A Calculus for Collective-Adaptive Systems and its Behavioural Theory[☆]

Yehia Abd Alrahman[a], Rocco De Nicola[b], Michele Loreti[c]

[a] *University of Gothenburg | Chalmers University of Technology, Gothenburg, Sweden*
[b] *IMT School for Advanced Studies, Lucca, Italy*
[c] *Università di Camerino, Camerino, Italy*

Abstract

We propose a process calculus, named AbC, to study the behavioural theory of interactions in collective-adaptive systems by relying on attribute-based communication. An AbC system consists of a set of parallel components each of which is equipped with a set of attributes. Communication takes place in an implicit multicast fashion, and interaction among components is dynamically established by taking into account "connections" as determined by predicates over their attributes. The structural operational semantics of AbC is based on *Labeled Transition Systems* that are also used to define bisimilarity between components. Labeled bisimilarity is in full agreement with a barbed congruence, defined by relying on simple basic observables and context closure. The introduced equivalence is used to study the expressiveness of AbC in terms of encoding aspects of broadcast channel-based interactions and to establish formal relationships between system descriptions at different levels of abstraction.

Keywords: Collective-adaptive systems, Attribute-Based Communication, Process calculus, Operational semantics, Behavioural theory

1. Introduction

Collective-adaptive systems (CAS) [1] are new emerging computational systems, consisting of a large number of components, featuring complex interaction mechanisms. These systems are usually distributed, heterogeneous, decentralised and interdependent, and are operating in dynamic and often unpredictable environments. CAS components combine their behaviours, by forming collectives, to achieve specific goals depending on their attributes, objectives, and functionalities. CAS are inherently scalable and their boundaries are fluid in the sense that components may enter or leave the collective at any time; so they need to dynamically adapt to their environmental conditions and contextual data. New engineering techniques to address the challenges of developing, integrating, and deploying such systems are needed [2].

Most of the current communication models cannot naturally model highly adaptive and loosely-coupled systems with fluid boundaries like CAS. They actually suffer from limitations due to: their lack of knowledge representation, e.g., π-calculus [3], rigid communication interfaces, e.g., CBS [4]; and/or due to their lack of scalability to coordinate large number of agents in distributed settings, e.g., Psi-calculus [5], the fusion calculus [6], and the concurrent constraint programming [7, 8, 9].

For instance in π-calculus and Actors [10] communication links are established among distributed components by relying on channel-names or unique identities that are totally independent of the run-time properties, status, and capabilities of components. This makes it hard to program, coordinate, and adapt complex behaviours that highly depend on the actual status of components. Furthermore, CBS-like calculi

[☆] Yehia Abd Alrahman is funded by the ERC consolidator grant D-SynMA under the European Union's Horizon 2020 research and innovation programme (grant agreement No 772459).

Email addresses: yehia.abd.alrahman@gu.se (Yehia Abd Alrahman), rocco.denicola@imtlucca.it (Rocco De Nicola), michele.loreti@unicam.it (Michele Loreti)

with fixed communication structures cannot deal with reconfiguration and dynamic creation of communication links. A way to mitigate the previous shortcomings is to rely on Psi, fusion or concurrent constraint calculi that can easily deal with adaptation by using a sort of channel-equivalence which dynamically establishes communication links based on the combined knowledge of the communicating agents. However, these models are based on point-to-point communication and this leads to a critical problem of how to discover and coordinate a large number of agents or services in a flexible manner during their lifecycle. Point-to-point communication does not scale well when large number of agents need to coordinate their behaviour frequently. The problem gets more serious when considering open systems where agents may join or leave at any time without disrupting the overall system behaviour.

In this article, we address the challenge of finding a novel approach that naturally accommodates the above mentioned concepts of CAS systems, while simplifying the meta-theory. We introduce a calculus, named AbC that concentrates on primitives and interaction mechanisms that are crucial for dealing with CAS. AbC provides an explicit notion of knowledge representation, termed *attribute environment*; supports dynamic and run-time reconfigurable communication links by means of *interaction predicates*; and relies on a notion of *implicit and non-blocking multicast communication*. These ingredients constitute what we call *attribute-based communication*, a novel paradigm that enables groups of partners to interact by considering the predicates over the (dynamic) values of the attributes they expose. An AbC system is rendered as a set of parallel components, each equipped with a set of attributes, termed as attribute environment, and with a behaviour, termed as a process. The attribute values can be modified by internal actions and the behaviour of a component is parametrised to these values. AbC components communicate anonymously in an implicit multicast fashion without any prior agreement.

Interaction in AbC relies on two prefixing actions:

- $(\tilde{E})@\Pi$ is the *attribute-based send* that is used to send the values of the sequence of expressions $\tilde{E}$ to those components whose attributes satisfy predicate Π;
- $\Pi(\tilde{x})$ is the *attribute-based receive* that binds to the sequence $\tilde{x}$ the values received from any component whose attributes (and possibly transmitted values) satisfy the predicate Π.

Receiving operations are blocking while sending operations are not. This breaks synchronisation dependencies between interacting partners, and permits modelling systems where interacting partners can enter or leave a group at any time without disturbing its overall behaviour. Groups are dynamically formed at the time of interaction by means of available and interested receiving components that satisfy sender's predicates. In this way, run-time changes of attributes introduce opportunistic interactions between components.

We demonstrate the expressive power of AbC by showing how it can be used to encode different communication paradigms and we also provide a uniform encoding of a broadcast channel-based process calculus into AbC. We conjecture that the converse is not possible.

The operational semantics of AbC is given in terms of a labelled transition system (LTS) that is also used as the basis for defining a notion of bisimulation-based equivalence over AbC components. We first introduce a context-based reduction barbed congruence by using very simple observables only testing predicates and then we provide the corresponding extensional labelled bisimilarity. We show how to use the introduced bisimilarity to establish formal relationships between systems at different level of abstractions. We also prove the correctness of encoding a process calculus (inspired by CBS [4] and based on broadcast channels [11]) into AbC up to our proposed equivalences.

This article is an extended and revised version of the conference paper presented in [12]. Here, we extend the calculus with operators for controlling visibility of actions, extend the behavioural theory and provide equational laws for it. Moreover, we add explanatory examples and provide full proofs of all results. The scope of this paper is focused on the theoretical aspects of our calculus while those aspects concerned with programming methodologies are reported in a companion paper [13]; there we show how to program complex and challenging scenarios, featuring collaboration, adaptation and reconfiguration in an intuitive way.

The rest of the paper is organised as follows. In Section 2 we formally present the syntax of AbC, while in Section 3 we introduce its operational semantics. In Section 4 we define a behavioural theory for AbC by introducing a barbed congruence and then an equivalent definition of a labelled bisimulation. Section 5

is used to introduce a number of equational laws. In Section 6, we illustrate the expressive power of AbC; we discuss how the calculus can be used to model other communication paradigms and prove correctness and completeness of an encoding of a channel-passing process calculus into AbC. Finally, in Section 7 and Section 8, we sum up our main contributions, relate our work to closely related state of arts and list research directions that deserve further investigation.

2. Syntax of the AbC Calculus

In this section we formally present the syntax of AbC and discuss the intuition behind the different operators we introduce. We make use of a running example to show how to use AbC primitives to model a collective scenario of a flock of drones, a slight variant and multiparty version of the one presented in [14].

In the rest of this article, we will use $\mathcal{V}$ to denote the set of *values* that can be used in an AbC system. Elements in $\mathcal{V}$ are denoted by b, v or n (sometime with indexes). Moreover, we will also use the notation $\tilde{\cdot}$ to denote a sequence of *elements* and $\{\tilde{\cdot}\}$ to indicate the set of elements in the sequence $\tilde{\cdot}$.

The syntax of the AbC calculus is reported in Table 1. The top-level entities of the calculus are *components* (C). A component, $\Gamma :_I P$, is a process P associated with an *attribute environment* Γ, and an *interface* I. An *attribute environment* $\Gamma : \mathcal{A} \twoheadrightarrow \mathcal{V}$ is a partial map from attribute identifiers[1] $a \in \mathcal{A}$ to values $v \in \mathcal{V}$ where $\mathcal{A} \cap \mathcal{V} = \emptyset$. A value could be a number, a name (string), a tuple, etc. We will use Env to denote a set of *attribute environments* Γ.

An *interface* $I \subseteq \mathcal{A}$ consists of a *finite* set of *attributes* that are exposed by a component to control the interactions with other components. We will refer to the attributes in I as *public attributes*, and to those in $dom(\Gamma) - I$ as *private attributes*.

Components are composed using the parallel operator $\|$, e.g., $C_1 \| C_2$.

Example 2.1 (A flock of drones (Step 1/5))**.** *We assume that each drone in a particular flock is equipped with a sensor that determines whether its battery-level is low or not, and each drone must trigger an adaptation mechanism (i.e., terminate the mission and return to the base station) when there exists at least five drones in the flock with low battery-level. Such decision cannot be determined locally based on one or on a subset of drones but rather globally. Thus, drones need to coordinate and combine their behaviour to take a decision. This is a typical example of CAS where a component infers a global knowledge of the system by means of interactions. We can use* AbC *to define a simple and distributed protocol ensuring that once drones start communicating, the adaptation mechanism of each drone may eventually be triggered.*

We assume that drones have unique identities id *and a limited memory that contains the following attributes: input* i*, indicating drones' battery-level* 1 *(if low) and* 0 *(otherwise); and a counter* c *to count the number of drones with low battery-level. We model each drone as an* AbC *component* $\Gamma :_I P$ *where* $I = \{\mathsf{id}, \mathsf{i}\}$*. Initially, drones are not aware of the identities of each others and only learn them at run-time through interactions to possibly establish dedicated links dynamically. Also the initial values of attributes in* Γ *are assigned as follows:* i *is provided by the sensor and* $\mathsf{c} = \mathsf{i}$ *(the counter is initially set to the value provided by the sensor). The flock of drones is the parallel composition of many drones:*

$$\Gamma_1 :_I P \parallel \Gamma_2 :_I P \parallel \ldots \parallel \Gamma_n :_I P$$

Note that drones execute the same code (i.e., P*) and collaborate to answer a global question. Since* P *is parametric with respect to* Γ_i*, different drones could exhibit different behaviours based on the values of their attributes.* □

To control the interactions of a component C, the *restriction* operators $[\ C\ ]^{\triangleleft f}$ and $[\ C\ ]^{\triangleright f}$ can be used. There f is a function associating a predicate Π to each tuple of values $\tilde{v} \in \mathcal{V}^*$ and attribute environment Γ. The operators $[\ C\ ]^{\triangleleft f}$ and $[\ C\ ]^{\triangleright f}$ can be used to restrict the messages that component C can receive and send, respectively. For instance, consider a set of parallel components C one of which has a public

[1] In the rest of this article, we shall occasionally use the term "attribute" instead of "attribute identifier".

(Components)	$C ::= \Gamma :_I P \quad \mid \quad C_1 \| C_2 \quad \mid \quad [\ C\ ]^{\triangleleft f} \quad \mid \quad [\ C\ ]^{\triangleright f}$
(Processes)	$P ::= \mathsf{0} \quad \mid \quad \Pi(\tilde{x}).U \quad \mid \quad (\tilde{E})@\Pi.U \quad \mid \quad \langle \Pi \rangle P \quad \mid \quad P_1 + P_2 \quad \mid \quad P_1 \mid P_2 \quad \mid \quad K(x_1, \ldots, x_n)$
(Updates)	$U ::= [a := E]U \quad \mid \quad P$
(Predicates)	$\Pi ::= \mathsf{tt} \mid \mathsf{ff} \quad \mid \quad p_k(E_1, \ldots, E_k) \quad \mid \quad \Pi_1 \wedge \Pi_2 \quad \mid \quad \Pi_1 \vee \Pi_2 \quad \mid \quad \neg \Pi$
(Expressions)	$E ::= v \quad \mid \quad x \quad \mid \quad a \quad \mid \quad this.a \quad \mid \quad o_k(E_1, \ldots, E_k)$

Table 1: The syntax of the AbC calculus

attribute environment Γ and sends $\tilde{v}$ to components satisfying Π. When the message outgoes $[\ C\ ]^{\triangleright f}$, the target predicate is updated to consider also predicate $\Pi' = f(\Gamma, \tilde{v})$, thus the components satisfying $\Pi \wedge \Pi'$ will receive the message. To prevent a *secret* s from being spread outside C, the following function can be used: $f_s(\Gamma, \tilde{v}) = \mathsf{tt}$ if $s \notin \tilde{v}$ and ff otherwise. Similarly, $[\ C\ ]^{\triangleleft f}$ can be used to limit the ability of C to receive messages. In particular, if a component with public attribute environment Γ sends a message $\tilde{v}$ to components C satisfying Π, only components in C satisfying $\Pi \wedge f(\Gamma, \tilde{v})$ are eligible to receive the message.

A *process* P can be the *inactive* process $\mathsf{0}$, an *action-prefixed* process, $act.U$, where act is a communication action and U is a process possibly preceded by an *attribute update*, a *self-aware* process $\langle \Pi \rangle P$, a *nodeterministic choice* between two processes $P_1 + P_2$, an *interleaving composition* of two processes $P_1 | P_2$, or a parametrised process call with a unique identifier K and a sequence of formal parameters $(x_1, \ldots, x_n)$ used in the process definition $K(x_1, \ldots, x_n) \triangleq P$. We require that x_i be pairwise distinct, i.e., $i \neq j \implies x_i \neq x_j$.

The *self-awareness construct*, $\langle \Pi \rangle P$, is used to trigger local behaviours (i.e., P) when the environment of the executing component is changed (i.e., $\Gamma \not\models \Pi$). It blocks the execution of P until predicate Π is satisfied given the attribute environment where the process $\langle \Pi \rangle P$ is executing. This construct is used to coordinate co-located processes where an attribute update of one process might influence the behaviour of another co-located one. Thus enabling modelling a notion of *interdependence* among co-located processes. The *parallel operator*, $P|Q$, models the interleaving between co-located processes, i.e., processes executing in the same component. Those processes can only communicate indirectly through inspecting the attribute environment they share.

We now explain what we mean by expressions and predicates, then we continue by describing the communication actions and their (possible) side-effects, i.e., attribute updates.

