Q \xrightarrow{\alpha} \tilde{Q} \xrightarrow{\omega'} \tilde{Q}' \downarrow$ and $length(\omega) = depth(Q)$. Because $P \approx Q$ and Lemma 16 there is $\tilde{P}$ s.t. $P \stackrel{\alpha}{\Longrightarrow} \tilde{P}$, $\tilde{P} \approx \tilde{Q}$ and $depth(P) = n+1 > depth(\tilde{P})$. By the complete induction $depth(\tilde{P}) = depth(\tilde{Q})$. Then, we reach a contradiction, $n+1 > depth(\tilde{P}) = depth(\tilde{Q}) \geq n+1$. □

Lemma 17. *If $P \in \Pi_f$ and $P \xrightarrow{\alpha} P'$ with $\alpha \neq \tau$ then $P \not\approx P'$.*

Proof. Let $\omega \in A^*$ be the largest sequence s.t. $P' \stackrel{\omega}{\Longrightarrow} P'' \downarrow$. Then there is no Q s.t. $P' \stackrel{\alpha\omega}{\Longrightarrow} Q$. On the other hand $P \stackrel{\alpha\omega}{\Longrightarrow}$, therefore $P \not\approx P'$. □

Lemma 18. *If $P \in \Pi_{\not\emptyset}$ is s.t. $P \not\sim \mathbf{0}$ and for all $P' \in \Pi_f, \alpha \in A_\tau$, $P \xrightarrow{\alpha} P' \downarrow$, then there is $\alpha' \neq \tau$ s.t. $P \xrightarrow{\alpha'}$.*

Proof. Because $P \not\sim \mathbf{0}$ there is α s.t. $P \xrightarrow{\alpha}$. If for all $P' \in \Pi_f, \alpha \in A_\tau$, $P \xrightarrow{\alpha} P' \downarrow$ and $\alpha = \tau$ then $P \approx \mathbf{0}$ and therefore all transitions that can be executed by P are stuttering transitions. This contradicts $P \in \Pi_{\not\emptyset}$. □

Theorem 5. *If $P,Q,R \in \Pi_{\not\emptyset}$, $P \not\approx \mathbf{0}$, $Q \not\approx \mathbf{0}$ and $P \parallel Q \approx R$ then $depth(P) < depth(R)$ and $depth(Q) < depth(R)$.*

Proof. We prove $depth(P) < depth(R)$, the proof that $depth(Q) < depth(R)$ is analogous. Note that, since $Q \not\approx \mathbf{0}$, there is Q' with $depth(Q') = 1$ that is reachable from Q, that is, there exists $\omega \in A^*$ s.t. $Q \stackrel{\omega}{\Longrightarrow} Q'$ and $Q' \not\sim \mathbf{0}$ (we remark the symbol $\not\sim$) and for all $Q'' \in \Pi_f, \alpha \in A_\tau$, $Q' \xrightarrow{\alpha} Q'' \downarrow$. Then, by Lemma 15, $Q' \in \Pi_{\not\emptyset}$ and, by Lemma 18, $Q' \xrightarrow{\alpha}$ with $\alpha \neq \tau$. Furthermore, by Convention 1 and the symmetric version of rule (Par-L) we have that $P \parallel Q \stackrel{\omega}{\Longrightarrow} P \parallel Q'$. By Lemma 17, $P \parallel Q' \xrightarrow{\alpha} P \parallel \mathbf{0}$ and $\alpha \neq \tau$ imply $P \parallel Q' \not\approx P \parallel \mathbf{0} \approx P$. Because $P \in \Pi_{\not\emptyset}$, whenever $S \in \Pi_{\not\emptyset}$ and $S \approx P \parallel Q'$, $depth(S) \geq 1 + depth(P)$ $(\star)$. Given that $R \approx P \parallel Q$, $P \parallel Q \stackrel{\omega}{\Longrightarrow} P \parallel Q'$ implies there is R' s.t. $R \stackrel{\omega}{\Longrightarrow} R'$ and $R' \approx P \parallel Q'$. By Lemma 15, $R' \in \Pi_{\not\emptyset}$. In addition, by $(\star)$, $depth(R') \geq 1 + depth(P)$. Finally $depth(R) \geq depth(R') \geq 1 + depth(P) > depth(P)$. □

Lemma 19. *If $P,Q \in \Pi_{\not\emptyset}$, $P \approx Q$ and $P \stackrel{\alpha}{\Longrightarrow} P'$, with $\alpha \in A_\tau$, then there is Q' s.t. $Q \stackrel{\alpha}{\Longrightarrow} Q'$.*

5.2 Unique parallel decomposition

The development in this section is similar to the development in Section 4.2, for this reason in some cases we use the same notation. This will not be a problem because both developments are independent. In order to use Theorem 1 we need to define a commutative monoid with a decomposition order. The commutative monoid is defined by

- $\mathbf{P}_{\approx} = \{[P]_{\approx} : P \in \Pi_f\}$ where $[P]_{\approx} = \{P' : P' \approx P\}$
- $e = [\mathbf{0}]_{\approx} \in \mathbf{P}_{\approx}$.
- $\| \subseteq \mathbf{P}_{\approx} \times \mathbf{P}_{\approx}$ is s.t. $[P]_{\approx} \| [Q]_{\approx} = [P \mid Q]_{\approx}$

Notice that we cannot ensure that for all $P', P'' \in [P]_{\approx}$, $depth(P') = depth(P'')$. Then, we extend the notion of depth in the following way. Define $[P]_{\approx}^{\not\emptyset} = [P]_{\approx} \cap \Pi_{\not\emptyset}$. For $[P]_{\approx} \in \mathbf{P}_{\approx}$, $depth([P]_{\approx}) = depth(P')$ with $P' \in [P]_{\approx}^{\not\emptyset}$. This definition is sound because of Lemma 14 and Theorem 4.

Lemma 20. *$\mathbf{P}_{\approx}$ with neutral element $[\mathbf{0}]_{\approx}$ and binary operation $\|$ is a commutative monoid. I.e., $\| \subseteq \mathbf{P}_{\approx} \times \mathbf{P}_{\approx}$ satisfies the associativity, commutativity and identity properties.*

We shall define the partial order $\preccurlyeq$ over $\mathbf{P}_{\approx}$ using the relation $\rightarrow \subseteq \mathbf{P}_{\approx} \times \mathbf{P}_{\approx}$ defined as follows:

$$\begin{aligned}
\rightarrow_0 = \{([P]_{\approx}, [Q]_{\approx}) : \exists P' \in [P]_{\approx}^{\not\emptyset}, Q' \in [Q]_{\approx}^{\not\emptyset} : P' \stackrel{\alpha}{\Longrightarrow} Q', \alpha \in A_\tau \\
\text{and } \nexists P_0, P_1 \in \Pi_{\not\emptyset} \text{ s.t. } P_0 \not\approx \mathbf{0}, P_1 \not\approx \mathbf{0}, P_0 \mid P_1 \approx P\}
\end{aligned}$$

$$\begin{aligned}
\rightarrow_{k+1} = \{([P_0 \mid P_1]_{\approx}, [Q_0 \mid P_1]_{\approx}) : [P_0]_{\approx} \rightarrow_k [Q_0]_{\approx}, P_1 \in \Pi_f\} \\
\cup \{([P_0 \mid P_1]_{\approx}, [P_0 \mid Q_1]_{\approx}) : [P_1]_{\approx} \rightarrow_k [Q_1]_{\approx}, P_0 \in \Pi_f\}
\end{aligned}$$

$$\rightarrow = \bigcup_{k=0}^{\infty} \rightarrow_k$$

The partial order $\preccurlyeq$ is defined as the inverse of the reflexive-transitive closure of $\rightarrow$ i.e., $\preccurlyeq = (\rightarrow^*)^{-1}$. We write $[P]_{\approx} \prec [Q]_{\approx}$ if $[P]_{\approx} \preccurlyeq [Q]_{\approx}$ and $[P]_{\approx} \neq [Q]_{\approx}$.

Notice that in this case $\rightarrow$ takes into account processes in $\Pi_{\not\emptyset}$ and weak transitions that execute at least one transition. Also notice that we are avoiding communications between the arguments of the parallel composition in order to avoid scope extrusion.

Similarly to the strong setting, we need two lemmas, Lemmas 21 and 22, to prove that $\preccurlyeq$ is a partial order (Lemma 23). In addition, to prove that $\preccurlyeq$ is a decomposition order, we need the Lemma 24 that is equivalent to Lemma 11. The proofs of these results follow similarly to their respective counterpart in the strong setting. (The complete proofs are in the appendix.)

Lemma 21. *If $[P]_{\approx} \rightarrow [Q]_{\approx}$ then for all $\tilde{P} \in [P]_{\approx}$ there are $\alpha \in A_\tau$ and $\tilde{Q} \in [Q]_{\approx}$ s.t. $\tilde{P} \stackrel{\alpha}{\Longrightarrow} \tilde{Q}$.*

Lemma 22. *If $[P]_{\approx} \rightarrow [Q]_{\approx}$ then $depth([P]_{\approx}) > depth([Q]_{\approx})$.*

Lemma 23. *$\preccurlyeq$ is a partial order.*

Lemma 24. *If $P \in \Pi_{\not\emptyset}$ and $depth(P) > 0$ then there is Q s.t. $[P]_{\approx} \rightarrow [Q]_{\approx}$.*

We are ready to prove that $\preccurlyeq \subseteq \mathbf{P}_{\approx} \times \mathbf{P}_{\approx}$ is a decomposition order. This proof does not present changes w.r.t. proof of Lemma 12 except that for proving $\preccurlyeq$ is Archimedean, we use Theorem 5. Notice that there is no lemma equivalent to Lemma 6 in the weak setting.

Lemma 25. *$\preccurlyeq \subseteq \mathbf{P}_{\approx} \times \mathbf{P}_{\approx}$ is a decomposition order.*

By Theorem 1, it follows that $\mathbf{P}_{\approx}$ has unique decomposition.

Corollary 2. *The commutative monoid $\mathbf{P}_{\approx}$ has unique decomposition.*

6 Final Remarks

In this paper we have proved that finite processes of the π-calculus satisfy UPD w.r.t. both strong bisimilarity and weak bisimilarity. We have obtained these results using the technique presented in [15] (see Theorem 1) and different properties that are satisfied in each setting. For the strong setting, we had to prove properties related to the depth of processes. For the weak setting, we had to prove properties related to processes that execute no stuttering transitions. Our results show that the abstract framework of [15] can be used in the context of the π-calculus, dealing with the complications that arise from scope extrusion. In addition, the same framework can be used to deal with the weak setting if one considers processes without stuttering transitions. In this way, we have avoided the abstract technique introduced in [14] which is considerably more involved than the technique that we have used in this paper.

In Section 4 we showed with two examples that norm is not additive for the π-calculus and therefore some proofs in [7] are flawed. After pointing out this problem to Dreier et al., they proposed us an alternative definition of norm to solve it. Call this variant $norm'$. Roughly, $norm'$ should not consider traces where there is a scope extrusion of processes. We think this solution may work for the applied π-calculus, but is not suitable for the variant of the π-calculus considered in the present paper. We first explain what is the problem in our context, and then why this problem is not present in applied π. In the first example in Section 4, we had $P = P_0 \mid P_1 = \nu z(\overline{a}z) \mid a(x).!\overline{x}a$. Process P is not normed, i.e. $norm'(P) = \infty$, because the only finite trace that the process executes, $P \xrightarrow{\tau} \nu z(\mathbf{0} \mid !\overline{z}a) \downarrow$, goes through a scope extrusion. Now, consider the process

$$P' = \nu z(\overline{a}z).(\mathbf{0} \mid a(x).!\overline{x}a) \;+\; a(x).(\nu z(\overline{a}z) \mid !\overline{x}a) \;+\; \tau.(\nu z(\mathbf{0} \mid !\overline{z}a)) \;\;;$$

it would be normed according to the alternative definition suggested above (the τ-transition from P' is not the result of a scope extrusion). Now, since P' is just the expansion of P, it is clear that $P \sim P'$. Thus, we find that the property of being normed is not compatible with bisimilarity. Since the applied π-calculus does not include the construct for non-deterministic choice needed for the expansion, this problem is not present there.

An open question that leaves this paper is related with UPD of the π-calculus w.r.t. *strong full bisimilarity*[20]. Strong full bisimilarity is a stronger notion of bisimulation that is a congruence for all constructs of the π-calculus. We have tried to apply the abstract technique in this setting so far without success. When we tried to repeat the result in Section 4, taking into account the universal quantification in the definition of strong full bisimilarity, a problem arose when we wanted to prove that the order is a decomposition order. Particularly, we were not able to prove that the order is strict compatible. Notice that this problem is not present in the *asynchronous π-calculus*[20], a well-known fragment of the π-calculus, because (strong) bisimilarity and (strong) full bisimilarity coincide.

Acknowledgement. The authors thank Daniel Hirschkoff for discussions, comments and suggestions on various drafts of this paper, and anonymous reviewers for their thorough reviews and good suggestions.

References

[1] L. Aceto, W. Fokkink, A. Ingólfsdóttir & B. Luttik (2005): *CCS with Hennessy's merge has no finite-equational axiomatization*. *Theor. Comput. Sci.* 330(3), pp. 377–405, doi:10.1016/j.tcs.2004.10.003.

[2] L. Aceto, W. Fokkink, A. Ingólfsdóttir & B.. Luttik (2009): *A finite equational base for CCS with left merge and communication merge*. *ACM Trans. Comput. Log.* 10(1), doi:10.1145/1459010.1459016.

[3] L. Aceto, A. Ingólfsdóttir, B. Luttik & P. van Tilburg (2008): *Finite Equational Bases for Fragments of CCS with Restriction and Relabelling*. In: *IFIP 20th World Computer Congress, TC 1, Foundations of Computer Science*, pp. 317–332, doi:10.1007/978-0-387-09680-3_22.

[4] I. Castellani (1998): *Bisimulations for Concurrency*. Ph.D. thesis, University of Edinburgh. Also published as LFCS-88-51.

[5] S. Christensen (1993): *Decidability and Decomposition in Process Algebra*. Ph.D. thesis, University of Edinburgh.