An *expression* E is built from constant values $v \in \mathcal{V}$, variables x, attribute identifiers a, a reference to the value of a ($this.a$) in the component that is executing the code, or through a standard operators $o_k(E_1, \ldots, E_k)$. The latter indicates a generic operator with k-arity over *values* in $\mathcal{V}$. For the sake of simplicity, we omit the specific syntax of operators used to build expressions, we will only assume for each o_k a (possibly partial) function $\mathcal{E}_{o_k} : \mathcal{V}^k \to \mathcal{V}$ describing the semantics of o_k. We will use $o(\tilde{E})$ when the value k does not play any role or it is clear from the context. The evaluation of expression E under Γ is denoted by $[\![E]\!]_\Gamma$. The definition of $[\![\cdot]\!]_\Gamma$ is standard, the only interesting cases are $[\![a]\!]_\Gamma = [\![this.a]\!]_\Gamma = \Gamma(a)$.

A *predicate* Π is built from boolean constants, tt and ff, from an *atomic predicate* $p_k(E_1, \ldots, E_k)$ and also by using standard boolean operators ($\neg$, $\wedge$ and $\vee$). The precise set of atomic predicates is not detailed here; we only assume that each p_k denotes a *decidable predicates* in $\mathcal{V}^k$, i.e. $p_k \subseteq \mathcal{V}^k$. Examples of *basic predicates* are the standard binary relations like $=, >, <, \leq, \geq$.

The *satisfaction relation* $\Gamma \models \Pi$ is formally defined in Table 2 and shows when an attribute environment Γ satisfies a predicate Π. In the rest of this paper we will use $\mathcal{M}(\Pi)$ to denote the set of *attribute environments* that satisfies Π, i.e., $\{\Gamma \mid \Gamma \models \Pi\}$. We also shall use the relation $\doteq$ to denote a semantic equivalence for predicates as defined below.

Definition 2.1 (Predicate Equivalences)**.** *Let* Π_1 *and* Π_2 *be two predicates, we have that:*

- $\Pi_1 \Rightarrow \Pi_2$ *if and only if* $\mathcal{M}(\Pi_1) \subseteq \mathcal{M}(\Pi_2)$*;*
- $\Pi_1 \doteq \Pi_2$ *if and only if* $\Pi_1 \Rightarrow \Pi_2$ *and* $\Pi_2 \Rightarrow \Pi_1$*.*

$\Gamma \models \mathsf{tt}$ for all Γ
$\Gamma \not\models \mathsf{ff}$ for all Γ
$\Gamma \models p_k(E_1, \ldots, E_k)$ iff $([\![E_1]\!]_\Gamma, \ldots, [\![E_k]\!]_\Gamma) \in p_k$
$\Gamma \models \Pi_1 \wedge \Pi_2$ iff $\Gamma \models \Pi_1$ and $\Gamma \models \Pi_2$
$\Gamma \models \Pi_1 \vee \Pi_2$ iff $\Gamma \models \Pi_1$ or $\Gamma \models \Pi_2$
$\Gamma \models \neg\Pi$ iff not $\Gamma \models \Pi$

Table 2: The predicate satisfaction

In what follows, we shall use the notation $\{\Pi\}_\Gamma$ to indicate the *closure* of predicate Π under the attribute environment Γ; it yields a new predicate Π' obtained from Π after replacing each occurrence of $this.a$ with its value $\Gamma(a)$. Note that, attribute identifiers occurring in an *equality operator* will also occur in its closure, e.g., for an attribute identifier a, we have that $\{a = v\}_\Gamma$ is equivalent to $(a = v)$ while $\{a = this.a\}_\Gamma$ is equivalent to $(a = v)$ given that $\Gamma(a) = v$.

Example 2.2 (A flock of drones (Step 2/5))**.** *If we further specify the structure of process P in Example 2.1, we have that P is defined as the interleaving of three processes: R, A, and T:*

$$P \triangleq R \mid A \mid T \quad \textit{where} \quad R \triangleq b.0, \qquad A \triangleq \langle this.\mathsf{c} \geq 5\rangle M, \qquad T \triangleq (r_1.T_1 + r_2.T_2 + r_3.T_3 + r_4.T_4) \mid T$$

We use b to denote a broadcast action and r_i to denote an input action. We will further specify b and r_i in the next step. Process R broadcasts a request b to drones with low battery-level and terminates; process A blocks until the counter is "≥ 5" and then activates the adaptation process M; and process T handles message reception and non-deterministically selects a branch depending on the received message. Note that T replicates itself each time a message is received to guarantee that drones do not lose messages. □

The *attribute-based output* $(\tilde{E})@\Pi$ is used to send the evaluation of the sequence of expressions $\tilde{E}$ to the components whose attributes satisfy the predicate Π.

The *attribute-based input* $\Pi(\tilde{x})$ is used to receive messages from any component whose attributes (and possibly transmitted values) satisfy the predicate Π; the sequence $\tilde{x}$ acts as a placeholder for received values. Note that the receiving predicates, used in attribute-based input actions, can also refer to variables in $\tilde{x}$ and the received values can be used to check whether specific conditions are satisfied. For instance, the action

$$((x = \text{“}\mathsf{Req}\text{”}) \wedge (y + this.\mathsf{c} \geq 5) \wedge (id \neq -1))(x, y, z)$$

can be used to receive a message like $(\text{“}\mathsf{Req}\text{”}, 2, 3)$ where the sum of the value received on y (i.e, 2) and $this.\mathsf{c}$ is greater or equal 5 and the value of the attribute id of the sending component is different from -1. Thus, the predicate can be used to check both the received values and the exposed values of the sender attributes. Note that while the values of exposed attributes are used by the receiver to select a sender, message values are used to distinguish different incoming messages from the same sender. This also suggests that the interface is redundant and can be encoded using a structured message where the values of the attributes in the interface are considered as additional values similar to our early version [12]. Although the two approaches are equivalent, we prefer to cleanly separate values from public attributes. This also suggests that there is no need to have dynamic interfaces because structured messages serve the purpose.

A predicate can also refer to *local* attributes of components. Thus, an action like

$$(\text{“}\mathsf{Req}\text{”}, 1, 3)@(\mathsf{i} \geq this.\mathsf{i})$$

can be used to send the message $(\text{“}\mathsf{Req}\text{”}, 1, 3)$ to all components whose attribute i is not less than $this.\mathsf{i}$.

An *attribute update*, $[a := E]$, is used to assign the result of the evaluation of E to the attribute identifier a. The syntax is devised in such a way that sequences of updates are only possible after communication actions. Actually, updates can be viewed as side effects of interactions. It should be noted that the execution of a communication action and the following update(s) is atomic. This possibility allows components to modify their attribute values and thus triggering new behaviours in response to collected contextual data.

Free and bound variables. Input action $\Pi(\tilde{x})$ acts as binder $\tilde{x}$ in $\Pi(\tilde{x}).U$. We use $bv(P)$ and $fv(P)$ to denote the set of bound and free variables of P, respectively.

Now everything is precisely defined and we can fully specify the running example:

Example 2.3 (A flock of drones (Step 3/5))**.** *Processes R, A, and T in Example 2.2 can be specified as follows:*

$$
\begin{aligned}
R &\triangleq (\text{“Req”}, this.\mathsf{i}, this.\mathsf{id})@(\mathsf{i} \neq 0).0 \\
A &\triangleq \langle this.\mathsf{c} \geq 5 \rangle M \\
T &\triangleq (\quad ((x = \text{“Req”}) \wedge (y + this.\mathsf{c} \geq 5))(x, y, z). \\
&\qquad (\text{“Ack”}, this.\mathsf{i}, -1)@(\mathsf{id} = z).[this.\mathsf{c} := 5]0 \\
&\quad + ((x = \text{“Req”}) \wedge (y + this.\mathsf{c} < 5))(x, y, z). \\
&\qquad (\text{“Ack”}, this.\mathsf{i}, -1)@(\mathsf{id} = z).[this.\mathsf{c} := this.\mathsf{c} + y]0 \\
&\quad + ((x = \text{“Ack”}) \wedge (y + this.\mathsf{c} \geq 5))(x, y, z).[this.\mathsf{c} := 5]0 \\
&\quad + ((x = \text{“Ack”}) \wedge (y + this.\mathsf{c} < 5))(x, y, z).[this.\mathsf{c} := this.\mathsf{c} + y]0 \\
&\quad)| \; T
\end{aligned}
$$

Process R broadcasts a request for all drones with low battery-level, i.e., such that $\mathsf{i} = 1$ *and terminates. The request contains the drone battery-level* $this.\mathsf{i}$ *and its identity* $this.\mathsf{id}$*. Process A blocks the adaptation process M until the value of the counter is greater or equal* 5 *in which case M is triggered.*

The first two branches in T are selected when a request is received and the other two are for acknowledgement. If a request is received where the sum of the drone's counter and the value received on y is greater or equal to 5*, the counter is set to* 5 *otherwise the counter is incremented by the value received on y and in both cases an acknowledgement is sent to the requester. If an acknowledgement is received, where the sum of the drone's counter and the value received on y is greater or equal to* 5*, the counter is set to* 5 *otherwise the counter is incremented by the value received on y. Note that T replicates itself every time a message is received. Note that process T controls A through the value of the counter attribute.* □

3. AbC Operational Semantics

The operational semantics of AbC is based on two relations. The transition relation $\mapsto$ that describes the behaviour of individual components and the transition relation $\longrightarrow$ that relies on $\mapsto$ and describes system behaviour.

3.1. Operational semantics of components

We use the transition relation $\mapsto \subseteq \text{Comp} \times \text{CLAB} \times \text{Comp}$ to define the local behaviour of a component where Comp denotes the set of components and CLAB is the set of transition labels, α, generated by the following grammar:

$$\alpha ::= \lambda \quad | \quad \widetilde{\Gamma \triangleright \Pi(\tilde{v})} \qquad\qquad \lambda ::= \Gamma \triangleright \overline{\Pi}(\tilde{v}) \quad | \quad \Gamma \triangleright \Pi(\tilde{v})$$

The λ-labels are used to denote AbC output $\Gamma \triangleright \overline{\Pi}(\tilde{v})$ and input $\Gamma \triangleright \Pi(\tilde{v})$ actions. The former contains the sender's predicate Π, that specifies the expected interacting partners, the transmitted values $\tilde{v}$, and the portion of the sender *attribute environment* Γ that can be perceived by receivers. The latter label is just the complementary label selected among all the possible ones that the receiver may accept.

The α-labels include an additional label $\widetilde{\Gamma \triangleright \Pi(\tilde{v})}$ to model the case where a component is not able to receive a message. As it will be seen later, this kind of *negative* label is crucial to appropriately handle dynamic operators like choice and awareness.

The transition relation $\mapsto$ is defined in Table 3 and Table 4 inductively on the syntax of Table 1. For each process operator we have two types of rules: one describing the actions a term can perform, the other one showing how a component discards undesired input messages.

$$\frac{[\![\tilde{E}]\!]_\Gamma = \tilde{v} \quad \{\Pi_1\}_\Gamma = \Pi}{\Gamma :_I (\tilde{E})@\Pi_1.U \xmapsto{\Gamma \downarrow I \triangleright \overline{\Pi}(\tilde{v})} \{\!|\Gamma :_I U|\!\}}\ \textsc{Brd} \qquad \frac{}{\Gamma :_I (\tilde{E})@\Pi.U \xmapsto{\widetilde{\Gamma' \triangleright \Pi'(\tilde{v})}} \Gamma :_I (\tilde{E})@\Pi.U}\ \textsc{FBrd}$$

$$\frac{\Gamma' \models \{\Pi_1[\tilde{v}/\tilde{x}]\}_{\Gamma_1} \quad \Gamma_1 \downarrow I \models \Pi}{\Gamma_1 :_I \Pi_1(\tilde{x}).U \xmapsto{\Gamma' \triangleright \Pi(\tilde{v})} \{\!|\Gamma_1 :_I U[\tilde{v}/\tilde{x}]|\!\}}\ \textsc{Rcv} \qquad \frac{\Gamma' \not\models \{\Pi[\tilde{v}/\tilde{x}]\}_\Gamma \vee \Gamma_1 \downarrow I \not\models \Pi'}{\Gamma_1 :_I \Pi(\tilde{x}).U \xmapsto{\widetilde{\Gamma' \triangleright \Pi'(\tilde{v})}} \Gamma_1 :_I \Pi(\tilde{x}).U}\ \textsc{FRcv}$$

$$\frac{\Gamma \models \Pi \quad \Gamma :_I P \xmapsto{\lambda} \Gamma' :_I P'}{\Gamma :_I \langle \Pi \rangle P \xmapsto{\lambda} \Gamma' :_I P'}\ \textsc{Aware} \qquad \frac{\Gamma \not\models \Pi}{\Gamma :_I \langle \Pi \rangle P \xmapsto{\widetilde{\Gamma' \triangleright \Pi'(\tilde{v})}} \Gamma :_I \langle \Pi \rangle P}\ \textsc{FAware1}$$

$$\frac{\Gamma \models \Pi \quad \Gamma :_I P \xmapsto{\widetilde{\Gamma' \triangleright \Pi'(\tilde{v})}} \Gamma :_I P}{\Gamma :_I \langle \Pi \rangle P \xmapsto{\widetilde{\Gamma' \triangleright \Pi'(\tilde{v})}} \Gamma :_I \langle \Pi \rangle P}\ \textsc{FAware2}$$

Table 3: Operational Semantics of Components (Part 1)

$$\frac{\Gamma :_I P_1 \xmapsto{\lambda} \Gamma' :_I P_1'}{\Gamma :_I P_1 + P_2 \xmapsto{\lambda} \Gamma' :_I P_1'}\ \textsc{SumL} \qquad \frac{\Gamma :_I P_2 \xmapsto{\lambda} \Gamma' :_I P_2'}{\Gamma :_I P_1 + P_2 \xmapsto{\lambda} \Gamma' :_I P_2'}\ \textsc{SumR}$$

$$\frac{\Gamma :_I P_1 \xmapsto{\widetilde{\Gamma' \triangleright \Pi'(\tilde{v})}} \Gamma :_I P_1 \quad \Gamma :_I P_2 \xmapsto{\widetilde{\Gamma' \triangleright \Pi'(\tilde{v})}} \Gamma :_I P_2}{\Gamma :_I P_1 + P_2 \xmapsto{\widetilde{\Gamma' \triangleright \Pi'(\tilde{v})}} \Gamma :_I P_1 + P_2}\ \textsc{FSum}$$

$$\frac{\Gamma :_I P_1 \xmapsto{\lambda} \Gamma' :_I P'}{\Gamma :_I P_1 \mid P_2 \xmapsto{\lambda} \Gamma' :_I P' \mid P_2}\ \textsc{IntL} \qquad \frac{\Gamma :_I P_2 \xmapsto{\lambda} \Gamma' :_I P'}{\Gamma :_I P_1 \mid P_2 \xmapsto{\lambda} \Gamma' :_I P_1 \mid P'}\ \textsc{IntR}$$

$$\frac{\Gamma :_I P_1 \xmapsto{\widetilde{\Gamma' \triangleright \Pi'(\tilde{v})}} \Gamma :_I P_1 \quad \Gamma :_I P_2 \xmapsto{\widetilde{\Gamma' \triangleright \Pi'(\tilde{v})}} \Gamma :_I P_2}{\Gamma :_I P_1 \mid P_2 \xmapsto{\widetilde{\Gamma' \triangleright \Pi'(\tilde{v})}} \Gamma :_I P_1 \mid P_2}\ \textsc{FInt}$$

$$\frac{\Gamma :_I P[\tilde{v}/\tilde{x}] \xmapsto{\lambda} \Gamma' :_I P' \quad K(\tilde{x}) \triangleq P}{\Gamma :_I K(\tilde{v}) \xmapsto{\lambda} \Gamma' :_I P'}\ \textsc{Rec} \qquad \frac{\Gamma :_I P[\tilde{v}_1/\tilde{x}] \xmapsto{\widetilde{\Gamma' \triangleright \Pi'(\tilde{v})}} \Gamma :_I P' \quad K(\tilde{x}) \triangleq P}{\Gamma :_I K(\tilde{v}_1) \xmapsto{\widetilde{\Gamma' \triangleright \Pi'(\tilde{v})}} \Gamma :_I K(\tilde{v}_1)}\ \textsc{FRec}$$

$$\frac{}{\Gamma :_I \mathsf{0} \xmapsto{\widetilde{\Gamma' \triangleright \Pi'(\tilde{v})}} \Gamma :_I \mathsf{0}}\ \textsc{FZero}$$

Table 4: Operational Semantics of Components (Part 2)

The behaviour of an *attribute-based output* is defined by rule Brd in Table 3. This rule states that when an output is executed, the sequence of expressions $\tilde{E}$ is evaluated, say to $\tilde{v}$, and the *closure* Π of predicate Π_1 under Γ is computed. Hence, these values are sent to other components together with $\Gamma \downarrow I$. This represents the portion of the *attribute environment* that can be perceived by the context and it is obtained from the local Γ by limiting its domain to the attributes in the interface I as defined below:

$$(\Gamma \downarrow I)(a) = \begin{cases} \Gamma(a) & a \in I \\ \bot & \text{otherwise} \end{cases}$$

As the usual treatment of partial functions in denotational semantics, the element $\bot$ is returned when $\Gamma(a)$ is undefined.

Possible updates U, following the action, are applied. This is expressed in terms of a recursive function $\{\!|C|\!\}$ defined below:

$$\{\!|C|\!\} = \begin{cases} \{\!|\, \Gamma[a \mapsto [\![E]\!]_\Gamma] :_I U \,|\!\} & C = \Gamma :_I [a := E]U \\ \Gamma :_I P & C = \Gamma :_I P \end{cases}$$

where $\Gamma[a \mapsto v]$ denotes an attribute update such that $\Gamma[a \mapsto v](a') = \Gamma(a')$ if $a \neq a'$ and v otherwise. Rule Brd is not sufficient to fully describe the behaviour of an output prefix; we need another rule (FBrd) to model the fact that all incoming messages are *discarded* in case only output actions are possible.

Rule Rcv governs the execution of input actions. It states that a message can be received when two *communication constraints* are satisfied: the local attribute environment restricted to interface I ($\Gamma_1 \downarrow I$) satisfies Π, the predicate used by the sender to identify potential receivers; the sender environment Γ' satisfies the receiving predicate $\{\Pi_1[\tilde{v}/\tilde{x}]\}_{\Gamma_1}$. When these two constraints are satisfied the input action is performed and the update U is applied under the substitution $[\tilde{v}/\tilde{x}]$.

Rule FRcv states that an input is *discarded* when the local attribute environment does not satisfy the *sender's predicate*, or the *receiving predicate* is not satisfied by the sender's environment.

Example 3.1 (A flock of drones (Step 4/5))**.** *Let us consider three components from Example 2.3 where* $\Gamma_1(\mathsf{i}) = \Gamma_2(\mathsf{i}) = 1$*,* $\Gamma_2(\mathsf{c}) = 1$*, and* $\Gamma_3(\mathsf{i}) = 0$ *The following transition can be generated by the first component using rule* Brd*:*

$$\Gamma_1 :_{\{\mathsf{id},\mathsf{l}\}} R \mid A \mid T \xrightarrow{\{\mathsf{id}=1,\mathsf{i}=1\} \triangleright \overline{\mathsf{i}\neq 0}(\text{“Req”},1,1)} \Gamma_1 :_{\{\mathsf{id},\mathsf{i}\}} \mathsf{0} \mid A \mid T$$

While the following transition can be generated by the second component using rule Rcv*:*

$$\Gamma_2 :_{\{\mathsf{id},\mathsf{l}\}} R \mid A \mid T \xrightarrow{\{\mathsf{id}=1,\mathsf{i}=1\} \triangleright \mathsf{i}\neq 0(\text{“Req”},1,1)} \Gamma_2 :_{\{\mathsf{id},\mathsf{l}\}} R \mid A \mid T'[\mathsf{Req}/x, 1/y, 1/z] \mid T$$

Instead, the following transition can be generated by third component using FRcv

$$\Gamma_3 :_{\{\mathsf{id},\mathsf{l}\}} R \mid A \mid T \xrightarrow{\widetilde{\{\mathsf{id}=1,\mathsf{i}=1\} \triangleright \mathsf{i}\neq 0(\text{“Req”},1,1)}} \Gamma_3 :_{\{\mathsf{id},\mathsf{l}\}} R \mid A \mid T$$

The third component discards the message because sender's $\Gamma_3 \downarrow I \not\models (\mathsf{i} \neq 0)$*, i.e., because* $\Gamma_3(\mathsf{i}) = 0$*.* □

The behaviour of a component $\Gamma :_I \langle\Pi\rangle P$ is the same as of $\Gamma :_I P$ only when $\Gamma \models \Pi$, while the component is inactive when $\Gamma \not\models \Pi$. This is rendered by rules Aware, FAware1 and FAware2.

Rules SumL, SumR, and FSum describe behaviour of $\Gamma :_I P_1 + P_2$. Rules SumL and SumR are standard and just say that $\Gamma :_I P_1 + P_2$ behaves nondeterministically either like $\Gamma :_I P_1$ or like $\Gamma :_I P_2$. A message is *discarded* by $\Gamma :_I P_1 + P_2$ if and only if both P_1 and P_2 are not able to receive it. We can observe here that the presence of discarding rules is fundamental to prevent processes that cannot receive messages from evolving without performing actions. Thus *dynamic operators*, that are the ones *disappearing* after a transition like awareness and choice, persist after a message refusal.

The behaviour of the interleaving operator is described by rules IntL, IntR and FInt. The first two are standard process algebraic rules for parallel composition while the discarding rule FInt has a similar interpretation as of rule FSum: a message can be discarded only if both the parallel processes can discard it.

Finally, rules Rec, FRec and FZero are the standard rules for handling process definition and the inactive process. The latter states that process $\mathsf{0}$ always discards messages.

$$\frac{\Gamma :_I P \overset{\lambda}{\mapsto} \Gamma' :_I P'}{\Gamma :_I P \xrightarrow{\lambda} \Gamma' :_I P'}\ \textsc{Comp} \qquad \frac{\Gamma :_I P \xmapsto{\widetilde{\Gamma' \triangleright \Pi'(\tilde{v})}} \Gamma :_I P}{\Gamma :_I P \xrightarrow{\Gamma' \triangleright \Pi'(\tilde{v})} \Gamma :_I P}\ \textsc{FComp}$$

$$\frac{C_1 \xrightarrow{\Gamma \triangleright \Pi(\tilde{v})} C_1' \quad C_2 \xrightarrow{\Gamma \triangleright \Pi(\tilde{v})} C_2'}{C_1 \parallel C_2 \xrightarrow{\Gamma \triangleright \Pi(\tilde{v})} C_1' \parallel C_2'}\ \textsc{Sync}$$

$$\frac{C_1 \xrightarrow{\Gamma \triangleright \overline{\Pi}(\tilde{v})} C_1' \quad C_2 \xrightarrow{\Gamma \triangleright \Pi(\tilde{v})} C_2'}{C_1 \parallel C_2 \xrightarrow{\Gamma \triangleright \overline{\Pi}(\tilde{v})} C_1' \parallel C_2'}\ \textsc{ComL} \qquad \frac{C_1 \xrightarrow{\Gamma \triangleright \Pi(\tilde{v})} C_1' \quad C_2 \xrightarrow{\Gamma \triangleright \overline{\Pi}(\tilde{v})} C_2'}{C_1 \parallel C_2 \xrightarrow{\Gamma \triangleright \overline{\Pi}(\tilde{v})} C_1' \parallel C_2'}\ \textsc{ComR}$$

$$\frac{C \xrightarrow{\Gamma \triangleright \overline{\Pi}(\tilde{v})} C' \quad f(\Gamma, \tilde{v}) = \Pi'}{[\ C\ ]^{\triangleright f} \xrightarrow{\Gamma \triangleright \overline{\Pi \wedge \Pi'}(\tilde{v})} [\ C'\ ]^{\triangleright f}}\ \textsc{ResO} \qquad \frac{C \xrightarrow{\Gamma \triangleright \Pi \wedge \Pi'(\tilde{v})} C' \quad f(\Gamma, \tilde{v}) = \Pi'}{[\ C\ ]^{\triangleleft f} \xrightarrow{\Gamma \triangleright \Pi(\tilde{v})} [\ C'\ ]^{\triangleleft f}}\ \textsc{ResI}$$

$$\frac{C \xrightarrow{\Gamma \triangleright \Pi(\tilde{v})} C'}{[\ C\ ]^{\triangleright f} \xrightarrow{\Gamma \triangleright \Pi(\tilde{v})} [\ C'\ ]^{\triangleright f}}\ \textsc{ResOPass} \qquad \frac{C \xrightarrow{\Gamma \triangleright \overline{\Pi}(\tilde{v})} C'}{[\ C\ ]^{\triangleleft f} \xrightarrow{\Gamma \triangleright \overline{\Pi}(\tilde{v})} [\ C'\ ]^{\triangleleft f}}\ \textsc{ResIPass}$$

Table 5: Operational Semantics of Systems

3.2. Operational semantics of systems

The behaviour of an *AbC* system is described by means of the transition relation $\longrightarrow \subseteq \text{Comp} \times \text{SLAB} \times \text{Comp}$, where Comp denotes the set of components and SLAB is the set of transition labels, λ, generated by the following grammar:

$$\lambda \ ::= \Gamma \triangleright \overline{\Pi}(\tilde{v}) \quad | \quad \Gamma \triangleright \Pi(\tilde{v})$$

The definition of the transition relation $\longrightarrow$ is provided in Table 5.

Rules Comp and FComp depends on relation $\longmapsto$ and they are used to lift the effect of local behaviour to the system level. The former rule states that the relations $\longmapsto$ and $\longrightarrow$ coincide when performing either an input or an output actions, while rule FComp states that a component $\Gamma :_I P$ can discard a message and remain unchanged. However, we would like to stress that the system level label of FComp coincides with that of Comp in case of input actions, which means that externally it cannot be observed whether a message has been accepted or discarded.

Rule Sync states that two parallel components C_1 and C_2 can receive the same message. Rule ComL and its symmetric variant ComR govern communication between two parallel components C_1 and C_2.

Rules ResO and ResI show how *restriction operators* $[\ C\ ]^{\triangleright f}$ and $[\ C\ ]^{\triangleleft f}$ limit *output* and *input* capabilities of C under function f.

Rule ResO states that if C evolves to C' with label $\Gamma \triangleright \overline{\Pi}(\tilde{v})$ and $f(\Gamma, \tilde{v}) = \Pi'$ then $[\ C\ ]^{\triangleright f}$ evolves with label $\Gamma \triangleright \overline{\Pi \wedge \Pi'}(\tilde{v})$ to $[\ C'\ ]^{\triangleright f}$. This means that when C sends messages to all the components satisfying Π, the *restriction operator* limits the interaction to only those that also satisfy Π'.

Rule ResI is similar. However, in this case, the *restriction operator* limits the input capabilities of C. Indeed, $[\ C\ ]^{\triangleleft f}$ will receive the message $\tilde{v}$ and evolve to $[\ C'\ ]^{\triangleleft f}$ with a label $\Gamma \triangleright \Pi(\tilde{v})$ only when $C \xrightarrow{\Gamma \triangleright \Pi \wedge \Pi'(\tilde{v})} C'$ where $f(\Gamma, \tilde{v}) = \Pi'$. Thus, message $\tilde{v}$ is delivered only to those components that satisfy both Π and Π'. Note that, both $[\ C\ ]^{\triangleright f}$ and $[\ C\ ]^{\triangleleft f}$ completely hide input/output capabilities whenever $f(\Gamma, \tilde{v}) \wedge \Pi \doteq \mathsf{ff}$.

Rule ResOPass (resp. ResIPass) states any *input transition* (resp. *output transition*) performed by C is also done by $[\ C\ ]^{\triangleright f}$ (resp. $[\ C\ ]^{\triangleleft f}$).