[6] F. Corradini, R. Gorrieri & D. Marchignoli (1998): *Towards parallelization of concurrent systems*. *Informatique théorique et applications* 32(4-6), pp. 99–125.

[7] J. Dreier, C. Ene, P. Lafourcade & Y. Lakhnech (2016): *On the existence and decidability of unique decompositions of processes in the applied π-calculus*. *Theor. Comput. Sci.* 612, pp. 102–125, doi:10.1016/j.tcs.2015.11.033.

[8] J. Dreier, P. Lafourcade & Y. Lakhnech (2012): *Defining Privacy for Weighted Votes, Single and Multi-voter Coercion*. In: *ESORICS 2012*, pp. 451–468, doi:10.1007/978-3-642-33167-1_26.

[9] W. Fokkink & B. Luttik (2000): *An* omega*-Complete Equational Specification of Interleaving*. In: *Automata, Languages and Programming, 27th International Colloquium, ICALP 2000*, pp. 729–743, doi:10.1007/3-540-45022-X_61.

[10] J. Friso Groote & F. Moller (1992): *Verification of Parallel Systems via Decomposition*. In: *CONCUR '92, Third International Conference on Concurrency Theory*, pp. 62–76, doi:10.1007/BFb0084783.

[11] D. Hirschkoff & D. Pous (2008): *A Distribution Law for CCS and a New Congruence Result for the π-calculus*. *Logical Methods in Computer Science* 4(2), doi:10.2168/LMCS-4(2:4)2008.

[12] Y. Hirshfeld & M. Jerrum (1999): *Bisimulation Equivalence Is Decidable for Normed Process Algebra*. In: *Automata, Languages and Programming, 26th International Colloquium, ICALP'99*, pp. 412–421, doi:10.1007/3-540-48523-6_38.

[13] I. Lanese, J. A. Pérez, D. Sangiorgi & A. Schmitt (2011): *On the expressiveness and decidability of higher-order process calculi*. *Inf. Comput.* 209(2), pp. 198–226, doi:10.1016/j.ic.2010.10.001.

[14] B. 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.

[15] B. Luttik & V. van Oostrom (2005): *Decomposition orders another generalisation of the fundamental theorem of arithmetic*. *Theor. Comput. Sci.* 335(2-3), pp. 147–186, doi:10.1016/j.tcs.2004.11.019.

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

[17] F. Moller (1989): *Axioms for Concurrency*. Ph.D. thesis, University of Edinburgh.

[18] F. Moller (1990): *The Importance of the Left Merge Operator in Process Algebras*. In: *Automata, Languages and Programming, 17th International Colloquium, ICALP90*, pp. 752–764, doi:10.1007/BFb0032072.

[19] F. Moller (1990): *The Nonexistence of Finite Axiomatisations for CCS Congruences*. In: *Proceedings of LICS '90*, pp. 142–153, doi:10.1109/LICS.1990.113741.

[20] D. Sangiorgi & D. Walker (2001): *π-Calculus: A Theory of Mobile Processes*. Cambridge University Press, New York, NY, USA.

Reversible Multiparty Sessions with Checkpoints*

Mariangiola Dezani-Ciancaglini † Paola Giannini‡

Reversible interactions model different scenarios, like biochemical systems and human as well as automatic negotiations. We abstract interactions via multiparty sessions enriched with named checkpoints. Computations can either go forward or roll back to some checkpoints, where possibly different choices may be taken. In this way communications can be undone and different conversations may be tried. Interactions are typed with global types, which control also rollbacks. Typeability of session participants in agreement with global types ensures session fidelity and progress of reversible communications.

1 Introduction

Reversibility is an essential feature in the construction of reliable systems. If a system reaches an undesired state, some actions may be undone and the computation may be restarted from a consistent state. Several studies, see [6, 15, 16, 20], have investigated the theoretical foundations of reversible computations. The relevance of these papers for our work is briefly discussed at the beginning of Section 5.

Our focus is in the context of structured communications, more precisely multiparty sessions, see [12, 13]. The choreography of communications is described by global types, which are projected on the participants to get their interaction patterns [5, 10]. In order to fix the points of the computations we may revert to, we add checkpoints to the syntax of global types. In contrast to previous work, see [2, 22, 23], our checkpoints are named and rollbacks specify the name of the checkpoints to which we revert.

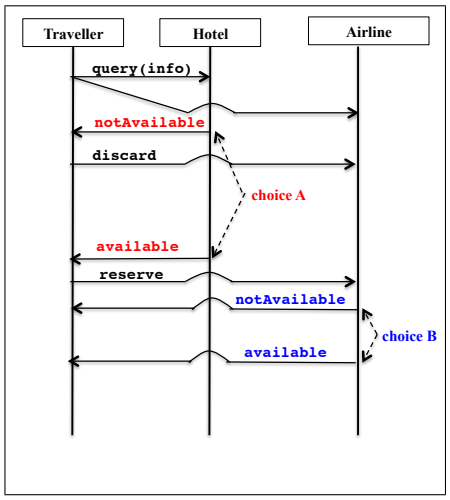


Figure 1: Traveller planning a trip.

We illustrate our approach by discussing an example. Consider the UML sequence diagram of Figure 1. In this example, there are three interacting participants, named Traveller (`Tr`), Hotel (`Ht`) and Airline (`Al`), that establish a session. `Tr`, planning a trip, sends a message labelled `query`, to `Ht` and to `Al` with the details of his journey (abstracted as a string `info`). `Ht` answers to `Tr` with either the message `notAvailable` or `available`. If `Ht` answers `notAvailable`, `Tr` sends to `Al` a message `discard` saying to ignore the previous `query`. If `Ht` answers `available`, then `Tr` send a message to `Al` asking to `reserve` flights for the journey, to which `Al` answers to `Tr` with either the message `notAvailable` or `available`.

The choice made by `Ht`, named A, and the one made by `Al`, named B, are checkpointed choices. This means that the computation could revert to one of these points of the interaction and the given participant could make a different choice. Rolling back to a choice point involves all the participants which crossed this choice point. Moreover, rollback may happen only when all the participants that have this choice point in their future have crossed it. Typing ensures that not involved participants are terminated.

*Partially supported by EU H2020-644235 Rephrase project, EU H2020-644298 HyVar project, ICT COST Actions IC1201 BETTY, IC1402 ARVI and Ateneo/CSP project RunVar.

†Dipartimento di Informatica, Università di Torino, dezani@di.unito.it

‡Computer Science Institute, DiSIT, Università del Piemonte Orientale, paola.giannini@uniupo.it

D. Gebler and K. Peters (Eds.): Combined Workshop on Expressiveness in Concurrency and Structural Operational Semantics (EXPRESS/SOS 2016).
EPTCS 222, 2016, pp. 60–74, doi:10.4204/EPTCS.222.5

Assume that Ht, after sending the message available to Tr, discovers that instead for the required dates it is fully booked, and wants to roll back to the choice point A. Before rolling back, it has to make sure that Tr has sent the reserve message to Al and Al has received the message. So that, they can all go back to the interaction before A. Otherwise, if Tr has not sent the reserve message, and Ht goes back to sending the message notAvailable to Tr which is not expecting a message from Ht, there could be an unpleasant misunderstanding, which is formally represented by a "stuck" computation.

Outline Section 2 introduces our calculus, completed by the type system of Section 3. Subject reduction, session fidelity and progress are proved in Section 4. Section 5 discusses related papers and future work.

2 Calculus

In this section we introduce the syntax and the semantics of multiparty sessions with *named checkpoints*.

Syntax A *multiparty session* is a series of interactions between a fixed number of participants, possibly with branching and recursion [13].

We use the following base sets: *values*, ranged over by $v, v', \ldots$; *expressions*, ranged over by $\mathtt{e}, \mathtt{e}', \ldots$; *expression variables*, ranged over by $x, y, z \ldots$; *labels*, ranged over by $\ell, \ell', \ldots$; *checkpoint names*, ranged over by $A, B, \ldots$; *session participants*, ranged over by $\mathtt{p}, \mathtt{q}, \ldots$; *process variables*, ranged over by $X, Y, \ldots$; *processes*, ranged over by $P, Q, \ldots$; *configurations*, ranged over by $\mathbb{C}, \mathbb{C}', \ldots$; *multiparty sessions*, ranged over by $\mathbb{M}, \mathbb{M}', \ldots$; *networks*, ranged over by $\mathbb{N}, \mathbb{N}', \ldots$.

Our processes are obtained from the processes of [10] by adding named checkpoints before external and internal choices.

Definition 2.1 Processes *are defined by:*

$$P ::= \sum_{i\in I} \mathtt{p}?\ell_i(x_i).P_i \mid \bigoplus_{i\in I} \mathtt{p}!\ell_i(\mathtt{e}_i).P_i \mid {\blacktriangle}_A \sum_{j\in J} \mathtt{p}?\ell_j(x_j).P_j \mid {\blacktriangle}_A \bigoplus_{j\in J} \mathtt{p}!\ell_j(\mathtt{e}_j).P_j \mid \mu X.P \mid X \mid \mathbf{0}$$

We say that ${\blacktriangle}_A P$ is a *process checkpointed by A*.

The input process $\sum_{i\in I} \mathtt{p}?\ell_i(x_i).P_i$ waits for a value and a label ℓ_i with $i \in I$ from participant p and the output process $\bigoplus_{i\in I} \mathtt{q}!\ell_i(\mathtt{e}_i).P_i$ sends the value of an expression $\mathtt{e}_i$ and a label ℓ_i with $i \in I$ to participant q. Checkpointed input and output processes behave in a similar way, except that, when sending/reading a message the checkpointed process is memorised, and can be executed again after a rollback. As usual, in writing processes we omit trailing $\mathbf{0}$'s, and empty parameters.

Example 2.2 *Consider the example of Figure 1. The processes associated with the participants* Tr, Ht, *and* Al *are defined as follows (we abbreviate the labels of messages with their first two consonants):*

$$\begin{array}{lcl} P_{\mathtt{Tr}} & = & \mathtt{Ht!qr(in).Al!qr(in).}P'_{\mathtt{Tr}} \text{ where } P'_{\mathtt{Tr}} = {\blacktriangle}_A \sum\{\mathtt{Ht?nAv.Al!ds}\ ,\ \mathtt{Ht?av.Al!rs.}{\blacktriangle}_B \sum\{\mathtt{Al?nAv}\ ,\ \mathtt{Al?av}\}\} \\ P_{\mathtt{Ht}} & = & \mathtt{Tr?qr}(x).{\blacktriangle}_A \bigoplus\{\mathtt{Tr!nAv}\ ,\ \mathtt{Tr!av}\} \\ P_{\mathtt{Al}} & = & \mathtt{Tr?qr}(x).P'_{\mathtt{Al}} \text{ where } P'_{\mathtt{Al}} = {\blacktriangle}_A \sum\{\mathtt{Tr?ds}\ ,\ \mathtt{Tr?rs.}{\blacktriangle}_B \bigoplus\{\mathtt{Tr!nAv}\ ,\ \mathtt{Tr!av}\}\} \end{array}$$

In order to allow backward reductions, the configurations of session participants contain both active processes and sequences of checkpointed internal and external choices, denoting the processes that should run in case of rollbacks.

Definition 2.3 Configurations*, ranged over by* $\mathbb{C}$*, are pairs* $R \prec P$*, where* P *is a process,* the active process*, and*

$$R ::= \varepsilon \mid R \cdot {\blacktriangle}_A P$$

is a (possibly empty) sequence of checkpointed processes, dubbed checkpointed sequence.

In the sequence $R \cdot P$ we call P the *top process*.

Multiparty sessions, ranged over by $\mathbb{M}$, are parallel compositions of pairs participant/configuration (denoted by $\mathsf{p} \triangleleft \mathbb{C}$):

$$\mathbb{M} ::= \mathsf{p} \triangleleft \mathbb{C} \mid \mathbb{M} \mid \mathbb{M}$$

Networks, ranged over by $\mathbb{N}$, are parallel composition of sessions:

$$\mathbb{N} ::= \mathbb{M} \mid \mathbb{N} \| \mathbb{N}$$

Operational Semantics The LTS of configurations is given in Figure 2. The forward rules are as expected, only internal choices with more than one branch can silently reduce. When the active process crosses a checkpoint, it is memorised at the top of the checkpointed sequence (rules [CkChc] and [CkRcv]). The backward rule can choose as the new active process an arbitrary process P in the current checkpointed sequence: the name of the checkpoint of P decorates the transition (rule [RbP]). This is essential in order to guarantee that the backward reduction of multiparty sessions produces well-behaved sessions, see rule [RbM] in Figure 3.

$$R \prec \bigoplus_{i\in I} \mathsf{p}!\ell_i(\mathsf{e}_i).P_i \xrightarrow{\tau} R \prec \mathsf{p}!\ell_k(\mathsf{e}_k).P_k \quad k \in I \neq \{k\} \quad [\text{Chc}]$$

$$R \prec {}_{\blacktriangle_A}\bigoplus_{j\in J} \mathsf{p}!\ell_j(\mathsf{e}_j).P_j \xrightarrow{\tau} R \cdot {}_{\blacktriangle_A}\bigoplus_{j\in J} \mathsf{p}!\ell_j(x_j).P_j \prec \mathsf{p}!\ell_k(\mathsf{e}_k).P_k \quad k \in J \neq \{k\} \quad [\text{CkChc}]$$

$$R \prec \mathsf{p}!\ell(\mathsf{e}).P \xrightarrow{\mathsf{p}!\ell(v)} R \prec P \quad \mathsf{e} \downarrow v \quad [\text{Snd}]$$

$$R \prec \sum_{i\in I} \mathsf{p}?\ell_i(x_i).P_i \xrightarrow{\mathsf{p}?\ell_j(v)} R \prec P_j\{v/x\} \quad j \in I \quad [\text{Rcv}]$$

$$R \prec {}_{\blacktriangle_A}\sum_{j\in J} \mathsf{p}?\ell_j(x_j).P_j \xrightarrow{\mathsf{p}?\ell_k(v)} R \cdot {}_{\blacktriangle_A}\sum_{j\in J} \mathsf{p}?\ell_j(x_j).P_j \prec P_k\{v/x\} \quad k \in J \quad [\text{CkRcv}]$$

$$R \cdot {}_{\blacktriangle_A}P \cdot R' \prec P' \xrightarrow{A} R \prec {}_{\blacktriangle_A}P \quad [\text{RbP}]$$

Figure 2: Reduction rules of configurations.