Example 3.2 (A flock of drones (Step 5/5))**.** *The system level evolution of Example 3.1 can be derived using rule* ComL*:*

$$C_1 \| C_2 \| C_3 \xrightarrow{\{\mathsf{id}=1,\mathsf{i}=1\} \triangleright \overline{\mathsf{i} \neq 0}(\text{“Req”},1,1)} \Gamma_1 :_{\{\mathsf{id},\mathsf{i}\}} (0 \mid A \mid T) \| \Gamma_2 :_{\{\mathsf{id},\mathsf{i}\}} (R \mid A \mid T'[\mathsf{Req}/x, 1/y, 1/z] \mid T) \| C_3$$

where $T' \triangleq (\text{“Ack”}, this.\mathsf{i}, -1)@(\mathsf{id} = z).[this.\mathsf{c} := this.\mathsf{c} + y]0$ □

In what follows, we shall use the following notations:

- $C \xrightarrow{\tau} C'$ iff $\exists \tilde{v}, \Gamma$and Π s.t. $C \xrightarrow{\Gamma \triangleright \overline{\Pi}(\tilde{v})} C'$ and $\Pi \doteq \mathsf{ff}$. In this case, we say that the transition is *silent.*
- $\Rightarrow$ denotes $(\xrightarrow{\tau})^*$.
- $\stackrel{\lambda}{\Rightarrow}$ denotes $\Rightarrow \xrightarrow{\lambda} \Rightarrow$ if $(\lambda \neq \tau)$.
- $\stackrel{\widehat{\lambda}}{\Rightarrow}$ denotes $\Rightarrow$ if $(\lambda = \tau)$ and $\stackrel{\lambda}{\Rightarrow}$ otherwise.
- $C \hookrightarrow^{\Pi} C'$ if and only if $\exists \tilde{v}, \Gamma,$ and Π' such that $\Pi \doteq \Pi'$ and $C \xrightarrow{\Gamma \triangleright \overline{\Pi'}(\tilde{v})} C'$.
- $C \hookrightarrow^{\Pi}_{\tau} C'$ if and only if $\exists \tilde{v}, \Gamma,$ and Π' such that $\Pi \doteq \Pi'$ and $C \xRightarrow{\widehat{\Gamma \triangleright \overline{\Pi'}(\tilde{v})}} C'$.

Lemma 3.1. *For any* AbC *component, the following properties hold:*

1. *For any* λ *such that* $\lambda = \Gamma' \triangleright \Pi(\tilde{v})$ *and* $\Pi \doteq \mathsf{ff}$*, then* $C \xrightarrow{\lambda} C$*;*
2. *if* $C_1 \xrightarrow{\tau} C_1'$ *then* $C_1 \| C \xrightarrow{\tau} C_1' \| C$ *and* $C \| C_1 \xrightarrow{\tau} C \| C_1'$*;*
3. *if* $C_1 \Rightarrow C_1'$ *then* $C_1 \| C \Rightarrow C_1' \| C$ *and* $C \| C_1 \Rightarrow C \| C_1'$*;*
4. *if* $C_1 \xRightarrow{\Gamma \triangleright \Pi_1(\tilde{v})} C_1'$ *and* $\Pi_1 \doteq \Pi_2$ *then* $C_1 \xRightarrow{\Gamma \triangleright \Pi_2(\tilde{v})} C_1'$*;*
5. *if* $C_1 \xrightarrow{\tau} C_1'$*, then for any* f*:* $[\ C_1\ ]^{\triangleright f} \xrightarrow{\tau} [\ C_1'\ ]^{\triangleright f}$ *and* $[\ C_1\ ]^{\triangleleft f} \xrightarrow{\tau} [\ C_1'\ ]^{\triangleleft f}$*;*
6. *if* $C_1 \Rightarrow C_1'$*, then for any* f*:* $[\ C_1\ ]^{\triangleright f} \Rightarrow [\ C_1'\ ]^{\triangleright f}$ *and* $[\ C_1\ ]^{\triangleleft f} \Rightarrow [\ C_1'\ ]^{\triangleleft f}$*.*

The full proof is reported in Appendix A.

4. Behavioural Theory for AbC

In this section, we define a behavioural theory for AbC. We start by introducing a reduction barbed congruence, then we present an equivalent definition of a labelled bisimulation and provide a number of equational laws for it. We also show how bisimulation can be used to prove relationships between systems at different level of abstractions.

4.1. Reduction barbed congruence

In the behavioural theory, two terms are considered as equivalent if they cannot be distinguished by any external observer. The choice of observables is important to assess models of concurrent systems and their equivalences. For instance, in the π-calculus both message transmission and reception are considered to be observable. However, this is not the case in AbC because message transmission is non-blocking and thus we cannot externally observe the actual reception of a message. It is important to notice that the transition $C \xrightarrow{\Gamma \triangleright \Pi(\tilde{v})} C'$ does not necessarily mean that C has performed an input action but rather it means that C *might* have performed it.

Indeed, this transition might happen due to one of two different rules in Table 5, namely Comp which guarantees reception and FComp which models non-reception. Hence, input actions cannot be observed by an external observer and only output actions are observable in AbC. This notion is actually borrowed from the corresponding notion in $b\pi$-calculus [11] and it is a natural notion for broadcast calculi in general.

The minimal piece of information we can consider as *observable* from an AbC component is the predicate attached to the sent message. We will use the term “barb” as synonymous with observable, following the works in [15, 16]. From now onwards, we will assume that predicate Π denotes its meaning, not its syntax. In other words, we consider predicates up to semantic equivalence $\doteq$.

Definition 4.1 (Barb)**.** *Let $C{\downarrow_\Pi}$ mean that component C can send a message with some exposed environment Γ and a predicate Π' where $\Pi' \doteq \Pi$ and $\Pi' \neq \mathsf{ff}$ (i.e., $C \xrightarrow{\Gamma \triangleright \overline{\Pi'}(\tilde{v})}$). We write $C \Downarrow_\Pi$ if $C \Rightarrow C' \downarrow_\Pi$ for some C'.*

To define *reduction barbed congruence* we need to define a notion of execution *context* for a component C.

Definition 4.2 (External context)**.** *An external context $\mathcal{C}[\bullet]$ is a component term with a hole, denoted by $[\bullet]$. The external contexts of the* AbC *calculus are generated by the following grammar:*

$$\mathcal{C}[\bullet] ::= \quad \bullet \quad | \quad \mathcal{C}[\bullet] \| C \quad | \quad C \| \mathcal{C}[\bullet] \quad | \quad [\, \mathcal{C}[\bullet] \,]^{\triangleleft f} \quad | \quad [\, \mathcal{C}[\bullet] \,]^{\triangleright f}$$

We define notions of strong and weak barbed congruence to reason about AbC components following the definition of maximal sound theory by Honda and Yoshida [17]. This definition is a slight variant of Milner and Sangiorgi's barbed congruence [16] and it is also known as open barbed bisimilarity [18]. To define *reduction barbed congruence* we have to limit our attention to relations that preserve *observation* and that are preserved in any *context* and after any *reduction.*

Definition 4.3 (Closures)**.** *Let $\mathcal{R}$ be a binary relation over* AbC*-components:*

Barb Preservation *$\mathcal{R}$ is barb-preserving iff for every $(C_1, C_2) \in \mathcal{R}$, $C_1{\downarrow_\Pi}$ implies $C_2 \Downarrow_\Pi$*

Reduction Closure *$\mathcal{R}$ is reduction-closed iff for every $(C_1, C_2) \in \mathcal{R}$ and predicate Π, $C_1 \overset{\Pi}{\hookrightarrow} C_1'$ implies $C_2 \overset{\Pi}{\hookrightarrow}_\tau C_2'$ for some C_2' such that $(C_1', C_2') \in \mathcal{R}$*

Context Closure *$\mathcal{R}$ is context-closed iff for every $(C_1, C_2) \in \mathcal{R}$ and for all contexts $\mathcal{C}[\bullet]$, $(\mathcal{C}[C_1], \mathcal{C}[C_2]) \in \mathcal{R}$*

Now, everything is in place to define reduction barbed congruence.

Definition 4.4 (Weak Reduction Barbed Congruence)**.** *A weak reduction barbed congruence is a symmetric relation $\mathcal{R}$ over the set of* AbC*-components which is barb-preserving, reduction closed, and context-closed.*

Two components are weak barbed congruent, written $C_1 \cong C_2$, if $(C_1, C_2) \in \mathcal{R}$ for some weak reduction barbed congruence relation $\mathcal{R}$. The strong reduction congruence "$\simeq$" is obtained in a similar way by replacing $\Downarrow$ with $\downarrow$ and $\hookrightarrow_\tau$ with $\hookrightarrow$.

Remark 4.1. *Note that, while in [17, 16, 18]* reduction closure *only takes into account* invisible actions τ*, here we also consider output actions but in this case reductions must also preserve the output predicates. In fact we have a* family *of reductions, one for each kind of interaction driven by a predicate* Π*. This because any interaction over a predicate* Π *is somehow* hidden *to any component that does not satisfy* Π*. The definition is in fact similar to the one of the bπ-calculus [11] except that in bπ it is not required that reductions preserve the output observations. To clarify the importance of this choice consider the following example:*

$$\begin{aligned} C_1 &\triangleq \{(c,0)\} :_{\{\}} P \\ C_2 &\triangleq \{(c,0)\} :_{\{\}} Q \end{aligned}$$

where processes P and Q are defined as follow:

$$\begin{aligned} P \triangleq\ & (\mathsf{tt})@(\mathsf{a} \geq 0).(\mathsf{tt})@(\mathsf{b} = \mathsf{tt}).0 \\ & +(\mathsf{tt})@(\mathsf{a} \geq 0 \wedge \mathsf{a} \leq \mathsf{this.c}).0 \\ & +()@\mathsf{ff}.[c := c+1]P \end{aligned}$$

$$\begin{aligned} Q \triangleq\ & (\mathsf{tt})@(\mathsf{a} \geq 0).0 \\ & +(\mathsf{tt})@(\mathsf{a} \geq 0 \wedge \mathsf{a} \leq \mathsf{this.c}).(\mathsf{tt})@(\mathsf{b} = \mathsf{tt}).0 \\ & +()@\mathsf{ff}.[c := c+1]Q \end{aligned}$$

Process P can either send a message to all the components satisfying $\mathsf{a} \geq 0$ and then execute $(\mathsf{tt})@(\mathsf{b} = \mathsf{tt})$ for any integer i (a value of c), send a message to all the components satisfying $\mathsf{a} \geq 0 \wedge \mathsf{a} \leq i$ and then terminates or increment c silently and continue as P. Process Q is similar, however it can either send a message to all the components satisfying $(\mathsf{a} \geq 0)$ and then terminates, send a message to all the components satisfying $\mathsf{a} \geq 0 \wedge \mathsf{a} \leq i$ and then execute $(\mathsf{tt})@(\mathsf{b} = \mathsf{tt})$, or update c silently and continuous as Q.

Clearly, components C_1 and C_2 are behaviourally different and one would not wish to equate them. Actually, they are distinguished by our definition of reduction barbed congruence only because of the way we define reduction closure. The reader should be convinced that if we relax the definition of reduction closure (by omitting predicates and allowing the reduction to range over silent and output actions) these components would be deemed weak reduction-barbed congruent. Indeed, there does not exist any finite *context that is able to distinguish C_1 and C_2.*

Note that, if we only limit our attention to systems with finite interaction capabilities*, namely with all C such that $\{\Pi \mid C \xrightarrow{\Gamma \triangleright \overline{\Pi}(\tilde{v})} C'\}$ is finite, reduction closures like those in [11, 17, 16, 18] can be considered.*

Lemma 4.1. *If $C_1 \cong C_2$ then*

- *$C_1 \Rightarrow C_1'$ implies $C_2 \Rightarrow C_{\cong C_1'}$ where $C_{\cong C_1'}$ denotes a component that is weakly barbed congruent to C_1'*
- *$C_1 \Downarrow_\Pi$ iff $C_2 \Downarrow_\Pi$.*

Proof. (We prove each statement separately)

- The proof of first item proceeds by induction on w by showing that if $C_1 \Rightarrow^w C_1'$ then $C_2 \Rightarrow C_{\cong C_1'}$, where w is the number of τ steps needed to move from C_1 to C_1'.
  - Base case, $w = 0$: For all C_1 we have that $C_1 \Rightarrow^0 C_1$. We also have that $C_2 \Rightarrow C_2$. The statement follows directly from the fact that $C_1 \cong C_2$.
  - Inductive Hypothesis: We assume that for all C_1 and $\forall k \leq w$ if $C_1 \Rightarrow^k C_1'$ then $C_2 \Rightarrow C_{\cong C_1'}$.
  - Inductive Step: Let $C_1 \Rightarrow^{w+1} C_1'$. By definition of $\Rightarrow^{w+1}$ we have that there exists C_1'' such that $C_1 \xrightarrow{\tau} C_1''$ and $C_1'' \Rightarrow^w C_1'$. Since $C_1 \cong C_2$, and $\cong$ is *reduction closed* (see Definition 4.3 and Definition 4.4) we have that there exists C_2'' such that $C_2 \Rightarrow C_2''$ and $C_1'' \cong C_2''$. Moreover, by inductive hypothesis we have that $C_2'' \Rightarrow C_{\cong C_1'}$. Hence, $C_2 \Rightarrow C_{\cong C_1'}$ as required.
- The proof of second item follows by observing that $C_1 \Downarrow_\Pi$ if and only if $C_1 \Rightarrow C_1' \downarrow_\Pi$. Directly from the previous point we have that there exists C_2' such that $C_2 \Rightarrow C_2'$ and $C_1' \cong C_2'$. Hence, since $\cong$ is *barb preserving* (see Definition 4.3 and Definition 4.4), we have that $C_2' \Downarrow_\Pi$ and, in turn, $C_2 \Downarrow_\Pi$.

□

4.2. Bisimulation Proof Methods

In this section, we first define a notion of labelled bisimilarity of AbC components, then we prove that it coincides with the reduction barbed congruence, introduced in the previous section. This "alternative" characterisation is useful to prove actual properties of AbC systems. In fact, barbed congruence could hardly serve the scope, since it requires testing components in every possible context.

First we need to introduce a notion of *semantic equivalence* among transition labels. The reason is that in standard process algebras, labelled bisimilarities can be defined in terms of a syntactic equivalence among labels while in AbC different labels may have the same meaning and impact on the context. One can consider the following two output labels $\Gamma \triangleright \overline{(a \neq 10)}(\tilde{v})$ and $\Gamma \triangleright \overline{\neg(a = 10)}(\tilde{v})$. Even if these two labels are *different*, their impact on the *context* is the same. Since this equivalence is based on the *semantic equivalence* among predicates of Definition 2.1, we use the same symbol $\doteq$ to denote this relation.

Definition 4.5. *We let $\doteq \subseteq SLAB \times SLAB$ be the smallest relation such that for any Γ, Γ', $\tilde{v}$, $\tilde{w}$:*

- $\Gamma \triangleright \overline{\Pi_1}(\tilde{v}) \backsimeq \Gamma \triangleright \overline{\Pi_2}(\tilde{v})$, *for any* $\Pi_1 \backsimeq \Pi_2$;
- $\Gamma \triangleright \overline{\Pi_1}(\tilde{v}) \backsimeq \Gamma' \triangleright \overline{\Pi_2}(\tilde{w})$, *for any* $\Pi_1 \backsimeq \Pi_2 \backsimeq \mathsf{ff}$;
- $\Gamma \triangleright \Pi_1(\tilde{v}) \backsimeq \Gamma \triangleright \Pi_2(\tilde{v})$, *for any* $\Pi_1 \backsimeq \Pi_2$.

Definition 4.6 (Weak Bisimulation)**.** *A symmetric binary relation $\mathcal{R}$ over the set of* AbC*-components is a* weak bisimulation *if and only if for any $(C_1, C_2) \in \mathcal{R}$ and for any λ_1*

$$C_1 \xrightarrow{\lambda_1} C_1' \text{ implies } \exists \lambda_2 : \lambda_1 \backsimeq \lambda_2 \text{ such that } C_2 \stackrel{\widehat{\lambda_2}}{\Longrightarrow} C_2' \text{ and } (C_1', C_2') \in \mathcal{R}$$

Two components C_1 and C_2 are weakly bisimilar, written $C_1 \approx C_2$ if there exists a weak bisimulation $\mathcal{R}$ relating them.

It is worth noting that *strong bisimulation* and *strong bisimilarity* ($\sim$) can be defined similarly, only $\stackrel{\widehat{\lambda_2}}{\Longrightarrow}$ is replaced by $\xrightarrow{\lambda_2}$. It is easy to prove that $\sim$ and $\approx$ are equivalence relations by relying on the classical arguments of [19]. However, our bisimilarities enjoy a much more interesting property: closure under any external context.

The following lemmas state that our weak labelled bisimilarities is preserved by parallel composition, and restriction. Similar lemmas do hold also for the strong variant. The proofs of these lemmas are reported in Appendix A.

Lemma 4.2 ($\approx$ is preserved by parallel composition)**.** *If C_1 and C_2 are two components, we have that $C_1 \approx C_2$ implies $C_1 \| C \;\approx\; C_2 \| C$ for all components C.*

Lemma 4.3 ($\approx$ is preserved by restriction)**.** *If C_1 and C_2 are two components, we have that $C_1 \approx C_2$ implies $[\, C_1 \,]^{\triangleleft f} \;\approx\; [\, C_2 \,]^{\triangleleft f}$ and $[\, C_1 \,]^{\triangleright f} \;\approx\; [\, C_2 \,]^{\triangleright f}$ for any f.*

As an immediate consequence of Lemma 4.2 and Lemma 4.3, we have that $\approx$ is a congruence relation (i.e., closed under any external AbC context). Notably, similar lemmas do hold also for $\sim$.

We are now ready to show how weak bisimilarity can be used as a proof technique for reduction barbed congruence.

Theorem 4.1 (Soundness)**.** *$C_1 \approx C_2$ implies $C_1 \;\cong\; C_2$, for any two components C_1 and C_2.*

Proof. It is sufficient to prove that bisimilarity is barb-preserving, reduction-closed, and context-closed.

- (Barb-preservation): By the definition of the barb $C_1{\downarrow_\Pi}$ if $C_1 \xrightarrow{\Gamma \triangleright \overline{\Pi}(\tilde{v})}$ for an output label $\Gamma \triangleright \overline{\Pi}(\tilde{v})$ with $\Pi \neq \mathsf{ff}$. As $(C_1 \approx C_2)$, we have that also $C_2 \stackrel{\Gamma \triangleright \overline{\Pi}(\tilde{v})}{\Longrightarrow}$ and $C_2 \Downarrow_\Pi$.
- (Reduction-closure): Let $C_1 \stackrel{\Pi}{\hookrightarrow} C_1'$. This means that there exist Γ, $\tilde{v}$ and Π' such that $C_1 \xrightarrow{\Gamma \triangleright \overline{\Pi'}(\tilde{v})} C_1'$ and $\Pi \backsimeq \Pi'$. As $(C_1 \approx C_2)$, then there exists C_2' such that $C_2 \stackrel{\widehat{\Gamma \triangleright \overline{\Pi''}(\tilde{v})}}{\Longrightarrow} C_2'$ with $\Pi' \backsimeq \Pi''$ and $(C_1' \approx C_2')$. Hence, $C_2 \stackrel{\Pi}{\hookrightarrow} C_2'$ and $(C_1' \approx C_2')$.
- (Context-closure): Let $(C_1 \approx C_2)$ and let $\mathcal{C}[\bullet]$ be an arbitrary AbC-context. By induction on the structure of $\mathcal{C}[\bullet]$ and using Lemma 4.2 and Lemma 4.3, we have that $\mathcal{C}[C_1] \approx \mathcal{C}[C_2]$.

In conclusion, we have that $C_1 \cong C_2$ as required. □

This soundness theorem allow us to use bisimilarity when we have to prove that two AbC components are *barbed equivalent*. We want now to study *completeness* in order to show that bisimilarity is more than a proof technique, it rather represents an alternative characterisation of reduction barbed congruence.

Theorem 4.2 (Completeness)**.** *$C_1 \cong C_2$ implies $C_1 \;\approx\; C_2$, for any two components C_1 and C_2.*

Proof. It is sufficient to prove that the relation $\mathcal{R} = \{(C_1, C_2) \mid \text{ such that } (C_1 \cong C_2)\}$ is a bisimulation. For this reason we have to show that:

- $\mathcal{R}$ is symmetric;
- for each $(C_1, C_2) \in \mathcal{R}$ and for any λ_1

$$C_1 \xrightarrow{\lambda_1} C_1' \text{ implies } \exists \lambda_2 : \lambda_1 \backsimeq \lambda_2 \text{ such that } C_2 \stackrel{\widehat{\lambda_2}}{\Longrightarrow} C_2' \text{ and } (C_1', C_2') \in \mathcal{R}$$

The first item derives directly from the fact that $\cong$ is symmetric. To prove the second item we have to consider three cases:

Case 1: $\lambda_1 = \Gamma \triangleright \overline{\Pi}(\tilde{v})$ *and* $\Pi \backsimeq \mathsf{ff}$. From Definition 4.4, we have that $C_1 \cong C_2$ is *reduction preserving*. This means that if $C_1 \stackrel{\mathsf{ff}}{\hookrightarrow} C_1'$ then there exists C_2' such that $C_2 \stackrel{\mathsf{ff}}{\hookrightarrow}_\tau C_2'$, that is there exists $\lambda_2 = \Gamma' \triangleright \overline{\Pi''}(\tilde{v}') \backsimeq \lambda_1$ such that $C_2 \stackrel{\widehat{\lambda_2}}{\Longrightarrow} C_2'$ and $C_1' \cong C_2'$.

Case 2: $\lambda_1 = \Gamma \triangleright \overline{\Pi}(\tilde{v})$ *and* $\Pi \neq \mathsf{ff}$. We have to prove that if $C_1 \xrightarrow{\lambda_1} C_1'$ there exists $\lambda_2 \backsimeq \lambda_1$ such that $C_2 \stackrel{\lambda_2}{\Longrightarrow} C_2'$ where $\lambda_2 = \Gamma' \triangleright \overline{\Pi'}(\tilde{v}')$, $\Pi \backsimeq \Pi'$, and $C_1' \cong C_2'$.

Let us consider the following context $\mathcal{C}[\bullet]$:

$$\mathcal{C}[\bullet] = \bullet \parallel \Gamma_\Pi : (\Pi_\Gamma \wedge (\tilde{x} = \tilde{v}))(\tilde{x}).((\mathsf{tt})@(\mathsf{b} = \mathsf{tt}).0 + (\mathsf{tt})@\mathsf{ff}.0)$$

where $\Gamma_\Pi \in \mathcal{M}(\Pi)$, Π_Γ is a predicate uniquely identifying Γ, i.e. $\mathcal{M}(\Pi_\Gamma) = \{\Gamma\}$, while b is an attribute occurring neither in C_1 nor in C_2. Let A be the (finite) set of attribute identifiers occurring in C_1 and C_2, predicate Π_Γ can be defined as tt if A is empty and otherwise as follows:

$$\Pi_\Gamma = \bigwedge_{\mathsf{a} \in A} (\mathsf{a} = \Gamma(\mathsf{a}))$$

Note that because A is finite and contains all attribute identifiers in both C_1 and C_2, we ensure that Π_Γ uniquely identifies Γ.

Since $\cong$ is *context closed*, we have that $\mathcal{C}[C_1] \cong \mathcal{C}[C_2]$. Moreover, by applying rules of Table 5, we have that:

$$\mathcal{C}[C_1] \xrightarrow{\lambda_1} C_1' \parallel \Gamma_\Pi : (\mathsf{tt})@(\mathsf{b} = \mathsf{tt}).0 + (\mathsf{tt})@\mathsf{ff}.0$$

This implies that

$$\mathcal{C}[C_1] \stackrel{\Pi}{\hookrightarrow} C_1' \parallel \Gamma_\Pi : (\mathsf{tt})@(\mathsf{b} = \mathsf{tt}).0 + (\mathsf{tt})@\mathsf{ff}.0$$

Since $\cong$ is reduction closed we have that

$$\mathcal{C}[C_2] \stackrel{\Pi}{\hookrightarrow}_\tau C' \qquad \text{and} \qquad C_1' \parallel \Gamma_\Pi : (\mathsf{tt})@(\mathsf{b} = \mathsf{tt}).0 + (\mathsf{tt})@\mathsf{ff}.0 \cong C'$$

Since b does not occur in C_2, and $C' \Downarrow_{(\mathsf{b}=\mathsf{tt})}$, we can infer that there exists $\lambda_2 = \Gamma' \triangleright \overline{\Pi'}(\tilde{v}')$ such that $\Pi \backsimeq \Pi'$:

$$\mathcal{C}[C_2] \stackrel{\lambda_2}{\Longrightarrow} C_2' \parallel \Gamma_\Pi : (\mathsf{tt})@(\mathsf{b} = \mathsf{tt}).0 + (\mathsf{tt})@\mathsf{ff}.0$$

where $\Gamma' \models \Pi_\Gamma$ and $\tilde{v}' = \tilde{v}$. This implies that $\Gamma' \in \mathcal{M}(\Pi_\Gamma) = \{\Gamma\}$ and $\lambda_1 \backsimeq \lambda_2$.

We now observe that:

$$C_1' \parallel \Gamma_\Pi : (\mathsf{tt})@(\mathsf{b} = \mathsf{tt}).0 + (\mathsf{tt})@\mathsf{ff}.0 \stackrel{\mathsf{ff}}{\hookrightarrow} C_1' \parallel \Gamma_\Pi : 0$$

We can use again the fact that $\cong$ is *reduction closed* to conclude that:

$$C_2' \parallel \Gamma_\Pi : (\mathsf{tt})@(\mathsf{b} = \mathsf{tt}).0 + (\mathsf{tt})@\mathsf{ff}.0 \xrightarrow{\mathsf{ff}}_\tau C_2'' \parallel \Gamma_\Pi : 0$$

where $C_2' \Rightarrow C_2''$ and $C_1' \parallel \Gamma_\Pi : 0 \cong C_2'' \parallel \Gamma_\Pi : 0$. We can observe that $C_1' \parallel \Gamma_\Pi : 0 \cong C_2'' \parallel \Gamma_\Pi : 0$ if and only if $C_1' \cong C_2''$.

Finally, we have that $C_2 \stackrel{\lambda_2}{\Longrightarrow} C_2''$ and $C_1' \cong C_2''$.

Case 3: $\lambda_1 = \Gamma \triangleright \Pi(\tilde{v})$. In this case the proof proceeds like for the previous one by considering the following context $\mathcal{C}[\bullet]$:

$$\mathcal{C}[\bullet] = \bullet \parallel \Gamma : (\Pi)@\tilde{v}.((\mathsf{tt})@(\mathsf{b} = \mathsf{tt}).0 + (\mathsf{tt})@\mathsf{ff}.0)$$

□

Theorem 4.3 (Characterisation)**.** *Bisimilarity and reduction barbed congruence coincide.*

Proof. As a direct consequence of Theorem 4.1 and Theorem 4.2, we have that weak bisimilarity and weak reduction barbed congruence coincide. □

The proof for the strong variant of equivalence (i.e., $C_1 \simeq C_2$ coincides with $C_1 \sim C_2$) follows in a similar way and it is omitted for the sake of brevity.

5. Bisimulations at work

In the previous section we proved that bisimilarity is a congruence relation for all external *AbC* contexts, i.e., system level contexts as described in Definition 4.2. In this section we show that, due to the dependencies of processes on the attribute environment, almost all process-level operators do not preserve bisimilarity, the only exception being the awareness operator. However, this problem can be solved by closing bisimilarity under any possible substitution as we will see later. Notice that our bisimilarity is still a congruence because it is defined at the level of system components and thus only external contexts matter. The rest of the section concentrates on other properties and equational laws exhibited by bisimilarity. Unless stated otherwise, the properties hold for both strong and weak bisimilarity.

5.1. Equational Laws for AbC *Bisimulation*

As mentioned above, weak bisimilarity is not preserved by most process level operators.

Remark 5.1. *For some attribute environment* Γ*, an interface* I*, and two processes* P *and* Q *where* $\Gamma :_I P \approx \Gamma :_I Q$*, we have that*

1. $\Gamma :_I P\sigma \not\approx \Gamma :_I Q\sigma$ *for some substitution* σ
2. $\Gamma :_I \alpha.P \not\approx \Gamma :_I \alpha.Q$ *for some action* α
3. $\Gamma :_I P|R \not\approx \Gamma :_I Q|R$ *for some process* R
4. $\Gamma :_I \langle\Pi\rangle P \approx \Gamma :_I \langle\Pi\rangle Q$ *for every predicate* Π
5. $\Gamma :_I \alpha.[a := E]P \not\approx \Gamma :_I \alpha.[a := E]Q$ *for some update* $[a := E]$

Proof. Let $C_1 = \Gamma :_I \overbrace{\langle this.a = w\rangle(v')@\Pi.0}^{P}$ where $\Gamma(a) = v$, $C_2 = \Gamma :_I \overbrace{0}^{Q}$, and $R = ()@\mathsf{ff}.[a := w]0$. It is easy to see that $C_1 \approx C_2$, because both components are not able to progress. Notice that $\Gamma \not\models (this.a = w)$.

1. If we apply the substitution $[v/w]$ to both processes P and Q, we have that $\Gamma :_I P[v/w] \xrightarrow{\Gamma\downarrow I \triangleright \overline{\Pi}(v')}$ and $\Gamma :_I Q[v/w] \not\xrightarrow{\Gamma\downarrow I \triangleright \overline{\Pi}(v')}$ and $\Gamma :_I P\sigma \not\approx \Gamma :_I Q\sigma$ as required.
2. The statement, $\Gamma :_I \alpha.P \not\approx \Gamma :_I \alpha.Q$ for some action α, is a direct consequence of the first statement. For instance, consider an input prefix of the following form $(\mathsf{tt})(w)$.