The operational semantics of sessions and network is shown in Figure 3, where α ranges over $\tau, \mathsf{p}!\ell(v), \mathsf{p}?\ell(v), A$. This semantics relies on a structural equivalence $\equiv$ for which the parallel operators $\mid$ and $\|$ are commutative and associative and have $\mathsf{p} \triangleleft \varepsilon \prec \mathbf{0}$ as neutral element.

The only interesting rule is rule [RbM]. In this rule we use the mapping $\mathscr{A}$, that associates to a configuration the set of the checkpoint names of processes belonging to its checkpointed sequence. Formally:

$$\mathscr{A}(R \prec \mathbf{0}) = \mathscr{A}(R) \qquad \mathscr{A}(\varepsilon) = \emptyset \qquad \mathscr{A}(R \cdot {}_{\blacktriangle_A}P) = \mathscr{A}(R) \cup \{A\}$$

This mapping is defined only for configurations having $\mathbf{0}$ as their active process. This is enough since the typing rules ensure that the processes which did not traverse some checkpoints are terminated. A multiparty session can roll back to processes at the checkpoint named A only if all the sets $\mathscr{A}$ of the configurations which remain unchanged are defined (i.e. $\mathbf{0}$ is the active process of these configurations) and they do not contain A.

In networks the different sessions reduce independently. For this reason the same participant can interact in different sessions belonging to the same network. We use $\xrightarrow{\tau}{}^*$ to denote the transitive and reflexive closure of the $\xrightarrow{\tau}$ relation.

$$\frac{\mathbb{C} \xrightarrow{\alpha} \mathbb{C}'}{\mathsf{p} \triangleleft \mathbb{C} \xrightarrow{\alpha} \mathsf{p} \triangleleft \mathbb{C}'}\ [\text{PC}] \qquad \frac{\mathbb{M} \xrightarrow{\tau} \mathbb{M}'}{\mathbb{M} \mid \mathbb{M}'' \xrightarrow{\tau} \mathbb{M}' \mid \mathbb{M}''}\ [\text{PRM}] \qquad \frac{\mathbb{M}_1 \equiv \mathbb{M}_1' \quad \mathbb{M}_1' \xrightarrow{\tau} \mathbb{M}_2' \quad \mathbb{M}_2' \equiv \mathbb{M}_2}{\mathbb{M}_1 \xrightarrow{\tau} \mathbb{M}_2}\ [\text{EQM}]$$

$$\frac{\mathsf{p} \triangleleft \mathbb{C}_\mathsf{p} \xrightarrow{\mathsf{q}!\ell(v)} \mathsf{p} \triangleleft \mathbb{C}_\mathsf{p}' \qquad \mathsf{q} \triangleleft \mathbb{C}_\mathsf{q} \xrightarrow{\mathsf{p}?\ell(v)} \mathsf{q} \triangleleft \mathbb{C}_\mathsf{q}'}{\mathsf{p} \triangleleft \mathbb{C}_\mathsf{p} \mid \mathsf{q} \triangleleft \mathbb{C}_\mathsf{q} \xrightarrow{\tau} \mathsf{p} \triangleleft \mathbb{C}_\mathsf{p}' \mid \mathsf{q} \triangleleft \mathbb{C}_\mathsf{q}'}\ [\text{COM}]$$

$$\frac{\mathsf{p}_i \triangleleft \mathbb{C}_{\mathsf{p}_i} \xrightarrow{A} \mathsf{p}_i \triangleleft \mathbb{C}'_{\mathsf{p}_i}\ \ \forall i \in I \qquad A \notin \mathscr{A}(\mathbb{C}_{\mathsf{q}_j})\ \ \forall j \in J}{\Pi_{i\in I}\mathsf{p}_i \triangleleft \mathbb{C}_{\mathsf{p}_i} \mid \Pi_{j\in J}\mathsf{q}_j \triangleleft \mathbb{C}_{\mathsf{q}_j} \xrightarrow{\tau} \Pi_{i\in I}\mathsf{p}_i \triangleleft \mathbb{C}'_{\mathsf{p}_i} \mid \Pi_{j\in J}\mathsf{q}_j \triangleleft \mathbb{C}_{\mathsf{q}_j}}\ [\text{RBM}]$$

$$\frac{\mathbb{N} \xrightarrow{\tau} \mathbb{N}'}{\mathbb{N} \| \mathbb{N}'' \xrightarrow{\tau} \mathbb{N}' \| \mathbb{N}''}\ [\text{PRN}] \qquad \frac{\mathbb{N}_1 \equiv \mathbb{N}_1' \quad \mathbb{N}_1' \xrightarrow{\tau} \mathbb{N}_2' \quad \mathbb{N}_2' \equiv \mathbb{N}_2}{\mathbb{N}_1 \xrightarrow{\tau} \mathbb{N}_2}\ [\text{EQN}]$$

Figure 3: Reduction rules of sessions and networks.

Example 2.4 *Consider the processes of Example 2.2. We first give some reductions possible for the configurations of the three participants starting from empty checkpointed sequences.*

$$\begin{array}{rll}
\varepsilon \prec P_{\mathtt{Tr}} & \xrightarrow{\mathtt{Ht!qr(in)}} \varepsilon \prec \mathtt{Al!qr(in)}.P'_{\mathtt{Tr}} = \mathbb{C}_1 & [\text{SND}]\ (1)\\
& \xrightarrow{\mathtt{Al!qr(in)}} \varepsilon \prec P'_{\mathtt{Tr}} = \mathbb{C}_2 & [\text{SND}]\ (2)\\
& \xrightarrow{\mathtt{Ht?av}} P'_{\mathtt{Tr}} \prec \mathtt{Al!rs}.(\blacktriangle_B \sum\{\mathtt{Al?nAv}\ ,\ \mathtt{Al?av}\}) = \mathbb{C}_3 & [\text{CKRCV}]\ (3)\\
& \xrightarrow{\mathtt{Al!rs}} P'_{\mathtt{Tr}} \prec \blacktriangle_B \sum\{\mathtt{Al?nAv}\ ,\ \mathtt{Al?av}\} = \mathbb{C}_4 & [\text{SND}]\ (4)\\
& \xrightarrow{\mathtt{Al?nAv}} P'_{\mathtt{Tr}} \cdot (\blacktriangle_B \sum\{\mathtt{Al?nAv}\ ,\ \mathtt{Al?av}\}) \prec \mathbf{0} = \mathbb{C}_5 & [\text{CKRCV}]\ (5)\\
\varepsilon \prec P_{\mathtt{Ht}} & \xrightarrow{\mathtt{Tr?qr(in)}} \varepsilon \prec \blacktriangle_A \bigoplus\{\mathtt{Tr!nAv}, \mathtt{Tr!av}\} = \mathbb{C}_6 & [\text{RCV}]\ (6)\\
& \xrightarrow{\tau} \blacktriangle_A \bigoplus\{\mathtt{Tr!nAv}, \mathtt{Tr!av}\} \prec \mathtt{Tr!av} = \mathbb{C}_7 & [\text{CKCHC}]\ (7)\\
& \xrightarrow{\mathtt{Tr!av}} \blacktriangle_A \bigoplus\{\mathtt{Tr!nAv}, \mathtt{Tr!av}\} \prec \mathbf{0} = \mathbb{C}_8 & [\text{SND}]\ (8)\\
\varepsilon \prec P_{\mathtt{Al}} & \xrightarrow{\mathtt{Tr?qr(in)}} \varepsilon \prec P'_{\mathtt{Al}} = \mathbb{C}_9 & [\text{RCV}]\ (9)\\
& \xrightarrow{\mathtt{Tr?rs}} P'_{\mathtt{Al}} \prec \blacktriangle_B \bigoplus\{\mathtt{Tr!nAv}, \mathtt{Tr!av}\} = \mathbb{C}_{10} & [\text{CKRCV}]\ (10)\\
& \xrightarrow{\tau} P'_{\mathtt{Al}} \cdot \blacktriangle_B \bigoplus\{\mathtt{Tr!nAv}, \mathtt{Tr!av}\} \prec \mathtt{Tr!av} = \mathbb{C}_{11} & [\text{CKCHC}]\ (11)\\
& \xrightarrow{\mathtt{Tr!av}} P'_{\mathtt{Al}} \cdot \blacktriangle_B \bigoplus\{\mathtt{Tr!nAv}, \mathtt{Tr!av}\} \prec \mathbf{0} = \mathbb{C}_{12} & [\text{SND}]\ (12)
\end{array}$$

Starting from the initial session $\mathbb{M} = \mathtt{Tr} \triangleleft \varepsilon \prec P_{\mathtt{Tr}} \mid \mathtt{Ht} \triangleleft \varepsilon \prec P_{\mathtt{Ht}} \mid \mathtt{Al} \triangleleft \varepsilon \prec P_{\mathtt{Al}}$*, in which all participants have their associated processes as active processes and empty checkpointed sequences, we can have the reductions shown in Figure 4.*

$$\begin{array}{rll}
\mathbb{M} & \xrightarrow{\tau} \mathtt{Tr} \triangleleft \mathbb{C}_1 \mid \mathtt{Ht} \triangleleft \mathbb{C}_6 \mid \mathtt{Al} \triangleleft \varepsilon \prec P_{\mathtt{Al}} & \text{using (1) and (6) and rules [PC], [COM], and [PRM]}\\
& \xrightarrow{\tau} \mathtt{Tr} \triangleleft \mathbb{C}_2 \mid \mathtt{Ht} \triangleleft \mathbb{C}_6 \mid \mathtt{Al} \triangleleft \mathbb{C}_9 & \text{using (2) and (9) and rules [PC], [COM], and [PRM]}\\
& \xrightarrow{\tau} \mathtt{Tr} \triangleleft \mathbb{C}_2 \mid \mathtt{Ht} \triangleleft \mathbb{C}_7 \mid \mathtt{Al} \triangleleft \mathbb{C}_9 & \text{using (7) and rules [PC], and [PRM]}\\
& \xrightarrow{\tau} \mathtt{Tr} \triangleleft \mathbb{C}_3 \mid \mathtt{Ht} \triangleleft \mathbb{C}_8 \mid \mathtt{Al} \triangleleft \mathbb{C}_9 & \text{using (3) and (8) and rules [PC], [COM], and [PRM]}\\
& \xrightarrow{\tau} \mathtt{Tr} \triangleleft \mathbb{C}_4 \mid \mathtt{Ht} \triangleleft \mathbb{C}_8 \mid \mathtt{Al} \triangleleft \mathbb{C}_{10} & \text{using (4) and (10) and rules [PC], [COM], and [PRM]}\\
& \xrightarrow{\tau} \mathtt{Tr} \triangleleft \mathbb{C}_4 \mid \mathtt{Ht} \triangleleft \mathbb{C}_8 \mid \mathtt{Al} \triangleleft \mathbb{C}_{11} & \text{using (11) and rules [PC], and [PRM]}\\
& \xrightarrow{\tau} \mathtt{Tr} \triangleleft \mathbb{C}_5 \mid \mathtt{Ht} \triangleleft \mathbb{C}_8 \mid \mathtt{Al} \triangleleft \mathbb{C}_{12} & \text{using (5) and (12) and rules [PC], [COM], and [PRM]}
\end{array}$$

Figure 4: Example of multiparty session reductions.

From the final session we can do a rollback to B as follows:

$$\dfrac{\dfrac{\mathbb{C}_5 \xrightarrow{B} \mathbb{C}_4 \quad [\text{RBP}]}{\mathtt{Tr} \triangleleft \mathbb{C}_5 \xrightarrow{B} \mathtt{Tr} \triangleleft \mathbb{C}_4}\,[\text{PART}] \qquad \dfrac{\mathbb{C}_{12} \xrightarrow{B} \mathbb{C}_{10} \quad [\text{RBP}]}{\mathtt{Al} \triangleleft \mathbb{C}_{12} \xrightarrow{B} \mathtt{Al} \triangleleft \mathbb{C}_{10}}\,[\text{PART}] \qquad B \notin \mathscr{A}(\mathbb{C}_8)}{\mathtt{Tr} \triangleleft \mathbb{C}_5 \mid \mathtt{Ht} \triangleleft \mathbb{C}_8 \mid \mathtt{Al} \triangleleft \mathbb{C}_{12} \xrightarrow{\tau} \mathtt{Tr} \triangleleft \mathbb{C}_4 \mid \mathtt{Ht} \triangleleft \mathbb{C}_8 \mid \mathtt{Al} \triangleleft \mathbb{C}_{10}}\,[\text{RBM}]$$

In a similar way, we can do a rollback to A from the session $\mathtt{Tr} \triangleleft \mathbb{C}_5 \mid \mathtt{Ht} \triangleleft \mathbb{C}_8 \mid \mathtt{Al} \triangleleft \mathbb{C}_{12}$ *producing* $\mathtt{Tr} \triangleleft \mathbb{C}_2 \mid \mathtt{Ht} \triangleleft \mathbb{C}_6 \mid \mathtt{Al} \triangleleft \mathbb{C}_9$.

3 Type System

Types *Sorts* are ranged over by S and defined by: $S ::= \mathtt{Int} \mid \mathtt{Bool} \mid \ldots$

Single-threaded global types describe the whole conversation scenarios of multiparty sessions, when they reduce forward. The communications can be either without or with named checkpoints.

Global types instead take into account both forward and backward reductions of multiparty sessions. They have therefore a structure which mimics the structure of configurations.

Definition 3.1 *1.* Single-threaded global types *are defined by:*

$$\mathsf{G} ::= \mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I} \mid \blacktriangle_A \mathsf{p} \to \mathsf{q} : \{\ell_j(S_j).\mathsf{G}_j\}_{j\in J} \mid \mu \boldsymbol{t}.\mathsf{G} \mid \boldsymbol{t} \mid \mathsf{end}$$

2. Global types *are pairs* $\Upsilon \prec \mathsf{G}$*, where* Υ *is a (possibly empty) sequence of single-threaded global types with checkpoints having distinct names:*

$$\Upsilon ::= \varepsilon \mid \Upsilon \cdot \blacktriangle_A \mathsf{G}$$

We say that G is the *active type* of $\Upsilon \prec \mathsf{G}$. The condition in point (2) of previous definition ensures that, if $\Upsilon \prec \mathsf{G}$ is a global type, and $\Upsilon = \Upsilon' \cdot \blacktriangle_A \mathsf{G}' \cdot \Upsilon''$, then no single-threaded global type in Υ', Υ'' can be checkpointed by A.

Session types correspond to projections of single-threaded global types onto the individual participants. Therefore, they can be decorated by named checkpoints. Inspired by [19], we use intersection and union types instead of standard branching and selection, see [13], to take advantage of the subtyping induced by subset inclusion. The grammar of session types, ranged over by T, is then

$$\mathsf{T} ::= \bigwedge_{i\in I} \mathsf{p}?\ell_i(S_i).\mathsf{T}_i \mid \bigvee_{i\in I} \mathsf{p}!\ell_i(S_i).\mathsf{T}_i \mid \blacktriangle_A \bigwedge_{j\in J} \mathsf{p}?\ell_j(S_j).\mathsf{T}_j \mid \blacktriangle_A \bigvee_{j\in J} \mathsf{p}!\ell_j(S_j).\mathsf{T}_j \mid \mu \mathbf{t}.\mathsf{T} \mid \mathbf{t} \mid \mathsf{end}$$

In both global and session types we require that:

- I, J are not empty sets and J is not a singleton;
- $\ell_h \neq \ell_k$ if $h,k \in I$ or $h,k \in J$;
- the name A does not occur in a type checkpointed by A;
- recursion is guarded.

We constrain J to contain at least two elements since it makes sense to reverse only when an alternative branch could be taken.

Recursive types with the same regular tree are considered equal [21, Chapter 20, Section 2]. In writing types we omit unnecessary brackets, intersections, unions, and end.

We extend the original definition of projection of single-threaded global types onto participants of [13] in the line of [10]. This generalisation allows session participants to behave differently in alternative branches of the same global type, after they have received a message identifying the branch. We define the partial operator $\between$ from sets of session types - all uncheckpointed or checkpointed by the same name - to (possibly checkpointed) intersection types. The operator is defined when the set of session types contains only intersection of types with same sender and different labels. Otherwise, it is undefined. More precisely:

$$\bigwedge\!\!\!\!\bigwedge(\{\mathsf{T}_i\}_{i\in I}) = \begin{cases} \bigwedge_{i\in I}\mathsf{T}_i & \text{if there is } \mathsf{p} \text{ such that } \mathsf{T}_i = \bigwedge_{h\in H_i}\mathsf{p}?\ell_h^{(i)}(S_h^{(i)}).\mathsf{T}_h^{(i)} \text{ for all } i\in I, \\ & \text{and } \ell_h^{(i)} \neq \ell_k^{(j)} \text{for all } h\in H_i, k\in H_j \text{ and } i,j\in I, i\neq j \\ \blacktriangle_A \bigwedge\!\!\!\!\bigwedge(\{\mathsf{T}'_i\}_{i\in I}) & \text{if } \mathsf{T}_i = \blacktriangle_A \mathsf{T}'_i \text{ for all } i\in I \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Notice that in defining $\bigwedge\!\!\!\!\bigwedge$ we could allow identical types in T_i and T_j by the idempotence of intersection. We prefer the current choice for its simplicity.

Figure 5 gives the projection of single-threaded global types onto participants. Notice that, the projection of a checkpointed type onto a participant not involved in the initial communication is defined only when it receives uncheckpointed messages in all the branches. For this reason, and since $\bigwedge\!\!\!\!\bigwedge$ is a partial operator, projections onto some participants may be undefined. A single-threaded global type G is *well formed* if all the projections of G onto all participants are defined. In the following we assume that all single-threaded global types are well formed.

$$(\mathsf{p}\to\mathsf{q}:\{\ell_i(S_i).\mathsf{G}_i\}_{i\in I})\upharpoonright\mathsf{r} = \begin{cases} \bigvee_{i\in I}\mathsf{q}!\ell_i(S_i).\mathsf{G}_i\upharpoonright\mathsf{r} & \text{if } \mathsf{r}=\mathsf{p} \\ \bigwedge_{i\in I}\mathsf{p}?\ell_i(S_i).\mathsf{G}_i\upharpoonright\mathsf{r} & \text{if } \mathsf{r}=\mathsf{q} \\ \mathsf{end} & \text{if } \mathsf{r}\neq\mathsf{p} \text{ and } \mathsf{r}\neq\mathsf{q} \text{ and } \mathsf{G}_i\upharpoonright\mathsf{r}=\mathsf{end} \text{ for all } i\in I \\ \bigwedge\!\!\!\!\bigwedge(\{\mathsf{G}_i\upharpoonright\mathsf{r}\mid i\in I\}) & \text{if } \mathsf{r}\neq\mathsf{p} \text{ and } \mathsf{r}\neq\mathsf{q} \end{cases}$$

$$(\blacktriangle_A\mathsf{p}\to\mathsf{q}:\{\ell_j(S_j).\mathsf{G}_j\}_{j\in J})\upharpoonright\mathsf{r} = \begin{cases} \blacktriangle_A\bigvee_{j\in J}\mathsf{q}!\ell_j(S_j).\mathsf{G}_j\upharpoonright\mathsf{r} & \text{if } \mathsf{r}=\mathsf{p} \\ \blacktriangle_A\bigwedge_{j\in J}\mathsf{p}?\ell_j(S_j).\mathsf{G}_j\upharpoonright\mathsf{r} & \text{if } \mathsf{r}=\mathsf{q} \\ \mathsf{end} & \text{if } \mathsf{r}\neq\mathsf{p} \text{ and } \mathsf{r}\neq\mathsf{q} \text{ and } \mathsf{G}_j\upharpoonright\mathsf{r}=\mathsf{end} \text{ for all } j\in J \\ \blacktriangle_A\bigwedge\!\!\!\!\bigwedge(\{\mathsf{G}_j\upharpoonright\mathsf{r}\mid j\in J\}) & \text{if } \mathsf{r}\neq\mathsf{p} \text{ and } \mathsf{r}\neq\mathsf{q} \\ & \text{and } \bigwedge\!\!\!\!\bigwedge(\{\mathsf{G}_j\upharpoonright\mathsf{r}\mid j\in J\}) \text{ is not checkpointed} \end{cases}$$

$$(\mu\mathbf{t}.\mathsf{G})\upharpoonright\mathsf{p} = \begin{cases} \mu\mathbf{t}.\mathsf{G}\upharpoonright\mathsf{p} & \text{if } \mathsf{p}\in\mathsf{G}, \\ \mathsf{end} & \text{otherwise.} \end{cases} \qquad \mathbf{t}\upharpoonright\mathsf{p}=\mathbf{t} \qquad \mathsf{end}\upharpoonright\mathsf{p}=\mathsf{end}$$

Figure 5: Projection of single-threaded global types onto participants.

Example 3.2 *Assuming that* Str *is the sort of strings, the global type for the interaction of Figure 1 is* G *defined as follows:*

$$\mathsf{G} = \mathtt{Tr}\to\mathtt{Ht}:\mathtt{qr}(\mathsf{Str}).\mathtt{Tr}\to\mathtt{Al}:\mathtt{qr}(\mathsf{Str}).\mathsf{G}_1 \textit{ where}$$

$$\mathsf{G}_1 = \blacktriangle_A\mathtt{Ht}\to\mathtt{Tr}:\{\mathtt{nAv}.\mathtt{Tr}\to\mathtt{Al}:\mathtt{ds}\ ,\ \mathtt{av}.\mathtt{Tr}\to\mathtt{Al}:\mathtt{rs}.\mathsf{G}_2\} \textit{ and } \mathsf{G}_2 = \blacktriangle_B\mathtt{Al}\to\mathtt{Tr}:\{\mathtt{nAv}\ ,\ \mathtt{av}\}$$

G *is well formed since the projections onto its participants are:*

$$\begin{array}{lcl} \mathsf{G}\upharpoonright\mathtt{Tr} & = & \mathtt{Ht}!\mathtt{qr}(\mathsf{Str}).\mathtt{Al}!\mathtt{qr}(\mathsf{Str}).\blacktriangle_A\bigwedge\{\mathtt{Ht}?\mathtt{nAv}.\mathtt{Al}!\mathtt{ds}\ ,\ \mathtt{Ht}?\mathtt{av}.\mathtt{Al}!\mathtt{rs}.\blacktriangle_B\bigwedge\{\mathtt{Al}?\mathtt{nAv}\ ,\ \mathtt{Al}?\mathtt{av}\}\} \\ \mathsf{G}\upharpoonright\mathtt{Ht} & = & \mathtt{Tr}?\mathtt{qr}(\mathsf{Str}).\blacktriangle_A\bigvee\{\mathtt{Tr}!\mathtt{nAv}\ ,\ \mathtt{Tr}!\mathtt{av}\} \\ \mathsf{G}\upharpoonright\mathtt{Al} & = & \mathtt{Tr}?\mathtt{qr}(\mathsf{Str}).\blacktriangle_A\bigwedge\{\mathtt{Tr}?\mathtt{ds}\ ,\ \mathtt{Tr}?\mathtt{rs}.\blacktriangle_B\bigvee\{\mathtt{Tr}!\mathtt{nAv}\ ,\ \mathtt{Tr}!\mathtt{av}\}\} \end{array}$$

In order to type checkpointed sequences and configurations we need (possibly empty) *sequences of checkpointed session types*, ranged over by ρ:

$$\rho ::= \varepsilon \mid \rho\cdot\blacktriangle_A\mathsf{T}$$

and pairs $\rho\prec\mathsf{T}$, dubbed *configuration types*.

The typing is made more flexible by a subtyping relation on session types exploiting the standard inclusions for intersection and union. A checkpointed type can be a subtype only of a type checkpointed by the same name. Figure 6 gives the subtyping rules: the double line in rules indicates that the rules are interpreted *coinductively* [21, Chapter 21]. Subtyping can be easily decided, see for example [11].

$$\frac{}{\mathsf{end} \leqslant \mathsf{end}}\ [\text{SUB-END}] \qquad \frac{\mathsf{T} \leqslant \mathsf{T}'}{\blacktriangle_A \mathsf{T} \leqslant \blacktriangle_A \mathsf{T}'}\ [\text{SUB-CK}]$$

$$\frac{\forall i \in I: \quad \mathsf{T}_i \leqslant \mathsf{T}'_i}{\bigwedge_{i \in I \cup J} \mathsf{p}?\ell_i(S_i).\mathsf{T}_i \leqslant \bigwedge_{i \in I} \mathsf{p}?\ell_i(S_i).\mathsf{T}'_i}\ [\text{SUB-IN}] \qquad \frac{\forall i \in I: \quad \mathsf{T}_i \leqslant \mathsf{T}'_i}{\bigvee_{i \in I} \mathsf{p}!\ell_i(S_i).\mathsf{T}_i \leqslant \bigvee_{i \in I \cup J} \mathsf{p}!\ell_i(S_i).\mathsf{T}'_i}\ [\text{SUB-OUT}]$$

Figure 6: Subtyping rules.

Typing Rules We distinguish six kinds of typing judgements

$$\Gamma \vdash \mathsf{e} : S \qquad \Gamma \vdash P : \mathsf{T} \qquad \vdash R : \rho \qquad \vdash \mathbb{C} : \rho \prec \mathsf{T} \qquad \vdash \mathbb{M} : \Upsilon \prec \mathsf{G} \qquad \vdash \mathbb{N}\ \checkmark$$

where Γ is the environment $\Gamma ::= \emptyset \mid \Gamma, x : S \mid \Gamma, X : \mathsf{T}$ that associates expression variables with sorts and process variables with session types.

Figure 7 gives the typing rules for processes. Processes typing exploits the correspondence between external choices and intersections, internal choices and unions. A checkpointed process has a type checkpointed by the same name.

$$\frac{\Gamma, x : S \vdash P_i : \mathsf{T}_i}{\Gamma \vdash \sum_{i \in I} \mathsf{p}?\ell_i(\mathsf{e}_i).P_i : \bigwedge_{i \in I} \mathsf{p}?\ell_i(S_i).\mathsf{T}_i}\ [\text{T-IN}] \qquad \frac{\Gamma \vdash \mathsf{e}_i : S_i \quad \Gamma \vdash P_i : \mathsf{T}_i}{\Gamma \vdash \bigoplus_{i \in I} \mathsf{p}!\ell_i(\mathsf{e}_i).P_i : \bigvee_{i \in I} \mathsf{p}!\ell_i(S_i).\mathsf{T}_i}\ [\text{T-OUT}]$$

$$\frac{\Gamma \vdash P : \mathsf{T}}{\Gamma \vdash \blacktriangle_A P : \blacktriangle_A \mathsf{T}}\ [\text{T-CK}] \qquad \Gamma \vdash \mathbf{0} : \mathsf{end}\ \ [\text{T-}\mathbf{0}]$$

$$\frac{\Gamma, X : \mathsf{T} \vdash P : \mathsf{T}}{\Gamma \vdash \mu X.P : \mathsf{T}}\ [\text{T-REC}] \qquad \Gamma, X : \mathsf{T} \vdash X : \mathsf{T}\ \ [\text{T-VAR}]$$

Figure 7: Typing rules for processes.

Figure 8 gives the remaining typing rules. Sequences of checkpointed processes are typed by sequences of checkpointed types (rule [T-SP]). Configurations are typed by configuration types (rule [T-C]).