3. The statement, $\Gamma :_I P|R \not\approx \Gamma :_I Q|R$ for some process R, can be easily observed from our example when we put the process R in parallel of the processes P and Q.
4. The statement, $\Gamma :_I \langle\Pi\rangle P \approx \Gamma :_I \langle\Pi\rangle Q$ for every predicate Π, is a direct consequence of operational rules for the awareness operator.
5. The last statement can be observed from the following update $[a := w]$.

□

It should be noted that if we close bisimilarity under substitutions by definition, all of the statements in Remark 5.1 can be proved to preserve bisimilarity. The definition would be a slight variant of the notion of full bisimilarity proposed by Sangiorgi and Walker in [18]. In this way, the components C_1 and C_2 in the proof above are no longer bisimilar since they are not equivalent after substitution $[v/w]$. However, the new notion of bisimilarity induced by the closure is finer than the one proposed in this article.

The following remark shows that, as expected, non-deterministic choice does not preserve weak bisimilarity. Below we explain the issue with concrete examples.

Remark 5.2. *For some attribute environment Γ, an interface I, and two processes P and Q where $\Gamma :_I P \approx \Gamma :_I Q$, we have that $\Gamma :_I P + R \not\approx \Gamma :_I Q + R$ for some process R*

Proof. Let $C_1 = \Gamma :_I \Pi_1(x).\mathsf{0}$, $C_2 = \Gamma :_I \Pi_2(x).\mathsf{0}$, and $R = (v)@\Pi.\mathsf{0}$. Though the receiving predicates for both components are different we still have that $C_1 \approx C_2$ and this is because that input actions are not perceived. When a message $\Gamma' \triangleright \overline{\Pi_3}(w)$ arrives, where $\Gamma \downarrow I \models \Pi_3$, $\Gamma' \models [\![\Pi_1[w/x]]\!]_\Gamma$ and $\Gamma' \not\models [\![\Pi_2[w/x]]\!]_\Gamma$, component C_1 applies rule Comp and evolves to $\Gamma :_I \mathsf{0}$ while component C_2 applies rule FComp and stays unchanged. Both transitions carry the same label and again $\Gamma :_I \mathsf{0}$ and $\Gamma :_I \Pi_2(x).\mathsf{0}$ are equivalent for a similar reason. An external observer cannot distinguish them.

If we allow mixed choice within a single component, then one can distinguish between $\Pi_1(x)$ and $\Pi_2(x)$.

$$\Gamma :_I \Pi_1(x).\mathsf{0} + R \not\approx \Gamma :_I \Pi_2(x).\mathsf{0} + R$$

Assume that the message $\Gamma' \triangleright \overline{\Pi_3}(w)$ is arrived, we have that:

$$\Gamma :_I \Pi_1(x).\mathsf{0} + R \xrightarrow{\Gamma' \triangleright \Pi_3(w)} \Gamma :_I \mathsf{0} \xrightarrow{\Gamma \downarrow I \triangleright \overline{\Pi}(v)}$$

while

$$\Gamma :_I \Pi_2(x).\mathsf{0} + R \xrightarrow{\Gamma' \triangleright \Pi_3(w)} \Gamma :_I \Pi_2(x).\mathsf{0} + R \xrightarrow{\Gamma \downarrow I \triangleright \overline{\Pi}(v)} \Gamma :_I \mathsf{0}$$

However, this is obvious since our relation is defined at the system-level. So it abstracts from internal behaviour and characterises the behaviour of AbC systems from an external observer point of view. Another reason why + does not preserve weak bisimilarity is due to the equivalence of processes $\mathsf{0}$ and $()@\mathsf{ff}.\mathsf{0}$. The latter reason cannot be avoided while the former can be mitigated by prohibiting mixed choice, i.e., nondeterministic choice between input and output actions. In practice this would not be a problem since mixed choice is very hard to implement. □

The following lemmas prove useful properties about AbC operators (i.e., parallel composition is commutative, associative, ...). We omit their proofs; they follow directly from the operational semantics of AbC that we presented in Section 3.

Lemma 5.1 (Parallel composition)**.**

- $C_1 \| C_2 \approx C_2 \| C_1$
- $(C_1 \| C_2) \| C_3 \approx C_1 \| (C_2 \| C_3)$
- $\Gamma :_I \mathsf{0} \parallel C \approx C$

Lemma 5.2 (Non-deterministic choice)**.**

- $\Gamma :_I P_1 + P_2 \approx \Gamma :_I P_2 + P_1$
- $\Gamma :_I (P_1 + P_2) + P_3 \approx \Gamma :_I P_1 + (P_2 + P_3)$
- $\Gamma :_I P + \mathsf{0} \approx \Gamma :_I P$
- $\Gamma :_I P + P \approx \Gamma :_I P$
- $\Gamma :_I \langle \Pi \rangle (P + Q) \approx \Gamma :_I \langle \Pi \rangle P + \langle \Pi \rangle Q$

Lemma 5.3 (Interleaving)**.**

- $\Gamma :_I P_1 | P_2 \approx \Gamma :_I P_2 | P_1$
- $\Gamma :_I (P_1 | P_2) | P_3 \approx \Gamma :_I P_1 | (P_2 | P_3)$
- $\Gamma :_I P | \mathsf{0} \approx \Gamma :_I P$

Lemma 5.4 (Awareness)**.**

- $\Gamma :_I \langle \mathsf{ff} \rangle P \approx \Gamma :_I \mathsf{0}$
- $\Gamma :_I \langle \mathsf{tt} \rangle P \approx \Gamma :_I P$
- $\Gamma :_I \langle \Pi_1 \rangle \langle \Pi_2 \rangle P \approx \Gamma :_I \langle \Pi_1 \wedge \Pi_2 \rangle P$

Lemma 5.5 (Silent components cannot be observed)**.** *Let $Act(P)$ denote the set of actions in process P. If $Act(P)$ does not contain any output action, then:*

$$\Gamma :_I P \approx \Gamma :_I \mathsf{0}$$

Proof. The proof follows from the fact that components with no external side-effects (i.e., do not exhibit barbs) cannot be observed. When $Act(P)$ does not contain output actions, component $\Gamma :_I P$ can either make silent moves, which component $\Gamma :_I \mathsf{0}$ can mimic by simply doing nothing, or input a message, which component $\Gamma :_I \mathsf{0}$ can mimic by discarding the message. □

5.2. Proving equivalence of AbC *systems*

Now we proceed with a few examples to provide evidence of interesting features of the *AbC* calculus.

Example 5.1. *We have that $C_1 \approx C_2$ when $C_1 = \Gamma :_I \Pi(x).P$, $C_2 = \Gamma :_I \Pi_1(x).P + \Pi_2(x).P$ and $\Pi \doteq \Pi_1 \vee \Pi_2$.*

Clearly, components C_1 and C_2 are bisimilar because any message, accepted by C_2, can also be accepted by C_1 and vice versa. After a successful input both components proceed with the same continuation process $P[v/x]$. For instance, consider the message $\Gamma' \triangleright \overline{\Pi_1}(v)$ in which Γ' is only satisfied by predicate Π_2, it is still satisfied by predicate Π. The overlapping between the input and the non-deterministic choice constructs is clear in this scenario. For this special case we can replace the non-deterministic choice with an "or" predicate while preserving the observable behaviour. The intuition is illustrated in Figure 1.

It is worth noting that as a corollary of the above equivalence we have:

$$\Gamma :_I \Pi_1(\tilde{x}).P + \cdots + \Pi_n(\tilde{x}).P \approx \Gamma :_I (\Pi_1 \vee \Pi_2 \vee \cdots \vee \Pi_n)(\tilde{x}).P$$

Example 5.2. $\Gamma_1 :_I (E_1)@\Pi.P \approx \Gamma_2 :_I (E_2)@\Pi.P$ *if and only if* $\Gamma_1 \downarrow I = \Gamma_2 \downarrow I$, $[\![E_1]\!]_{\Gamma_1} = [\![E_2]\!]_{\Gamma_2}$, *and* $\{\Pi\}_{\Gamma_1} \doteq \{\Pi\}_{\Gamma_2}$.

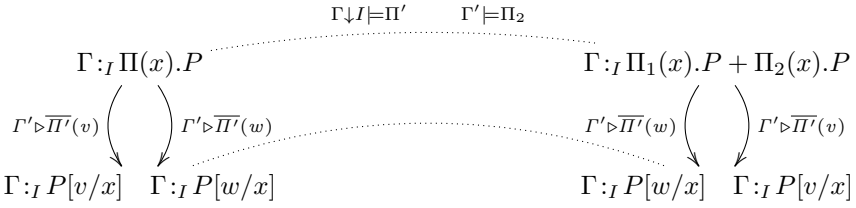

Figure 1: The relationship between the "or" predicate and the non-deterministic choice

It is clear that even if $\Gamma_1 \neq \Gamma_2$, these components are still bisimilar since their interfaces and exposed messages are equivalent. This is an important property which ensures that components need not to have isomorphic attribute environments to be equivalent. The intuition is that components can control what attribute values to be exposed to the interacting partners. In some sense the component has the power of selecting the criteria in which its communicated messages can be filtered.

Now we show some interesting properties about the restriction mechanism in AbC. The restriction mechanism in AbC where a predicate can be partially exposed is very useful in describing collective behaviour from a global point of view. This means that local interactions are hidden from an external observer which can only observe system level behaviour (collective-behaviour). The following lemma shows properties of our restriction operators. These properties follow directly from the operational semantics, presented in Section 3.

Lemma 5.6 (Restriction)**.**

- $[\ [\ C\ ]^{\triangleright f_1}\ ]^{\triangleright f_2} \approx [\ C\ ]^{\triangleright f_1 \wedge f_2}$.
- $[\ [\ C\ ]^{\triangleleft f_1}\ ]^{\triangleleft f_2} \approx [\ C\ ]^{\triangleleft f_1 \wedge f_2}$.
- $[\ [\ C\ ]^{\triangleleft f_1}\ ]^{\triangleright f_2} \approx [\ [\ C\ ]^{\triangleright f_2}\ ]^{\triangleleft f_1}$.
- $[\ 0\ ]^{\triangleright f_1} \approx 0$
- $[\ 0\ ]^{\triangleleft f_1} \approx 0$

Note that scope extrusion does not work for our output restriction operator. To see this, consider the constant function $f(\Gamma, \tilde{v}) = \mathsf{ff}$ for any Γ and $\tilde{v}$. This function has a preemptive power to hide any message regardless of Γ and $\tilde{v}$ and thus prevents any possible interaction outside the scope.

In the next example we show the expressive power of name restriction in a more elaborated scenario.

Example 5.3. *We consider two types of forwarding components, a source component $CP = \Gamma_p :_I P$ and an intermediate component $CF = \Gamma_i :_I F$ where the behaviour of processes P and F is defined below.*

$$P \triangleq\ (this.\mathsf{id},\ \tilde{v})@(\Pi_1 \vee (\mathsf{role} = \mathsf{fwd})).0$$

$$F \triangleq (x \in this.\mathsf{nbr})(x,\ \tilde{y}).(x,\ \tilde{y})@(\mathsf{role} = \mathsf{fwd}).(x,\ \tilde{y})@\Pi_1.0$$

Process P sends an advertisement message to all components that either satisfy predicate Π_1 where $\Pi_1 = (\mathsf{role} = \mathsf{client})$ or have a forwarder role (i.e., $(\mathsf{role} = \mathsf{fwd})$). Process F may receive an advertisement from a neighbour, after that it first sends it to nearby forwarders and then sends the advertisement to nearby clients. The scenario is simplified to allow at most two hops from the source component.

The goal of the source component is to ensure that its advertisement message reaches all clients across the network. To prove if the above specification guarantees this property[2], we first need to fix the topology of the network as reported in Figure 2. For the sake of simplicity we will only consider a network of one

[2]The results in this scenario only hold for weak bisimulation.

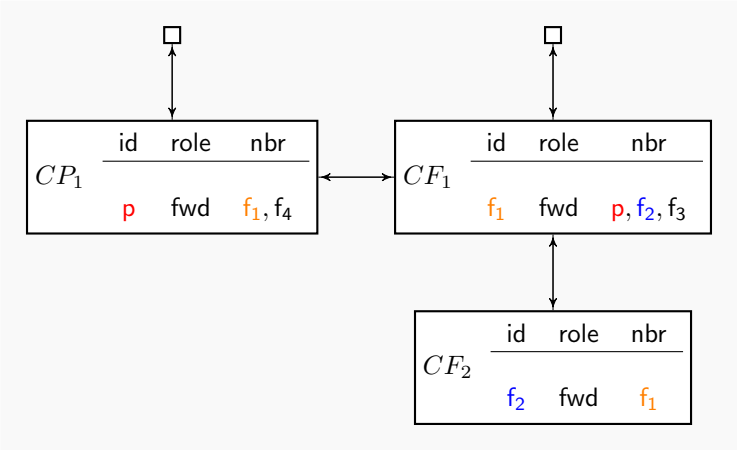

Figure 2: The system with assumptions about the network topology

source $CP_1 = \Gamma_p :_{\{role\}} P$ and two forwarders $CF_1 = \Gamma_1 :_{\{role\}} F$ and $CF_2 = \Gamma_2 :_{\{role\}} F$. Notice that the interface of these components contains the role attribute. We assume short-range communication where CP_1 messages can reach to CF_1 and CF_2 can only receive the messages when CF_1 forwards them. Assume that initially the attribute environments Γ_p, Γ_1 and Γ_2 are defined as follows:

$$\Gamma_p = \{(\mathsf{role}, \mathsf{fwd}),\ (\mathsf{id}, p),\ (\mathsf{nbr}, \{f_1, f_4\})\}, \quad \Gamma_1 = \{(\mathsf{id}, f_1),\ (\mathsf{role}, \mathsf{fwd}),\ (\mathsf{nbr}, \{p, f_2, f_3\})\}$$
$$\Gamma_2 = \{(\mathsf{id}, f_2),\ (\mathsf{role}, \mathsf{fwd}),\ (\mathsf{nbr}, \{f_1\})\}$$

To avoid interference with other components running in parallel we can use *restriction operators* to guarantee that interactions between the source and the forwarders are *private*. The full system is represented by the component N as defined below:

$$N = [\ CP_1 \parallel CF_1 \parallel CF_2\ ]^{\triangleright f_{\mathsf{fwd}}}$$

where $f_{\mathsf{fwd}}(\Gamma, \tilde{v})$ yields $(\mathsf{role} \neq \mathsf{fwd})$ and guarantees that only non-forwarders can receive a message outside the boundaries of the restriction.