The most interesting rule is rule [T-M] for typing multiparty sessions. The set $\mathsf{pt}(\mathsf{G})$ of participants

$$\frac{\vdash R : \rho \quad \vdash P : \mathsf{T}}{\vdash R \cdot P : \rho \cdot \mathsf{T}}\ [\text{T-SP}] \qquad \frac{\vdash R : \rho \quad \vdash P : \mathsf{T}}{\vdash \mathsf{p} \triangleleft R \prec P : \rho \prec \mathsf{T}}\ [\text{T-C}]$$

$$\frac{\vdash \mathsf{p}_i \triangleleft \mathbb{C}_i : \rho_i \prec \mathsf{T}_i \quad \rho_i \prec \mathsf{T}_i \ltimes_{\mathsf{p}_i} \Upsilon \prec \mathsf{G} \quad i \in I \quad |\Upsilon| = \max\{|\rho_i| \mid i \in I\} \quad \mathsf{pt}(\Upsilon) \cup \mathsf{pt}(\mathsf{G}) \subseteq \{\mathsf{p}_i \mid i \in I\}}{\vdash \Pi_{i \in I} \mathsf{p}_i \triangleleft \mathbb{C}_i : \Upsilon \prec \mathsf{G}}\ [\text{T-M}]$$

$$\frac{\vdash \mathbb{M} : \Upsilon \prec \mathsf{G}}{\vdash \mathbb{M}\ \checkmark}\ [\text{T-}\checkmark] \qquad \frac{\vdash \mathbb{N}\ \checkmark \quad \vdash \mathbb{N}'\ \checkmark}{\vdash \mathbb{N} \| \mathbb{N}'\ \checkmark}\ [\text{T-N}]$$

Figure 8: Typing rules for checkpointed sequences, multiparty sessions and networks.

of a single-threaded global type is defined by

$$\mathsf{pt}(\mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I}) = \{\mathsf{p},\mathsf{q}\} \cup \mathsf{pt}(\mathsf{G}_i)\ (i \in I)^1 \quad \mathsf{pt}(\blacktriangle_A \mathsf{G}) = \mathsf{pt}(\mu \mathbf{t}.\mathsf{G}) = \mathsf{pt}(\mathsf{G}) \quad \mathsf{pt}(\mathsf{end}) = \mathsf{pt}(\mathbf{t}) = \emptyset$$

The definition is extended to sequences of checkpointed single-threaded global types by

$$\mathsf{pt}(\varepsilon) = \emptyset \qquad \mathsf{pt}(\Upsilon \cdot \blacktriangle_A \mathsf{G}) = \mathsf{pt}(\Upsilon) \cup \mathsf{pt}(\mathsf{G})$$

The condition $\mathsf{pt}(\Upsilon) \cup \mathsf{pt}(\mathsf{G}) \subseteq \{\mathsf{p}_i \mid i \in I\}$ ensures the presence of all session participants and allows the typing of sessions containing $\mathsf{p} \triangleleft \varepsilon \prec \mathbf{0}$ (also if $\mathsf{p} \notin \mathsf{pt}(\Upsilon) \cup \mathsf{pt}(\mathsf{G})$), a property needed to guarantee invariance of types under structural equivalence. Rule [T-M] requires that the types of the active processes are subtypes of the projections of a unique global type. This condition must hold also after a rollback, in which all the processes checkpointed by A, in the respective checkpointed sequences, become the active processes (reduction rule [RBM]). For this reason we type a multiparty session by a global type such that the sequence of single-threaded global types has length equal to the maximum of the lengths of the sequences of session types in the types of configurations. This is expressed by the condition $|\Upsilon| = \max\{|\rho_i| \mid i \in I\}$ in the premise of rule [T-M], where $|\Upsilon|$ is the length of the sequence Υ and $|\rho|$ is the length of the sequence ρ. This requirement is clearly not enough. Further constraints are prescribed by the agreement between global types and configuration types of session participants. This agreement is made more flexible by the use of subtyping.

Definition 3.3 *Let* $\rho = \mathsf{T}_1 \cdot \ldots \cdot \mathsf{T}_n$ *and* $\Upsilon = \mathsf{G}_1 \cdot \ldots \cdot \mathsf{G}_m$. *The configuration type* $\rho \prec \mathsf{T}$ p-agrees *with the global type* $\Upsilon \prec \mathsf{G}$ *(notation* $\rho \prec \mathsf{T} \ltimes_{\mathsf{p}} \Upsilon \prec \mathsf{G}$*) if all the following conditions hold:*

1. $\mathsf{T}_i \leq \mathsf{G}_i \upharpoonright \mathsf{p}$ *for* $1 \leq i \leq n$;
2. *if* $\mathsf{T} = \mathsf{end}$, *then* $n \leq m$ *and* $\mathsf{G}_i \upharpoonright \mathsf{p} = \mathsf{G} \upharpoonright \mathsf{p} = \mathsf{end}$ *for* $n+1 \leq i \leq m$;
3. *if* T *is a union type, then* $n = m$ *and* $\mathsf{T} \leq \mathsf{G} \upharpoonright \mathsf{p}$;
4. *if* T *is an intersection type, then either* $n = m$ *and* $\mathsf{T} \leq \mathsf{G} \upharpoonright \mathsf{p}$ *or* $n = m-1$ *and* $\mathsf{T} \leq \mathsf{G}_m \upharpoonright \mathsf{p}$ *and* $\mathsf{T} = \blacktriangle_A \mathsf{T}'$ *and* $\mathsf{T}' \leq \mathsf{G} \upharpoonright \mathsf{p}$.

Condition 1, typing rule [T-CK], and the fact that the checkpoints in Υ have distinct names (see Definition 3.1), ensure that, after a rollback, the processes becoming active are in the same position inside the checkpointed sequences. Moreover, these processes have types which are subtypes of the projections of the same single-threaded global type. Condition 2 deals with the case in which the active process of p is $\mathbf{0}$. Once the active process of a participant is $\mathbf{0}$, no more process are added to its checkpointed sequence, so, in case of a rollback to a checkpoint crossed afterwards, by some other participants, its active process would remain $\mathbf{0}$. Therefore the projection of the global type should be end. Conditions 3 and 4 deal with the case in which the type of the active process of p is a union or an intersection. If $n = m$, then these conditions require that its type is subtype of the projection of the global type G, i.e., $\mathsf{T} \leq \mathsf{G} \upharpoonright \mathsf{p}$. If the active process of p is typed by an intersection, then it must be an external choice of input processes (rule [T-IN]). In this case the global type G could capture the situation in which a participant q has internally chosen one branch, memorised in the checkpointed sequence its active process, and the active process of q has became an uncheckpointed output (reduction rule [CKCHC]). This means that, the length of the checkpointed sequence of q must be m, while the length of the checkpointed sequence of p must be $m-1$. To deal with this situation, since an uncheckpointed and a checkpointed type cannot be projections of the same global type, in condition 4, we require that $\mathsf{T} \leq \mathsf{G}_m \upharpoonright \mathsf{p}$ and $\mathsf{T} = \blacktriangle_A \mathsf{T}'$ and $\mathsf{T}' \leq \mathsf{G} \upharpoonright \mathsf{p}$. Notice that, G must be uncheckpointed and G_m must be checkpointed by A. This condition is illustrated in Example 3.4.

Rule [T-M] requires that all types of the configurations which built the multiparty session agree with the global type of the session itself, conditions $\vdash \mathsf{p}_i \triangleleft \mathbb{C}_i : \rho_i \prec \mathsf{T}_i$ and $\rho_i \prec \mathsf{T}_i \ltimes_{\mathsf{p}_i} \Upsilon \prec \mathsf{G}$ for $i \in I$.

A network is well typed if and only if all its multiparty sessions are well typed.

[1] The projectability of G ensures $\mathsf{G}_i = \mathsf{G}_j$ for all $i, j \in I$.

Example 3.4 *To show the typings for the networks produced by the reduction of Example 2.4 consider the global type* G *of Example 3.2. We can derive*

1. $\vdash \mathtt{Tr} \triangleleft \varepsilon \prec P_{\mathtt{Tr}} \mid \mathtt{Ht} \triangleleft \varepsilon \prec P_{\mathtt{Ht}} \mid \mathtt{Al} \triangleleft \varepsilon \prec P_{\mathtt{Al}} : \varepsilon \prec \mathsf{G}$
2. $\vdash \mathtt{Tr} \triangleleft \mathbb{C}_1 \mid \mathtt{Ht} \triangleleft \mathbb{C}_6 \mid \mathtt{Al} \triangleleft \varepsilon \prec P_{\mathtt{Al}} : \varepsilon \prec \mathtt{Tr} \to \mathtt{Al} : \mathtt{qr}(\mathtt{in}).\mathsf{G}_1$
3. $\vdash \mathtt{Tr} \triangleleft \mathbb{C}_2 \mid \mathtt{Ht} \triangleleft \mathbb{C}_6 \mid \mathtt{Al} \triangleleft \mathbb{C}_9 : \varepsilon \prec \mathsf{G}_1$
4. $\vdash \mathtt{Tr} \triangleleft \mathbb{C}_2 \mid \mathtt{Ht} \triangleleft \mathbb{C}_7 \mid \mathtt{Al} \triangleleft \mathbb{C}_9 : \mathsf{G}_1 \prec \mathtt{Ht} \to \mathtt{Tr} : \mathtt{av}.\mathtt{Tr} \to \mathtt{Al} : \mathtt{rs}.\mathsf{G}_2$
5. $\vdash \mathtt{Tr} \triangleleft \mathbb{C}_3 \mid \mathtt{Ht} \triangleleft \mathbb{C}_8 \mid \mathtt{Al} \triangleleft \mathbb{C}_9 : \mathsf{G}_1 \prec \mathtt{Tr} \to \mathtt{Al} : \mathtt{rs}.\mathsf{G}_2$
6. $\vdash \mathtt{Tr} \triangleleft \mathbb{C}_4 \mid \mathtt{Ht} \triangleleft \mathbb{C}_8 \mid \mathtt{Al} \triangleleft \mathbb{C}_{10} : \mathsf{G}_1 \prec \mathsf{G}_2$
7. $\vdash \mathtt{Tr} \triangleleft \mathbb{C}_4 \mid \mathtt{Ht} \triangleleft \mathbb{C}_8 \mid \mathtt{Al} \triangleleft \mathbb{C}_{11} : \mathsf{G}_1 \cdot \mathsf{G}_2 \prec \mathtt{Al} \to \mathtt{Tr} : \mathtt{av}$
8. $\vdash \mathtt{Tr} \triangleleft \mathbb{C}_5 \mid \mathtt{Ht} \triangleleft \mathbb{C}_8 \mid \mathtt{Al} \triangleleft \mathbb{C}_{12} : \mathsf{G}_1 \cdot \mathsf{G}_2 \prec \mathsf{end}$

In typing 6, the active type G_2 *(see Example 3.2) is checkpointed by B and the type of* $\mathtt{Tr}$ *is* $\mathsf{G}_2 \upharpoonright \mathtt{Tr}$*, see the definition of* $\mathbb{C}_4$ *in Example 2.4. In typing 7, the active type is* $\mathtt{Al} \to \mathtt{Tr} : \mathtt{av}$*. The type of* $\mathtt{Tr}$ *remain the same and without the checkpoint B is a subtype of* $(\mathtt{Al} \to \mathtt{Tr} : \mathtt{av}) \upharpoonright \mathtt{Tr}$*. In these two typings the type of* $\mathtt{Tr}$ *agrees with the first and the second case of condition 4 in Definition 3.3, respectively.*

4 Main Properties

In this section we present the technical results of the paper. First we prove subject reduction for multiparty sessions (Theorem 4.3). This implies that well-typed networks respect the choreographies described by global types (Theorem 4.4). This property is usually called session fidelity, see [9]. The progress theorem (Theorem 4.5) establishes reachability of all communications and backward reductions.

As standard we start with an inversion lemma for processes, checkpointed sequences, configurations, multiparty sessions and networks.

Lemma 4.1 *[Inversion]*

1. *Let* $\Gamma \vdash P : \mathsf{T}$.
   - *(a) If* $P = \sum_{i\in I} \mathtt{p}?\ell_i(x_i).P_i$*, then* $\mathsf{T} = \bigwedge_{i\in I} \mathtt{p}?\ell_i(S_i).\mathsf{T}_i$*, and* $\Gamma, x{:}S_i \vdash P_i : \mathsf{T}_i$ *for* $i \in I$.
   - *(b) If* $P = \bigoplus_{i\in I} \mathtt{p}!\ell_i(\mathsf{e}_i).P_i$*, then* $\mathsf{T} = \bigvee_{i\in I} \mathtt{p}!\ell_i(S_i).\mathsf{T}_i$*,* $\Gamma \vdash \mathsf{e}_i : S_i$*, and* $\Gamma \vdash P_i : \mathsf{T}_i$ *for* $i \in I$.
   - *(c) If* $P = \blacktriangle_A \sum_{j\in J} \mathtt{p}?\ell_j(x_j).P_j$*, then* $\mathsf{T} = \blacktriangle_A \bigwedge_{j\in J} \mathtt{p}?\ell_j(S_j).\mathsf{T}_j$*, and* $\Gamma, x{:}S_j \vdash P_j : \mathsf{T}_j$ *for* $j \in J$.
   - *(d) If* $P = \blacktriangle_A \bigoplus_{j\in J} \mathtt{p}!\ell_j(\mathsf{e}_j).P_j$*, then* $\mathsf{T} = \blacktriangle_A \bigvee_{j\in J} \mathtt{p}!\ell_j(S_j).\mathsf{T}_j$*,* $\Gamma \vdash \mathsf{e}_j : S_j$*, and* $\Gamma \vdash P_j : \mathsf{T}_j$ *for* $j \in J$.
   - *(e) If* $P = \mu X.Q$*, then* $\Gamma, X{:}\mathsf{T} \vdash Q : \mathsf{T}$.
   - *(f) If* $P = X$*, then* $\Gamma = \Gamma', X{:}\mathsf{T}$.
   - *(g) If* $P = \mathbf{0}$*, then* $\mathsf{T} = \mathsf{end}$.
2. *If* $\vdash R \cdot P : \rho$*, then* $\rho = \rho' \cdot \mathsf{T}$ *and* $\vdash R : \rho'$ *and* $\vdash P : \mathsf{T}$.
3. *If* $\vdash \mathtt{p} \triangleleft R \prec P : \rho \prec \mathsf{T}$*, then* $\vdash R : \rho$ *and* $\vdash P : \mathsf{T}$.
4. *If* $\vdash \Pi_{i\in I} \mathtt{p}_i \triangleleft \mathbb{C}_i : \Upsilon \prec \mathsf{G}$*, then* $\vdash \mathtt{p}_i \triangleleft \mathbb{C}_i : \rho_i \prec \mathsf{T}_i$ *and* $\rho_i \prec \mathsf{T}_i \ltimes_{\mathtt{p}_i} \Upsilon \prec \mathsf{G}$ *for* $i \in I$ *and* $|\Upsilon| = max\{|\rho_i| \mid i \in I\}$ *and* $\mathsf{pt}(\Upsilon) \cup \mathsf{pt}(\mathsf{G}) \subseteq \{\mathtt{p}_i \mid i \in I\}$.
5. *If* $\vdash \mathbb{N}\ \checkmark$*, then either* $\vdash \mathbb{N} : \Upsilon \prec \mathsf{G}$ *or* $\mathbb{N} = \mathbb{N}' \mid \mathbb{N}''$ *and* $\vdash \mathbb{N}'\ \checkmark$ *and* $\vdash \mathbb{N}''\ \checkmark$.

Proof Easy from the definition of the typing relation.

The inversion lemma gives some important properties.

Points (1a), (1b), (1c),(1d) and (1g) ensure that the processes are checkpointed iff their types are checkpointed with the same name. Moreover, input processes have intersection types, output processes have union types and the process $\mathbf{0}$ has type end.

Point (2) says that the length of checkpointed sequences is equal to the length of the sequences of their checkpointed session types.

Point (4) and Definition 3.3 imply that in a well-typed multiparty session:

- exactly one of the active processes is an output process;

- at least one of the active processes is an input process.

More precisely, if $\mathsf{G} = {}_{\blacktriangle_A}\mathsf{p} \to \mathsf{q} : \{\ell_j(S_j).\mathsf{G}_j\}_{j\in J}$, then the active processes of participants p and q must have types which are subtypes of $\mathsf{G}\upharpoonright\mathsf{p}$ and $\mathsf{G}\upharpoonright\mathsf{q}$, respectively. E.g., this is the case of typing 3 of Example 3.4, where the active processes of `Ht` and `Tr` have types equal to the projections the active global type onto the respective participant. (Similarly for typing 6.) If $\mathsf{G} = \mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I}$, then the active process of participant p must have a type which is a subtype of $\mathsf{G}\upharpoonright\mathsf{p}$, as before. Instead, the active process of participant q can have either a type which is a subtype of $\mathsf{G}\upharpoonright\mathsf{q}$, or a checkpointed type ${}_{\blacktriangle_A}\mathsf{T}$ such that $\mathsf{T} \leqslant \mathsf{G}\upharpoonright\mathsf{q}$. E.g., this is the case of typing 7 in Example 3.4, where the active processes of `Al` has an uncheckpointed output type (a union of a single type), whereas the active process of `Tr` has an intersection type checkpointed by A. If G is checkpointed by A, then an active process different from $\mathbf{0}$ is checkpointed by A and the checkpointed sequence has length $|\Upsilon|$. Instead, if G is uncheckpointed, then the output process is uncheckpointed and its checkpointed sequence has length $|\Upsilon|$, while an input process can be:

- either uncheckpointed: in this case its checkpointed sequence has length $|\Upsilon|$,
- or checkpointed: in this case its checkpointed sequence has length $|\Upsilon| - 1$.

E.g., for the case of typing 3 of Example 3.4, the length of the checkpointed sequences of all participants is 0, and all participants have checkpointed active processes, whereas for typing 7, the length of the checkpointed sequence of `Al` is 2, whereas the ones of the other participants is 1. Moreover, the active process of `Tr` is checkpointed and the one of `Ht` is $\mathbf{0}$. These properties follow from conditions 3 and 4 of Definition 3.3. Condition 4 of Definition 3.3 implies also that, if G is uncheckpointed and the type of an active input process is checkpointed, then the name of its checkpoint is the name of the checkpoint of the global type on the top of Υ. Lastly, condition 1 of Definition 3.3 ensures that all the processes in the checkpointed sequences are checkpointed by the same names as the global types in Υ, in the exact order. The only difference can be the length of the sequences, which must satisfy the other conditions of Definition 3.3.

Global types are not preserved under multiparty session reductions: this is expected, as they evolve according to the silent actions, the communications and the rollbacks performed by the session participants. This evolution is formalised by the *reduction of global types*, which is the smallest pre-order relation closed under the rules of Figure 9. Rule [G-CkChc] corresponds to the reduction of the output process of participant p by rule [CkChc] of Figure 2. Notably, the process is checkpointed with name A. Rule [G-Com] corresponds to the communication performed by rule [Com] of Figure 3. I.e., the output process of participant p sends a message labelled ℓ_k (rule [Snd] of Figure 2) and the input process of participant q receives this message (rule [Rcv] or [CkRcv] of Figure 2). Notice that, when rule [CkRcv] is used, the checkpointed input process is added to the checkpointed sequence of participant q. Lastly, rule [G-Rb] is used when the multiparty session rolls back by means of rule [RbM] of Figure 3. Let A be the name of the checkpoint of G, the participants must either reduce by rule [RbP] of Figure 2 with a transition labelled A or remain unchanged. In the unchanged configurations the processes belonging to the checkpointed sequences are not checkpointed by A and the active processes are $\mathbf{0}$. We use $\Longrightarrow^*$ to denote the transitive and reflexive closure of the $\Longrightarrow$ relation.

A standard substitution lemma is handy.

Lemma 4.2 *If* $\Gamma, x : S \vdash P : \mathsf{T}$ *and* $\Gamma \vdash \mathsf{e} : S$ *and* $\mathsf{e} \downarrow v$, *then* $\Gamma \vdash P\{v/x\} : \mathsf{T}$.

We can show subject reduction for well-typed multiparty sessions, which implies subject reduction for well-typed networks.

Theorem 4.3 (SR) *If* $\vdash \mathbb{M} : \Upsilon \prec \mathsf{G}$ *and* $\mathbb{M} \xrightarrow{\tau}{}^* \mathbb{M}'$, *then* $\vdash \mathbb{M}' : \Upsilon' \prec \mathsf{G}'$ *and* $\Upsilon \prec \mathsf{G} \Longrightarrow^* \Upsilon' \prec \mathsf{G}'$.

$$\Upsilon \prec {}_{\blacktriangle_A}\mathsf{p} \to \mathsf{q} : \{\ell_j(S_j).\mathsf{G}_j\}_{j\in J} \Longrightarrow \Upsilon\cdot({}_{\blacktriangle_A}\mathsf{p} \to \mathsf{q} : \{\ell_j(S_j).\mathsf{G}_j\}_{j\in J}) \prec \mathsf{p} \to \mathsf{q} : \{\ell_j(S_j).\mathsf{G}_j\}_{j\in J} \quad \text{[G-CkChc]}$$

$$\Upsilon \prec \mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I} \Longrightarrow \Upsilon \prec \mathsf{G}_k \quad k \in I \quad \text{[G-Com]}$$

$$\Upsilon\cdot\mathsf{G}\cdot\Upsilon' \prec \mathsf{G}' \Longrightarrow \Upsilon \prec \mathsf{G} \quad \text{[G-Rb]}$$

Figure 9: Reduction rules of global types.

Proof By induction on multiparty session reductions. It is easy to verify that typing is invariant under structural equivalence of multiparty sessions, so we will omit the application of rule [EqM].
If $\mathbb{M} \xrightarrow{\tau} \mathbb{M}'$, then there are three cases:

1. $\mathbb{M} = \mathsf{p} \triangleleft \mathbb{C} \mid \mathbb{M}''$ and $\mathbb{M}' = \mathsf{p} \triangleleft \mathbb{C}' \mid \mathbb{M}''$ and $\mathsf{p} \triangleleft \mathbb{C} \xrightarrow{\tau} \mathsf{p} \triangleleft \mathbb{C}'$, i.e., rule [PrM] has been applied,
2. $\mathbb{M} = \mathsf{p} \triangleleft \mathbb{C}_\mathsf{p} \mid \mathsf{q} \triangleleft \mathbb{C}_\mathsf{q} \mid \mathbb{M}''$ and $\mathbb{M}' = \mathsf{p} \triangleleft \mathbb{C}'_\mathsf{p} \mid \mathsf{q} \triangleleft \mathbb{C}'_\mathsf{q} \mid \mathbb{M}''$ and $\mathsf{p} \triangleleft \mathbb{C}_\mathsf{p} \xrightarrow{\mathsf{q}!\ell(v)} \mathsf{p} \triangleleft \mathbb{C}'_\mathsf{p}$ and $\mathsf{q} \triangleleft \mathbb{C}_\mathsf{q} \xrightarrow{\mathsf{p}?\ell(v)} \mathsf{q} \triangleleft \mathbb{C}'_\mathsf{q}$, i.e., rule [PrM], with rule [Com] on the premise, has been applied,
3. $\mathbb{M} = \Pi_{i\in I}\mathsf{p}_i \triangleleft \mathbb{C}_{\mathsf{p}_i} \mid \Pi_{j\in J}\mathsf{p}_j \triangleleft \mathbb{C}_{\mathsf{p}_j}$ and $\mathbb{M}' = \Pi_{i\in I}\mathsf{p}_i \triangleleft \mathbb{C}'_{\mathsf{p}_i} \mid \Pi_{j\in J}\mathsf{p}_j \triangleleft \mathbb{C}_{\mathsf{p}_j}$ and $\mathsf{p}_i \triangleleft \mathbb{C}_{\mathsf{p}_i} \xrightarrow{A} \mathsf{p}_i \triangleleft \mathbb{C}'_{\mathsf{p}_i}$ for all $i \in I$ and $A \notin \mathscr{A}(\mathbb{C}_{\mathsf{p}_j})$ for all $j \in J$, i.e., rule [RbM] has been applied.

Case (1). From $\mathsf{p} \triangleleft \mathbb{C} \xrightarrow{\tau} \mathsf{p} \triangleleft \mathbb{C}'$ we get $\mathbb{C} \xrightarrow{\tau} \mathbb{C}'$. Let $\mathbb{C} = R \prec P$ and $\mathbb{C}' = R' \prec P'$. Therefore

(a) either $\mathbb{C} \xrightarrow{\tau} \mathbb{C}'$ with rule [Chc], which implies $P = \bigoplus_{i\in I} \mathsf{q}!\ell_i(\mathsf{e}_i).P_i$ and $R' = R$ and $P' = \mathsf{q}!\ell_k(\mathsf{e}_k).P_k$ for $k \in I \neq \{k\}$,

(b) or $\mathbb{C} \xrightarrow{\tau} \mathbb{C}'$ with rule [CkChc], which implies $P = {}_{\blacktriangle_A}\bigoplus_{j\in J} \mathsf{q}!\ell_j(\mathsf{e}_j).P_j$ and $R' = R\cdot P$ and $P' = \mathsf{q}!\ell_k(\mathsf{e}_k).P_k$ for $k \in J \neq \{k\}$.