The behaviour of N without any interventions from other source components is reported on the right side of Figure 3. The source component CP_1 initiates the interaction by sending an advertisement to nearby clients and forwarders and evolves to $\Gamma_p :_{\{role\}} 0$. Forwarder CF_1 receives the message and evolves to CF_1'. The overall system N applies rule ResO and evolves to $[\ \Gamma_p :_{\{role\}} 0 \parallel CF_1' \parallel CF_2\ ]^{\triangleright f_{\mathsf{fwd}}}$ with the label $\Gamma \triangleright \overline{(\Pi_1 \vee (\mathsf{role} = \mathsf{fwd})) \wedge ((\mathsf{role} \neq \mathsf{fwd}))}(p, \tilde{v})$ which is equivalent to $\Gamma \triangleright \overline{\Pi_1}(p, \tilde{v})$. Notice that Γ is equivalent to $\Gamma_p \downarrow I$. The forwarder CF_1' forwards its message to nearby forwarders, in our case this is CF_2. The overall system applies again rule ResO and evolves to $[\ \Gamma_p :_{\{role\}} 0 \parallel CF_1'' \parallel CF_2'\ ]^{\triangleright f_{\mathsf{fwd}}}$ with the label $\Gamma \triangleright \overline{(\mathsf{role} = \mathsf{fwd}) \wedge (\mathsf{role} \neq \mathsf{fwd})}(p, \tilde{v})$. This message is private and is perceived externally as a silent move. The overall system terminates after another internal action (performed by the second forwarder) and by emitting the message, $\Gamma \triangleright \overline{\Pi_1}(p, \tilde{v})$[3], two more times, one from each forwarder. By applying the rule ResO twice, the system evolves to $[\ \Gamma_p :_{\{role\}} 0 \parallel \Gamma_1 :_{\{role\}} 0 \parallel \Gamma_2 :_{\{role\}} 0\ ]^{\triangleright f_{\mathsf{fwd}}}$.

To prove that the advertising message is propagated to all clients in the network it is sufficient to show that each forwarder takes its turn in spreading the message. Formally it is sufficient to prove that the behaviour of the overall system is bisimilar to the behaviour of a test component T with $\Gamma_t(role) = \mathsf{fwd}$, defined below, which is able to send the same message three times sequentially and then terminates.

$$T = \Gamma_t :_{\{role\}} (p,\ \tilde{v})@\Pi_1.(p,\ \tilde{v})@\Pi_1.(p,\ \tilde{v})@\Pi_1.0$$

[3]Since role can only assume the values client and fwd, we have that $(\mathsf{role} = \mathsf{client}) \wedge (\mathsf{role} \neq \mathsf{fwd}) \doteq \mathsf{role} = \mathsf{client}) = \Pi_1$

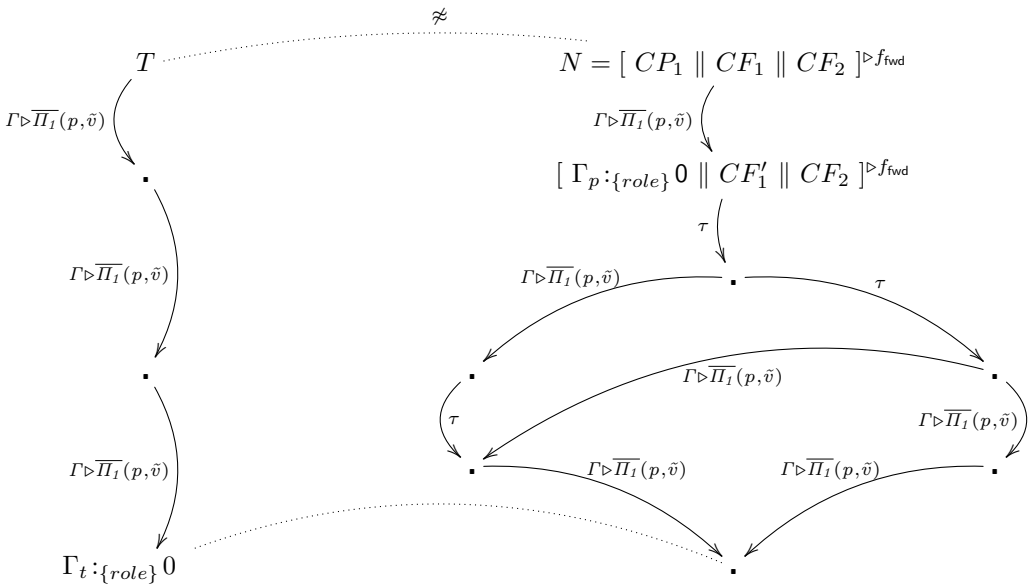

Figure 3: System N simulates the test component T, but initial interference is possible, Hence $N \not\approx T$

Figure 3 shows that system N weakly simulates component T, but they are not bisimilar, i.e., $T \not\approx N$. This is because a forwarder is initially prepared to accept any message $(a, \tilde{v})$ such that a coincides with the id of one of its neighbours. For instance if we put another component, say $CP_2 = \Gamma_h :_{\{\}} (f_3,\ \tilde{w})@(\mathsf{tt}).0$, there is a possibility that CF_1 first receives a message from CP_2 and the system can evolve as follows:

$$N \| CP_2 \xrightarrow{\{\}\triangleright\overline{\mathsf{tt}}(f_3,\tilde{w})} \xrightarrow{\Gamma\triangleright\overline{\Pi_1}(p,\tilde{v})} \xrightarrow{\Gamma\triangleright\overline{\mathsf{ff}}()} \xrightarrow{\Gamma\triangleright\overline{\mathsf{ff}}()} \xrightarrow{\Gamma\triangleright\overline{\Pi_1}(f_3,\tilde{w})} \xrightarrow{\Gamma\triangleright\overline{\Pi_1}(f_3,\tilde{w})}$$

while

$$T \| CP_2 \xrightarrow{\{\}\triangleright\overline{\mathsf{tt}}(f_3,\tilde{w})} \xrightarrow{\Gamma\triangleright\overline{\Pi_1}(p,\tilde{v})} \xrightarrow{\Gamma\triangleright\overline{\Pi_1}(p,\tilde{v})} \xrightarrow{\Gamma\triangleright\overline{\Pi_1}(p,\tilde{v})}$$

and it is easy to see that $N \| CP_2 \not\approx T \| CP_2$. One way to avoid interference and ensure that the property holds is to limit input capabilities of N:

$$N = [\ [\ CP_1 \parallel CF_1 \parallel CF_2\ ]^{\triangleright f_{\mathsf{fwd}}}\ ]^{\triangleleft g_*}$$

where $g_*(\Gamma, \tilde{v}) = \mathsf{ff}$ for any Γ and $\tilde{v}$. Under this restriction N is not able to receive any message and $N \approx T$.

Remark 5.3 (AbC restriction versus name restriction)**.** AbC *restriction operators are different from the name restriction operators in CCS-style [20] and π-style [3, 21] and here we outline the main differences.*

The restriction operators in AbC *are static and do not facilitate scope extrusion as mentioned before. On one hand, our style of restriction is more general than the CCS-style in that predicates can either be exposed (e.g., $f(\Gamma, \tilde{v}) = \mathsf{tt}$), constrained (e.g., $f(\Gamma, \tilde{v}) = \Pi$), or completely hidden (e.g., $f(\Gamma, \tilde{v}) = \mathsf{ff}$). Actually, the CCS-style operator can be encoded as follows: $f(\emptyset, \alpha) = \mathsf{tt}$ if $\alpha \in L$ for some action α and set L. On the other hand,* AbC *restriction and π-style restriction are actually different. It should be clear that further constraining of a channel in π-like restriction can be complicated, i.e., a channel can either be available to all components listening to it or only within a specific scope. In other words, two components cannot receive the same message because of different reasons as the case in the* AbC*-style, i.e., consider $f(\Gamma, \tilde{v}) = (a \neq 1)$ and a sender inside the scope of f sends a message with predicate $(c = 2 \vee a = 1)$, thus components inside the scope of f can receive because they satisfy either $(c = 2)$ or $(a = 1)$ while components outside the scope can only receive when they satisfy $(c = 2)$.*

Due to the interplay between names and predicates, considering name restriction in AbC *would introduce technical difficulties that outweigh its benefits. Actually, both local and scoped interaction are naturally supported in* AbC *either through local interaction between processes inside a component or through external interaction within static scopes. However, studying the benefits of using name restriction in* AbC *would be an interesting topic for future work.*

6. Encoding channel-based interaction

In this section, we briefly discuss the expressiveness of AbC and its relationships to other existing approaches. The main purpose is to give an intuition on applications where AbC might be simpler and more natural to use. Formal separation results between AbC and other related approaches are beyond the scope of this article; but they certainly deserve further investigations. First, we discuss how *group-based* [22, 23, 24] and *publish/subscribe-based* [25, 26] interaction patterns can be naturally rendered in AbC[4], then we concentrate on the encoding of a simple process calculus, featuring broadcast and channel-passing. Finally, we prove the correctness of the encoding using the behavioural theory presented in Section 4.

In the group-based model, when an agent wants to send a message, it attaches the group name/id in the message and only members of that group can receive the message when it is propagated. To model this interaction pattern in AbC, group names can be rendered as attributes and the constructs for joining or leaving a given group can be modelled as attribute updates. It is worth mentioning that a possible encoding of group communication into $b\pi$-calculus has been introduced in [11]. The encoding is relatively complicated and does not guarantee the causal order of message reception. "Locality" is neither a first class construct in $b\pi$-calculus nor in AbC. However, "locality" (the group name, in this case) can be naturally modelled as an attribute in AbC; in $b\pi$-calculus, more effort is needed.

In the publish/subscribe model, there are two types of agents: publishers and subscribers and there is an exchange server that mediates the interaction between them. Publishers produce messages tagged with topics and send them to the exchange server which is responsible for filtering and forwarding these messages to interested subscribers. Subscribers simply register their interests to the exchange server and based on their interests they receive messages. A natural modelling of the topic-based publish/subscribe model [26] with AbC can be accomplished by allowing publishers to broadcast messages with "tt" predicates (i.e., satisfied by all subscribers) and only subscribers can check the compatibility of the exposed publishers attributes with their subscriptions. The distinguishing features of such an AbC model with respect to others are due to the ease of specifying the discovery of the right place to subscribe and that the selective and implicit multicast interaction is cleaner than the point-to-point one.

In the next subsection we will show in full details how it is possible to model broadcast communication with channel-passing in AbC, but first we would like to spend some words about the difficulties that other calculi have in mimicking situations that are naturally expressed in AbC.

Clearly, classical calculi (that rely on fixed channels [3]) may not be able to express behaviours arising from AbC specifications while others with sort of channel-equivalence [5, 6, 9] may have potential. However, such an encoding (if it exists) would be much more complicated and less intuitive.

Here, we would like to argue that encoding the combination of (1) knowledge representation and (2) multiparty predicate-based interaction might prove to be rather complicated with existing calculi. The former facilities modelling a notion of interdependence between co-located processes while the latter facilitates modelling a notion of selective multicast where receivers engage in interactions because they satisfy the sender predicate. Consider the following simple AbC system:

$$\Gamma_1 :_{\{b,c\}} (\mathsf{m})@(\Pi_1 \vee \Pi_2).P_1 \quad \| \quad \overbrace{\Gamma_2 :_{\{b,c\}} (\mathsf{tt})(x).P_2}^{C_2} \quad \| \quad \overbrace{\Gamma_3 :_{\{b,c\}} (()@\mathsf{ff}.[this.a := 5]P_3 \mid (b \leq this.a)(x).Q)}^{C_3}$$

[4]Further details about the way these two communication paradigms are modelled in AbC can be found in [13].

If we assume that $\Gamma_2 \downarrow \{b, c\} \models \Pi_1$, $\Gamma_3 \downarrow \{b, c\} \models \Pi_2$, $\Gamma_1(b) = 3$ and $\Gamma_2(a) = 2$, we have that when the first process in C_3 changes the value of attribute a to "5", the second process in C_3 gains the opportunity of receiving message "m" from the first component. One could argue that using name-restriction to hide local communication and relying on bound input/output actions would be sufficient to encode such kind of behaviour in "fixed" channel-based process calculi like π-calculus. However, this is not the case because bound input/output actions can engage in interaction only when they are instantiated with concrete channel names. In the example above, the input action of the second process C_3 is always enabled. This means that before the update, an input is available on the predicate $b \leq 2$ and after the update it is available on the predicate $b \leq 5$. The difficulty of encoding this behaviour in classical process calculi is due to their lack of knowledge representation and rigidity of interfaces, i.e., a process does not have a dedicated knowledge and cannot communicate based on its local knowledge. The reason is because that all actions (internal or external) are treated similarly, i.e., they all have representation in the underlying LTS; τ for internal actions and names for external ones. Thus, due to interleaving there is always a chance for a process to miss an incoming message because it cannot inspect its local knowledge and adjust its interface instantaneously.

We can use the same example to argue that this behaviour might not be easily expressed in process calculi that support knowledge representation, e.g., [5, 21, 6, 9]. Actually, it is easy to see that when the message m is sent from the first component, component C_2 receives it because its attribute environment satisfies Π_1 while C_3 receives it because it satisfies Π_2, i.e., the same message can be received because of different conditions. This is due to the flexibility of the interaction predicates and their evaluation with respect to the local states of targeted components.

Of course there is room to argue that this example could be mimicked by relying on a combination of channel-equivalence [5], fusion [6], or constraints [9] and name restriction to limit the scope of interaction. However, such an encoding (if it exists) must be hard-coded in the structure of the specifications by relying on the extensive use of name restriction. This does not only affect readability but also complicates reasoning about the evolution of scopes boundaries as side-effects of interactions. *AbC* provides a clear separation between the concept of a scope (or locality) and the logic of the program while compositionally managing external interactions by message-passing. Thus each component has a clear external interface to offer to the others. Furthermore, these models in general do not support broadcast and thus it would be difficult to model its precise effects with them. Note that a special form of indirect group interaction through shared knowledge arising from fusions, frames, etc., can be expressed. However, it would not provide a fine-grained control to coordinate the behaviour of the interacting components the way broadcast would.