By Lemma 4.1(4) $\vdash \mathsf{p} \triangleleft \mathbb{C} : \rho \prec \mathsf{T}$ and $\rho \prec \mathsf{T} \ltimes_\mathsf{p} \Upsilon \prec \mathsf{G}$. Then $\vdash R : \rho$ and $\vdash P : \mathsf{T}$ by Lemma 4.1(3). Lemma 4.1(1b) and (1d) imply that T is a union type, then we get $|\rho| = |\Upsilon|$ and $\mathsf{T} \leqslant \mathsf{G} \upharpoonright \mathsf{p}$ by condition 3 of Definition 3.3.
Case (1a). Lemma 4.1(1b) applied to $\vdash P : \mathsf{T}$ gives $\mathsf{T} = \bigvee_{i\in I} \mathsf{q}!\ell_i(S_i).\mathsf{T}_i$ and $\vdash \mathsf{e}_i : S_i$ and $\vdash P_i : \mathsf{T}_i$ for $i \in I$. Then $\mathsf{T} \leqslant \mathsf{G} \upharpoonright \mathsf{p}$ implies $\mathsf{G} = \mathsf{p} \to \mathsf{q} : \{\ell_i(S_i).\mathsf{G}_i\}_{i\in I\cup L}$. We choose $\Upsilon' = \Upsilon$ and $\mathsf{G}' = \mathsf{G}$. In fact, we can derive $\vdash P' : \mathsf{q}!\ell_k(S_k).\mathsf{T}_k$ and $\mathsf{q}!\ell_k(S_k).\mathsf{T}_k \leqslant \mathsf{G} \upharpoonright \mathsf{p}$. Therefore, we derive $\vdash \mathbb{M}' : \Upsilon' \prec \mathsf{G}'$ by rule [T-M].
Case (1b). Lemma 4.1(1d) applied to $\vdash P : \mathsf{T}$ gives $\mathsf{T} = {}_{\blacktriangle_A}\bigvee_{j\in J} \mathsf{q}!\ell_j(S_j).\mathsf{T}_j$ and $\vdash \mathsf{e}_j : S_j$ and $\vdash P_j : \mathsf{T}_j$ for $j \in J$. Then $\mathsf{T} \leqslant \mathsf{G} \upharpoonright \mathsf{p}$ implies $\mathsf{G} = {}_{\blacktriangle_A}\mathsf{p} \to \mathsf{q} : \{\ell_j(S_j).\mathsf{G}_j\}_{j\in J\cup L}$. We can choose $\Upsilon' = \Upsilon\cdot\mathsf{G}$ and $\mathsf{G}' = \mathsf{p} \to \mathsf{q} : \{\ell_j(S_j).\mathsf{G}_j\}_{j\in J\cup L}$. In fact $\Upsilon \prec \mathsf{G} \Longrightarrow \Upsilon\cdot\mathsf{G} \prec \mathsf{G}'$ by rule [G-CkChc] and we can derive $\vdash P' : \mathsf{q}!\ell_k(S_k).\mathsf{T}_k$ and $\mathsf{q}!\ell_k(S_k).\mathsf{T}_k \leqslant \mathsf{G}' \upharpoonright \mathsf{p}$.
If $\mathsf{r} \neq \mathsf{p}$ and $\mathsf{r} \triangleleft R_\mathsf{r} \prec P_\mathsf{r}$ occurs in $\mathbb{M}$, then by Lemma 4.1(4) $\vdash \mathsf{r} \triangleleft R_\mathsf{r} \prec P_\mathsf{r} : \rho_\mathsf{r} \prec \mathsf{T}_\mathsf{r}$ and $\rho_\mathsf{r} \prec \mathsf{T}_\mathsf{r} \ltimes_\mathsf{r} \Upsilon \prec \mathsf{G}$. Lemma 4.1(3) gives $\vdash R_\mathsf{r} : \rho_\mathsf{r}$ and $\vdash P_\mathsf{r} : \mathsf{T}_\mathsf{r}$. If $P_\mathsf{r} = \mathbf{0}$, then $\mathsf{T}_\mathsf{r} = \mathsf{G} \upharpoonright \mathsf{r} = \mathsf{G}' \upharpoonright \mathsf{r} = \mathsf{end}$ and $\rho_\mathsf{r} \prec \mathsf{T}_\mathsf{r} \ltimes_\mathsf{r} \Upsilon' \prec \mathsf{G}'$, since condition 2 of Definition 3.3 is satisfied. If P_r is an input process, then T_r is an intersection type and $|\rho_\mathsf{r}| = |\Upsilon|$ and $\mathsf{T}_\mathsf{r} \leqslant \mathsf{G} \upharpoonright \mathsf{r}$ by the first alternative in condition 4 of Definition 3.3. We have $|\rho_\mathsf{r}| = |\Upsilon'| - 1$ since $|\rho_\mathsf{r}| = |\Upsilon|$. From $\mathsf{T}_\mathsf{r} \leqslant \mathsf{G} \upharpoonright \mathsf{r}$ we get $\mathsf{T}_\mathsf{r} = {}_{\blacktriangle_A}\mathsf{T}'_\mathsf{r}$ and $\mathsf{T}'_\mathsf{r} \leqslant \mathsf{G}' \upharpoonright \mathsf{r}$. Then $\rho_\mathsf{r} \prec \mathsf{T}_\mathsf{r} \ltimes_\mathsf{r} \Upsilon' \prec \mathsf{G}'$ since the second alternative of condition 4 in Definition 3.3 is satisfied. We can then derive $\vdash \mathbb{M}' : \Upsilon' \prec \mathsf{G}'$ by rule [T-M].
Case (2). From $\mathsf{p} \triangleleft \mathbb{C}_\mathsf{p} \xrightarrow{\mathsf{q}!\ell(v)} \mathsf{p} \triangleleft \mathbb{C}'_\mathsf{p}$ and $\mathsf{q} \triangleleft \mathbb{C}_\mathsf{q} \xrightarrow{\mathsf{p}?\ell(v)} \mathsf{q} \triangleleft \mathbb{C}'_\mathsf{q}$ we get that $\mathbb{C}_\mathsf{p} \xrightarrow{\mathsf{q}!\ell(v)} \mathbb{C}'_\mathsf{p}$ and $\mathbb{C}_\mathsf{q} \xrightarrow{\mathsf{p}?\ell(v)} \mathbb{C}'_\mathsf{q}$. Let $\mathbb{C}_\mathsf{p} = R_\mathsf{p} \prec P_\mathsf{p}$ and $\mathbb{C}'_\mathsf{p} = R'_\mathsf{p} \prec P'_\mathsf{p}$ and $\mathbb{C}_\mathsf{q} = R_\mathsf{q} \prec P_\mathsf{q}$ and $\mathbb{C}'_\mathsf{q} = R'_\mathsf{q} \prec P'_\mathsf{q}$. Then $\mathbb{C}_\mathsf{p}$ reduces with rule [Snd], which implies $P_\mathsf{p} = \mathsf{q}!\ell(\mathsf{e}).P$ and $\mathsf{e} \downarrow v$ and $R'_\mathsf{p} = R_\mathsf{p}$ and $P'_\mathsf{p} = P$. The reduction of $\mathbb{C}_\mathsf{q}$ can be done

(a) either with rule [Rcv], which implies $P_\mathsf{q} = \sum_{i\in I} \mathsf{p}?\ell_i(x_i).P_i$ with $\ell_k = \ell$ and $R'_\mathsf{q} = R_\mathsf{q}$ and $P'_\mathsf{q} = P_k\{v/x\}$,

(b) or with rule [CKRCV], which implies $P_\mathsf{q} = \blacktriangle_A \sum_{j\in J} \mathsf{p}?\ell_j(x_j).P_j$ with $\ell_k = \ell$ and $R'_\mathsf{q} = R_\mathsf{q} \cdot P_\mathsf{q}$ and $P'_\mathsf{q} = P_k\{v/x\}$.

By Lemma 4.1(4) $\vdash \mathsf{p} \triangleleft \mathbb{C}_\mathsf{p} : \rho_\mathsf{p} \prec \mathsf{T}_\mathsf{p}$ and $\rho_\mathsf{p} \prec \mathsf{T}_\mathsf{p} \ltimes_\mathsf{p} \Upsilon \prec \mathsf{G}$ and $\vdash \mathsf{q} \triangleleft \mathbb{C}_\mathsf{q} : \rho_\mathsf{q} \prec \mathsf{T}_\mathsf{p}$ and $\rho_\mathsf{q} \prec \mathsf{T}_\mathsf{q} \ltimes_\mathsf{q} \Upsilon \prec \mathsf{G}$. Then $\vdash R_\mathsf{p} : \rho_\mathsf{p}$ and $\vdash P_\mathsf{p} : \mathsf{T}_\mathsf{p}$ and $\vdash R_\mathsf{q} : \rho_\mathsf{q}$ and $\vdash P_\mathsf{q} : \mathsf{T}_\mathsf{q}$ by Lemma 4.1(3). Lemma 4.1(1b) applied to $\vdash P_\mathsf{p} : \mathsf{T}_\mathsf{p}$ gives $\mathsf{T}_\mathsf{p} = \mathsf{q}!\ell(S).\mathsf{T}$ and $\vdash \mathsf{e} : S$ and $\vdash P : \mathsf{T}$. We have $|\rho_\mathsf{p}| = |\Upsilon|$ and $\mathsf{T}_\mathsf{p} \leqslant \mathsf{G} \upharpoonright \mathsf{p}$ by condition 3 of Definition 3.3. This implies $\mathsf{G} = \mathsf{p} \to \mathsf{q} : \{\ell_h(S_h).\mathsf{G}_h\}_{h\in H}$ with $\ell_k = \ell$, $S = S_k$ and $\mathsf{T} \leqslant \mathsf{G}_k \upharpoonright \mathsf{p}$. We derive $\vdash \mathsf{p} \triangleleft \mathbb{C}'_\mathsf{p} : \rho_\mathsf{p} \prec \mathsf{T}$.

We can choose $\Upsilon' = \Upsilon$ and $\mathsf{G}' = \mathsf{G}_k$ since $\Upsilon \prec \mathsf{G} \Longrightarrow \Upsilon \prec \mathsf{G}_k$ by rule [G-COM] and we will show that $\vdash \mathbb{M}' : \Upsilon' \prec \mathsf{G}'$ is derivable by checking the agreement conditions of Definition 3.3 for all pairs participant/configuration of $\mathbb{M}'$. From $\rho_\mathsf{p} \prec \mathsf{T}_\mathsf{p} \ltimes_\mathsf{p} \Upsilon \prec \mathsf{G}$ and $\mathsf{T} \leqslant \mathsf{G}_k \upharpoonright \mathsf{p}$ we get $\rho_\mathsf{p} \prec \mathsf{T} \ltimes_\mathsf{p} \Upsilon' \prec \mathsf{G}'$.

Case (2a). Lemma 4.1(1a) applied to $\vdash P_\mathsf{q} : \mathsf{T}_\mathsf{q}$ gives $\mathsf{T}_\mathsf{q} = \bigwedge_{i\in I} \mathsf{q}?\ell_i(S_i).\mathsf{T}_i$ and $x_i : S_i \vdash P_i : \mathsf{T}_i$ for $i \in I$. We get $|\rho_\mathsf{p}| = |\Upsilon|$ and $\mathsf{T}_\mathsf{q} \leqslant \mathsf{G} \upharpoonright \mathsf{q}$ by condition 4 of Definition 3.3. This implies $H \subseteq I$ and in particular $k \in I$. The Substitution Lemma implies $\vdash P'_\mathsf{q} : \mathsf{T}_k$. We derive $\vdash \mathsf{q} \triangleleft \mathbb{C}'_\mathsf{q} : \rho_\mathsf{q} \prec \mathsf{T}_k$. From $\rho_\mathsf{q} \prec \mathsf{T}_\mathsf{q} \ltimes_\mathsf{q} \Upsilon \prec \mathsf{G}$ and $\mathsf{T}_k \leqslant \mathsf{G}_k \upharpoonright \mathsf{q}$ we get $\rho_\mathsf{q} \prec \mathsf{T}_k \ltimes_\mathsf{q} \Upsilon' \prec \mathsf{G}'$.

Case (2b). Lemma 4.1(1c) applied to $\vdash P_\mathsf{q} : \mathsf{T}_\mathsf{q}$ gives $\mathsf{T}_\mathsf{q} = \blacktriangle_A \bigwedge_{j\in J} \mathsf{q}?\ell_j(S_j).\mathsf{T}_j$ and $x_j : S_j \vdash P_j : \mathsf{T}_j$ for $j \in J$. Let $\Upsilon = \Upsilon'' \cdot \mathsf{G}''$. We get $|\rho_\mathsf{p}| = |\Upsilon| - 1$ and $\mathsf{T}_\mathsf{q} \leqslant \mathsf{G}'' \upharpoonright \mathsf{q}$ and $\bigwedge_{j\in J} \mathsf{q}?\ell_j(S_j).\mathsf{T}_j \leqslant \mathsf{G} \upharpoonright \mathsf{q}$ by condition 4 of Definition 3.3. This implies $H \subseteq J$ and in particular $k \in J$. As in previous case the Substitution Lemma implies $\vdash P'_\mathsf{q} : \mathsf{T}_k$. We derive $\vdash \mathsf{q} \triangleleft \mathbb{C}'_\mathsf{q} : \rho_\mathsf{q} \cdot \mathsf{T}_\mathsf{q} \prec \mathsf{T}_k$. From $\rho_\mathsf{q} \prec \mathsf{T}_\mathsf{q} \ltimes_\mathsf{q} \Upsilon'' \cdot \mathsf{G}'' \prec \mathsf{G}$ and $\mathsf{T}_\mathsf{q} \leqslant \mathsf{G}'' \upharpoonright \mathsf{q}$ and $\mathsf{T}_k \leqslant \mathsf{G}_k \upharpoonright \mathsf{q}$ we get $\rho_\mathsf{q} \cdot \mathsf{T}_\mathsf{q} \prec \mathsf{T}_k \ltimes_\mathsf{q} \Upsilon' \prec \mathsf{G}'$.

Consider a participant $\mathsf{r} \neq \mathsf{p}, \mathsf{q}$. If $\mathsf{r} \triangleleft R_\mathsf{r} \prec P_\mathsf{r}$ occurs in $\mathbb{M}$, then by Lemma 4.1(4) $\vdash \mathsf{r} \triangleleft R_\mathsf{r} \prec P_\mathsf{r} : \rho_\mathsf{r} \prec \mathsf{T}_\mathsf{r}$ and $\rho_\mathsf{r} \prec \mathsf{T}_\mathsf{r} \ltimes_\mathsf{r} \Upsilon \prec \mathsf{G}$. Lemma 4.1(3) gives $\vdash R_\mathsf{r} : \rho_\mathsf{r}$ and $\vdash P_\mathsf{r} : \mathsf{T}_\mathsf{r}$. If $P_\mathsf{r} = \mathbf{0}$, then $\mathsf{T}_\mathsf{r} = \mathsf{G} \upharpoonright \mathsf{r} = \mathsf{G}' \upharpoonright \mathsf{r} = \mathsf{end}$ and $\rho_\mathsf{r} \prec \mathsf{T}_\mathsf{r} \ltimes_\mathsf{r} \Upsilon' \prec \mathsf{G}'$, since condition 2 of Definition 3.3 is satisfied. If P_r is an input process, then T_r is an intersection type and either $|\rho_\mathsf{r}| = |\Upsilon|$ and $\mathsf{T}_\mathsf{r} \leqslant \mathsf{G} \upharpoonright \mathsf{r}$ or $|\rho_\mathsf{r}| = |\Upsilon| - 1$ and $\mathsf{T}_\mathsf{r} \leqslant \mathsf{G}'' \upharpoonright \mathsf{r}$ and $\mathsf{T}_\mathsf{r} = \blacktriangle_A \mathsf{T}'$ and $\mathsf{T}' \leqslant \mathsf{G} \upharpoonright \mathsf{r}$ by condition 4 of Definition 3.3. In both cases $\mathsf{G} \upharpoonright \mathsf{r} \leqslant \mathsf{G}_k \upharpoonright \mathsf{r}$. If G_k is uncheckpointed we conclude $\rho_\mathsf{r} \prec \mathsf{T}_\mathsf{r} \ltimes_\mathsf{r} \Upsilon' \prec \mathsf{G}'$. If G_k is checkpointed we must have $|\rho_\mathsf{r}| = |\Upsilon|$ and $\mathsf{T}_\mathsf{r} \leqslant \mathsf{G} \upharpoonright \mathsf{r}$. In fact otherwise $\mathsf{T}' \leqslant \mathsf{G}_k \upharpoonright \mathsf{r}$ would imply $\mathsf{T}' = \blacktriangle_B \mathsf{T}''$, where B is the name of the checkpoint of G_k. We would get $\mathsf{T}_\mathsf{r} = \blacktriangle_A \blacktriangle_B \mathsf{T}''$ and this is not a session type according to our syntax.

Case (3). Let $\Upsilon = \Upsilon' \cdot \mathsf{G}' \cdot \Upsilon''$ and A be the name of the checkpoint of G'. Lemma 4.1(4) implies that $\vdash \mathsf{p}_l \triangleleft \mathbb{C}_{\mathsf{p}_l} : \rho_{\mathsf{p}_l} \prec \mathsf{T}_{\mathsf{p}_l}$ and $\rho_{\mathsf{p}_l} \prec \mathsf{T}_{\mathsf{p}_l} \ltimes_{\mathsf{p}_l} \Upsilon \prec \mathsf{G}$ for all $l \in I \cup J$. If $|\rho_{\mathsf{p}_l}| \leq |\Upsilon'|$, then $A \notin (\mathbb{C}_{\mathsf{p}_l})$ by Definition 3.1(2) and Definition 3.3. This implies $l \in J$ and $\rho_{\mathsf{p}_l} \prec \mathsf{T}_{\mathsf{p}_l} \ltimes_{\mathsf{p}_l} \Upsilon' \prec \mathsf{G}'$. Otherwise $l \in I$ and $\rho_{\mathsf{p}_l} = \rho'_{\mathsf{p}_l} \cdot \mathsf{T}_l \cdot \rho''_{\mathsf{p}_l}$ and $\mathsf{T}_l \leqslant \mathsf{G}' \upharpoonright \mathsf{p}_l$ by condition 1 of Definition 3.3. Let $\mathbb{C}_{\mathsf{p}_i} = R_{\mathsf{p}_i} \prec P_{\mathsf{p}_i}$. From $\vdash \mathsf{p}_i \triangleleft \mathbb{C}_{\mathsf{p}_i} : \rho_{\mathsf{p}_i} \prec \mathsf{T}_{\mathsf{p}_i}$ we get $\vdash R_{\mathsf{p}_i} : \rho'_{\mathsf{p}_i} \cdot \mathsf{T}_i \cdot \rho''_{\mathsf{p}_i}$ by Lemma 4.1(3). Then $R_{\mathsf{p}_i} = R'_{\mathsf{p}_i} \cdot P_i \cdot R''_{\mathsf{p}_i}$ and $\vdash P_i : \mathsf{T}_i$ by Lemma 4.1(2). The name of the checkpoint of P_i is A since $\mathsf{T}_i \leqslant \mathsf{G}' \upharpoonright \mathsf{p}_i$ and then $\mathbb{C}'_{\mathsf{p}_l} = R'_{\mathsf{p}_i} \prec P_i$. This implies $\rho_{\mathsf{p}_i} \prec \mathsf{T}_{\mathsf{p}_i} \ltimes_{\mathsf{p}_i} \Upsilon' \prec \mathsf{G}'$. We can then derive $\vdash \mathbb{M}' : \Upsilon' \prec \mathsf{G}'$ by rule [T-M].

From the proof of the previous theorem we get the following properties of well-typed networks, which are usual for session calculi [13]. We say that an application of the reduction rule [COM] has a *type mismatch*, if there is no sort that can be derived both for the communicated value and for variable associated to the communicated label in the input process.

Global types describe interaction protocols. The communications in well-typed networks evolve following the exact order of the associated global types.

Theorem 4.4 (Session Fidelity) *If $\vdash \mathbb{N}$ ✓, then reducing $\mathbb{N}$*

1. there is never a type mismatch;
2. the communications occur in the order prescribed by global types.

Notably property 1 holds in spite of the fact that session participants may exchange messages of different types. Property 2 says that session participants behave according to established communication protocols.

The standard definition of progress only ensures absence of deadlocks [21, Section 8.3]. Progress for session calculi means that all the requested interactions may happen [5]. In reversible sessions it is also natural to guarantee that all possible rollbacks may take place. This leads us to the following formulation of the progress theorem.

Theorem 4.5 (Progress) *If* $\vdash \mathbb{N}\ \checkmark$*, then:*

1. *if* $\mathbb{N}$ *contains an input or output process, then* $\mathbb{N}$ *forward reduces to* $\mathbb{N}'$ *and that input or output prefix does not occur in* $\mathbb{N}'$*;*
2. *if* $\mathbb{N}$ *contains a checkpoint named A, then there is a reduction of* $\mathbb{N}$ *in which the last step is a rollback making the processes checkpointed by A active processes.*

Proof As proved in [5], a single multiparty session in a standard calculus with global and session types, like the calculus in [13], always enjoys progress whenever it is well typed. In fact, by the Subject Reduction Theorem (Theorem 4.3), reduction preserves well-typedness of sessions. Moreover, all required session participants are present, as ensured by the condition $\mathsf{pt}(\Upsilon) \cup \mathsf{pt}(\mathsf{G}) \subseteq \{\mathsf{p}_i \mid i \in I\}$ in the premise of rule [T-M]. Thus, all communications among participants in a unique session will take place, in the order prescribed by the single-threaded active global type. This ensures that property 1 holds. For property 2 observe that, if $\mathbb{N}$ contains checkpoints named A, then there is at least one multiparty session $\mathbb{M}$ in $\mathbb{N}$ which is typed by a global type $\Upsilon \prec \mathsf{G}$ which contains A. If A occurs in G, then we can reduce forward $\mathbb{M}$ until the processes checkpointed by A will be all in the checkpointed sequences, and then apply the desired rollback. If A does not occur in G let $\Upsilon = \Upsilon' \cdot {}_{\blacktriangle_B}\mathsf{G}' \cdot \Upsilon''$ and A occurs in ${}_{\blacktriangle_B}\mathsf{G}'$. Then the checkpointed sequences of $\mathbb{M}$ have processes checkpointed by B. We can then apply the rollback which makes the processes checkpointed by B to become active processes. If $A = B$ we are done. Otherwise the active global type of the obtained session contains A and we can conclude as in previous case.

5 Related Work and Conclusions

Since the pioneering work by Danos and Krivine [6], reversible computations in process algebras have been widely studied. The calculus of [6] adds a distributed monitoring system to CCS [18] allowing computations to be rewound. Phillips and Ulidowski [20] propose a method for reversing process operators that are definable by SOS rules in a general format, using keys to bind synchronised actions together. A reversible variant of the higher-order π-calculus is defined in [16], using name tags for identifying threads and explicit memory processes. In [15], Lanese et al. enrich the calculus of [16] with a fine-grained rollback primitive. To the best of our knowledge the earliest work dealing with rollback of communicating systems are [7, 8, 14]. In these papers an extension of CCS models the combination of rollback recovery and coordinated checkpoints.

As pointed out in [20], reversibility in process calculi is challenging, since we cannot distinguish between the processes $a\|a$ and $a.a$ by simply recording the past actions. For this reason both histories and unique identifiers for threads have been used to track information. We do not have this problem in our calculus since each session participant reduces in a sequential way. A key requirement, dubbed *causal consistency* in [6], is that of undoing actions only if no other action depending on them has been executed (and not undone). In the present work causal consistency follows from the linearity of the interactions described by single-threaded global types.

The most widely used models of structured communication-based programming are session behaviours [3, 4] and session calculi [12, 13]. Reversibility has been incorporated into both these models.

Compliance and sub-behaviour for session behaviours with checkpoints has been first studied in [1]. There a process has the possibility, after a rollback, of resuming the computation along the very same branch of the computation on which the rollback has been performed. From a different point of view, instead, rollbacks could be used as a strategy to get compliance. For instance assuming the interacting processes to roll back whenever the current branch of the computation cannot proceed and a different branch could work instead. This approach has been investigated in [2].

The papers closer to ours are [22, 23, 17]. Tiezzi and Yoshida [22] use tags and memories to allow reversibility of binary sessions with delegation. Reversibility is full, i.e. each interaction can be undone and causal consistency is preserved. An extension of this calculus allows computation to go forward and backward until the session is committed by means of a specific irreversible action. Only processes are typed, but this is enough to ensure absence of errors. Two forms of reversibility are considered in [23]. Either a session can be completely reversed with one backward step, or any intermediate state can be restored with either one backward step or multiple ones. In the first case the memory is just the initial process, while in the second case the sequence of all the processes generated by the reduction is needed. Both binary and multiparty sessions are taken into account under the hypothesis that they are "single". A session is single when all participants interact only along that session. Mezzina and Pérez [17] use monitors as memories. A key novelty of [17] are session types with present and past, which allow the semantics of reversible actions to be streamlined.

The main contributions of this paper are the treatment of checkpointed interactions and the role played by global types in controlling reversibility. In defining of our calculus, we made some design choices. For simplicity, we did not consider

- session initialisation by means of request/accept,
- subsorting and covariance/contravariance of messages types in the subtyping of session types,
- asynchronous communications using message queues.

Including these features, which are present in [13, 11, 23], would be easy.
Moreover we did the following assumptions:

- the rollback to a checkpoint is done non-deterministically and simultaneously by all participants which traversed that checkpoint,
- all the communications can be undone.

In future work we plan to address the issue of communication that cannot be undone, such as "money dispensed by an ATM machine", and also add to the process language primitives triggering the rollback. In our calculus, when crossing a checkpoint we memorise all the branches of the choice. Including only the branches not taken, would get us also checkpointed single inputs/outputs, and this would require some care.

We will also study rollbacks with checkpoints for interleaved multiparty sessions with delegation. In this case, a crucial point is the dependency between different sessions when backward reductions are done, see [22].

Acknowledgments. We are grateful to the anonymous reviewers for their useful suggestions, which led to substantial improvements.

References

[1] Franco Barbanera, Mariangiola Dezani-Ciancaglini & Ugo de'Liguoro (2016): *Reversible client/server interactions*. *Formal Aspects of Computing* 28(4), pp. 697–722, doi:10.1007/s00165-016-0358-2.

[2] Franco Barbanera, Mariangiola Dezani-Ciancaglini, Ivan Lanese & Ugo de' Liguoro (2016): *Retractable Contracts*. In: *PLACES*, *EPTCS* 203, pp. 61–72, doi:10.4204/EPTCS.203.

[3] Franco Barbanera & Ugo de' Liguoro (2015): *Sub-behaviour relations for session-based client/server systems*. Mathematical Structures in Computer Science 25(6), pp. 1339–1381, doi:10.1017/S096012951400005X.

[4] Giovanni Bernardi & Matthew Hennessy (2016): *Modelling session types using contracts*. Mathematical Structures in Computer Science 26(3), pp. 510–560, doi:10.1017/S0960129514000243.

[5] Mario Coppo, Mariangiola Dezani-Ciancaglini, Nobuko Yoshida & Luca Padovani (2016): *Global Progress for Dynamically Interleaved Multiparty Sessions*. Mathematical Structures in Computer Science 26(2), pp. 238–302, doi:10.1017/S0960129514000188.

[6] Vincent Danos & Jean Krivine (2004): *Reversible Communicating Systems*. In: *CONCUR*, *LNCS* 3170, Springer, pp. 292–307, doi:10.1007/978-3-540-28644-8_19.

[7] Edsko de Vries, Vasileios Koutavas & Matthew Hennessy (2010): *Communicating Transactions - (Extended Abstract)*. In: *CONCUR*, *LNCS* 6269, Springer, pp. 569–583, doi:10.1007/978-3-642-15375-4_39.

[8] Edsko de Vries, Vasileios Koutavas & Matthew Hennessy (2010): *Liveness of Communicating Transactions - (Extended Abstract)*. In: *APLAS*, *LNCS* 6461, Springer, pp. 392–407, doi:10.1007/978-3-642-17164-2_27.

[9] Pierre-Malo Deniélou & Nobuko Yoshida (2011): *Dynamic Multirole Session Types*. In: *POPL*, ACM Press, pp. 435–446, doi:10.1145/1926385.1926435.

[10] Mariangiola Dezani-Ciancaglini, Silvia Ghilezan, Svetlana Jaksic, Jovanka Pantovic & Nobuko Yoshida (2016): *Precise subtyping for synchronous multiparty sessions*. In: *PLACES*, *EPTCS* 203, pp. 29–43, doi:10.4204/EPTCS.203.3.

[11] Simon Gay & Malcolm Hole (2005): *Subtyping for Session Types in the Pi Calculus*. Acta Informatica 42(2/3), pp. 191–225, doi:10.1007/s00236-005-0177-z.

[12] Kohei Honda, Vasco T. Vasconcelos & Makoto Kubo (1998): *Language Primitives and Type Disciplines for Structured Communication-based Programming*. In: *ESOP*, *LNCS* 1381, Springer, pp. 22–138, doi:10.1007/BFb0053567.

[13] Kohei Honda, Nobuko Yoshida & Marco Carbone (2008): *Multiparty Asynchronous Session Types*. In: *POPL*, ACM Press, pp. 273–284, doi:10.1145/1328897.1328472.

[14] Vasileios Koutavas, Carlo Spaccasassi & Matthew Hennessy (2014): *Bisimulations for Communicating Transactions - (Extended Abstract)*. In: *FOSSACS*, *LNCS* 8412, Springer, pp. 320–334, doi:10.1007/978-3-642-54830-7_21.

[15] Ivan Lanese, Claudio Antares Mezzina, Alan Schmitt & Jean-Bernard Stefani (2011): *Controlling Reversibility in Higher-Order Pi*. In: *CONCUR*, *LNCS* 6901, Springer, pp. 297–311, doi:10.1007/978-3-642-23217-6_20.

[16] Ivan Lanese, Claudio Antares Mezzina & Jean-Bernard Stefani (2010): *Reversing Higher-Order Pi*. In: *CONCUR*, *LNCS* 6269, Springer, pp. 478–493, doi:10.1007/978-3-642-15375-4_33.

[17] Claudio A. Mezzina & Jorge A. Pérez (2016): *Reversible Sessions Using Monitors*. In: *PLACES*, *EPTCS* 211, pp. 56–64, doi:10.4204/EPTCS.211.6.

[18] Robin Milner (1989): *Communication and concurrency*. PHI Series in computer science, Prentice Hall.

[19] Luca Padovani (2011): *Session Types = Intersection Types + Union Types*. In: *ITRS*, *EPTCS* 45, pp. 71–89, doi:10.4204/EPTCS.45.6.

[20] Iain C. C. Phillips & Irek Ulidowski (2007): *Reversing algebraic process calculi*. Journal of Logic and Algebraic Methods in Programming 73(1-2), pp. 70–96, doi:10.1016/j.jlap.2006.11.002.

[21] Benjamin C. Pierce (2002): *Types and Programming Languages*. MIT Press.

[22] Francesco Tiezzi & Nobuko Yoshida (2015): *Reversible Session-Based Pi-Calculus*. Journal of Logical and Algebraic Methods in Programming 84(5), pp. 684–707, doi:10.1016/j.jlamp.2015.03.004.

[23] Francesco Tiezzi & Nobuko Yoshida (2016): *Reversing Single Sessions*. In: *RC*, *LNCS* 9720, Springer, pp. 52–69, doi:10.1007/978-3-319-40578-0_4.