Broadcast Psi [21], on the other hand, can express adaptation arising either from the flexible connectivity of its broadcast channels or from evolving local state, but it has trouble doing both at the same time as the above example suggests. This will be made clear in Section 7 where we will provide additional details.

6.1. Encoding of broadcasting channels into AbC

We consider now the issue of encoding of broadcast *channel-passing* into *AbC*. Our main purpose is to find a common ground that relates *AbC* to standard process calculi rather than claiming superiority or separability. The reason is that some design choices in *AbC* are completely different from those of standard calculi. For instance, the notion of restricting the interaction in *AbC* is different from the standard notion of name-restrictions or fresh names. It is actually not obvious how π-like name-restriction would be beneficial in *AbC*. Furthermore, *AbC* has a structured syntax that forbids action-prefixing and nondeterminism to occur top-level and thus it does not make sense to compare it with a calculus with a flat syntax. From our point of view it is natural to use a structured syntax because usually we describe distributed systems as a set of components executing in parallel and influence each other through message-passing.

It may seem tempting to model a channel name as an *AbC* attribute, however this turns out not to be the right approach. To clarify this point we propose an encoding where each process is rendered as an *AbC* component. The attributes of a component are boolean and they represent the set of all possible channel names plus an additional attribute CH to be used to coordinate channels allocation among co-located processes. The interface I of a component has exactly the set of channel names in Γ, i.e., $I = \mathsf{dom}(\Gamma)$. Initially CH is enabled and other attributes are disabled. A possible encoding of broadcast receive and send actions are reported below:

(Component Level)

$(\!|G|\!)_c \triangleq [(\!|G|\!)_p] \qquad (\!|P_1 \| P_2|\!)_c \triangleq (\!|P_1|\!)_c \parallel (\!|P_2|\!)_c$

(Process Level)

$(\!|\mathtt{nil}|\!)_p \triangleq 0 \qquad (\!|\tau.G|\!)_p \triangleq ()@\mathsf{ff}.(\!|G|\!)_p$

$(\!|a(\tilde{x}).G|\!)_p \triangleq \Pi(y,\tilde{x}).(\!|G|\!)_p$
$\quad \mathtt{with} \quad \Pi = (y = a) \quad \mathtt{and} \quad y \notin n((\!|G|\!)_p)$

$(\!|\bar{a}\tilde{x}.G|\!)_p \triangleq (a,\tilde{x})@(\mathsf{tt}).(\!|G|\!)_p$

$(\!|(rec\ A\langle\tilde{x}\rangle.G)\langle\tilde{y}\rangle|\!)_p \triangleq A(\tilde{x})$
$\quad \mathtt{where} \quad A(\tilde{x}) \triangleq (\!|G|\!)_p \quad \text{and} \quad fn((\!|G|\!)_p) \subseteq \{\tilde{x}\}$
$(\!|G_1 + G_2|\!)_p \triangleq (\!|G_1|\!)_p + (\!|G_2|\!)_p$

(Labels)

$(\!|\bar{a}\tilde{z}|\!) \triangleq \{\} \triangleright \overline{\mathsf{tt}}(a,\tilde{z}) \qquad (\!|a(\tilde{z})|\!) \triangleq \{\} \triangleright \mathsf{tt}(a,\tilde{z})$
$(\!|\tau|\!) \triangleq \{\} \triangleright \overline{\mathsf{ff}}()$
$(\!|\bar{a}\tilde{z}:|\!) \triangleq \{\} \triangleright \mathsf{tt}(a,\tilde{z}) \qquad (\!|a(\tilde{z}):|\!) \triangleq \{\} \triangleright \mathsf{tt}(a,\tilde{z})$
$(\!|\tau:|\!) \triangleq \{\} \triangleright \mathsf{ff}()$

Table 6: Encoding PA-calculus into AbC

$$a(\tilde{x}).P \triangleq \Gamma :_I (a = \mathsf{tt})(\tilde{x}).(\!|P|\!)$$
$$\bar{a}(\tilde{v}).P \triangleq \Gamma :_I \langle \mathsf{CH} = \mathsf{tt}\rangle()@\mathsf{ff}.[a := \mathsf{tt},\ \mathsf{CH} = \mathsf{ff}](\tilde{v})@\mathsf{tt}.[a := \mathsf{ff},\ \mathsf{CH} = \mathsf{tt}](\!|P|\!)$$

Intuitively, to receive on channel a, a process checks if the exposed attribute a of the sender is enabled and only then accepts the message and evolves to $(\!|P|\!)$. To send on channel a, a process first checks if attribute CH is enabled, i.e., last message-sending is completed. Only then the process sets a to tt and turns off CH silently with a message on a false predicate. Afterwards the process sends the values of the message on a true predicate and as side effects it resets a and CH to their initial values.

The problem with the above encoding is that it does not support channel-passing, i.e., one cannot use a received channel name for a future broadcast. To overcome this problem, we would need to extend AbC with a construct to dynamically add/remove attributes to/from the attribute environment of a component. We can avoid such modification by exploiting the fact that the receiving predicates in AbC can also check the values contained in the received message. The key idea is to use structured messages to select interacting partners where the name of the channel is the first element of any messages; receivers only accept those messages that have attached channels that match their receiving channels. Actually, attributes do not play any role in such interaction so we can consider components with empty environments and empty interfaces i.e., $\emptyset :_\emptyset P$. This encoding is similar to the one proposed in [27] which naturally models channel-matching of the transitions in the rewriting rules as pattern-matching of structured messages.

To show feasibility of the approach just outlined, we encoded in AbC a process algebra, we call PA-calculus, (inspired by CBS [4] and bπ [11]). We consider the set of processes P as shown below.

$$P ::= G \mid P_1 \| P_2$$
$$G ::= \mathtt{nil} \mid a(\tilde{x}).G \mid \bar{a}\tilde{x}.G \mid G_1 + G_2 \mid (rec\ A\langle\tilde{x}\rangle.G)\langle\tilde{y}\rangle$$

Unlike CBS [4] and bπ [11], the syntax of PA consists of two-levels where the parallel composition occurs top-level like in AbC. This is important to make a fair comparison with AbC and to focus on comparing the

distinguishing features between AbC and classical channel-passing calculi. Name restriction is not considered because name restriction and AbC restriction are of different nature as explained in Remark 5.3. Furthermore, the two-level syntax discards the capability of dynamic process creation where parallel composition appears in the scope of a recursive process. Indeed AbC does not support dynamic creation of components in its current version. However, dynamic creation of components can be supported in a straightforward way. We can introduce another action, say fork, that can be used to create a component at run-time. Thus, a component can send, receive, and also create a component. Note that dynamic process creation inside a single component is already supported in AbC. The semantics of PA-calculus is the same of bπ [11] but the name restriction rules are removed.

Now, we may proceed describing the encoding as reported in Table 6. The encoding of a process P is rendered as an AbC component $(\!|P|\!)_c$ with $\Gamma = I = \emptyset$. In what follows, we use $[G]$ to denote a component with empty Γ and I, i.e., $\emptyset :_\emptyset \mathsf{G}$. Note that $(\!|G|\!)_c$ encodes a sequential process, $(\!|P|\!)_c$ encodes the parallel composition of sequential processes while $(\!|\alpha|\!)$ encodes an interaction label. The discard action label α: of PA-calculus is encoded as an input label in AbC because discarding is handled in AbC at process level. The channel is rendered as the first element in the sequence of values. For instance, in the output action $(a, \tilde{x})@(\mathsf{tt})$, a represents a channel name, so the input action $(y = a)(y, \tilde{x})$ will always check the first element of the received values to decide whether to accept or discard the message. Send predicates do not play any role in the encoding and for this reason send predicates are always true.

6.2. Correctness of the encoding

In this section, we provide the correctness proof of the encoding presented in Section 6.1. The criteria we use to assess our encoding are inspired by the work in [28]. Before we proceed with the proof, we first fix some notations and we define the properties that our encoding preserves.

Definition 6.1 (Divergence)**.** *P diverges, written $P \Uparrow$, iff $P \rightarrow^\omega$ where ω denotes an infinite number of reductions.*

Definition 6.2 (Uniform Encoding)**.** *An encoding $(\!|\ \mathbf{.}\ |\!) : \mathsf{PA} \rightarrow$ AbC is uniform if it enjoys the following properties:*

1. *(Homomorphic w.r.t. parallel composition): $(\!|P \| Q|\!) \triangleq (\!|P|\!) \| (\!|Q|\!)$*
2. *(Name invariance): $(\!|P\sigma|\!) \triangleq (\!|P|\!)\sigma$, for any permutation of names σ.*
3. *(Faithfulness): $P \Uparrow$ iff $(\!|P|\!) \Uparrow$*
4. *Operational correspondence*
    1. *(Operational completeness): if $P \xrightarrow{\alpha} P'$ then $(\!|P|\!) \xrightarrow{(\!|\alpha|\!)} \simeq (\!|P'|\!)$ where $\simeq$ is the strong barbed equivalence of* AbC*.*
    2. *(Operational soundness): if $(\!|P|\!) \xrightarrow{(\!|\alpha|\!)} Q$ then there exists a P' such that $P \xrightarrow{\alpha} P'$ and $Q \simeq (\!|P'|\!)$, where $\simeq$ is the strong barbed equivalence of* AbC*.*

Basically, when translating a term from PA-calculus into AbC, we expect that the translation is compositional and independent from contexts; is independent from the names of the source term (i.e., name invariance); preserves parallel composition (i.e., homomorphic w.r.t. '$\|$'); is faithful in the sense that it preserves divergence. Moreover, the encoding has to translate output (input) actions of PA-terms into corresponding output (input) AbC actions, and has to preserve the operational correspondence between the source and target calculus. This includes that the translation has to be complete (i.e., every computation of the source term can be mimicked by its translation) and sound (i.e., every computation of a translated term corresponds to some computation of its source term).

The main distinguishing features of our encoding with respect to the one in [28] is that: our encoding is stronger in the sense that every transition of the source calculus is mimicked by exactly one transition in the

destination. It also preserves labelled transitions as stated in the operational correspondence, Definition 6.2, 4. This will be crucial to prove that the encoding is sound and complete with respect to labelled bisimilarity as stated in Corollary 6.1 and Corollary 6.2 later. However, our encoding does not preserve the barb of the source term, i.e., $P \Downarrow_a$ iff $(\!|P|\!) \Downarrow_{(\!|a|\!)}$ does not hold. This is due to the fact that we encode channels as special values in the message and basically the only barb observed in any translated term is $\mathtt{tt}$. This means that this encoding preserves the labelled transitions, but not the barbs. It is worth mentioning that the absence of this condition does not weaken the encoding because labels are preserved.

Now we provide a sketch of the proof of the operational correspondence and we report a detailed one in the Appendix B.

Lemma 6.1 (Operational Completeness)**.** *If* $P \xrightarrow{\alpha} P'$ *then* $(\!|P|\!)_c \xrightarrow{(\!|\alpha|\!)} \simeq (\!|P'|\!)_c$.

Proof. (Sketch) The proof proceeds by induction on the structure of the derivation $\xrightarrow{\alpha}$. We have several cases depending on the last applied rule. We only consider one case of parallel composition when communication happens: $P_1 \| P_2 \xrightarrow{\bar{a}\tilde{z}} P_1' \| P_2'$. By applying the induction hypothesis on the premises $P_1 \xrightarrow{\bar{a}\tilde{z}} P_1'$ and $P_2 \xrightarrow{a(\tilde{z})} P_2'$, we have that $(\!|P_1|\!)_c \xrightarrow{(\!|\bar{a}\tilde{z}|\!)} \simeq (\!|P_1'|\!)_c$ and $(\!|P_2|\!)_c \xrightarrow{(\!|a(\tilde{z})|\!)} \simeq (\!|P_2'|\!)_c$. We can apply rule ComL.

$$\frac{(\!|P_1|\!)_c \xrightarrow{\{\}\triangleright\overline{\mathtt{tt}}(a,\tilde{z})} (\!|P_1'|\!)_c \qquad (\!|P_2'|\!)_c \xrightarrow{\{\}\triangleright\mathtt{tt}(a,\tilde{z})} (\!|P_2'|\!)_c}{(\!|P_1|\!)_c \parallel (\!|P_2|\!)_c \xrightarrow{\{\}\triangleright\overline{\mathtt{tt}}(a,\tilde{z})} (\!|P_1'|\!)_c \parallel (\!|P_2'|\!)_c}$$

Now, it is easy to see that: $(\!|P_1'\|P_2'|\!)_c \simeq (\!|P_1'|\!)_c \| (\!|P_2'|\!)_c$. □

Lemma 6.2 (Operational Soundness)**.** *If* $(\!|P|\!)_c \xrightarrow{(\!|\alpha|\!)} Q$ *then* $\exists P'$ *such that* $P \xrightarrow{\alpha} P'$ *and* $Q \simeq (\!|P'|\!)_c$.

Proof. (Sketch) The proof proceeds by induction on the structure of the derivation $\xrightarrow{(\!|\alpha|\!)}$. We have several cases depending on the last applied rule. We only consider one case of parallel composition when communication takes place: $(\!|P_1\|P_2|\!)_c \xrightarrow{\{\}\triangleright\overline{\mathtt{tt}}(a,\tilde{z})} (\!|P_1'|\!)_c \| (\!|P_2'|\!)_c$ where $(\!|P_1|\!)_c \xrightarrow{(\!|\bar{a}\tilde{z}|\!)} (\!|P_1'|\!)_c$ and $(\!|P_2|\!)_c \xrightarrow{(\!|a(\tilde{z})|\!)} (\!|P_2'|\!)_c$. By the induction hypothesis on the transitions $(\!|P_1|\!)_c \xrightarrow{(\!|\bar{a}\tilde{z}|\!)} (\!|P_1'|\!)_c$ and $(\!|P_2|\!)_c \xrightarrow{(\!|a(\tilde{z})|\!)} (\!|P_2'|\!)_c$ we have that there exist P' and P'' such that $P_1 \xrightarrow{\bar{a}\tilde{z}} P'$, $P_2 \xrightarrow{a(\tilde{z})} P''$, $(\!|P_1'|\!)_c \simeq (\!|P'|\!)_c$ and $(\!|P_2'|\!)_c \simeq (\!|P''|\!)_c$. We can apply rule (11) Table 4 [11]:

$$\frac{P_1 \xrightarrow{\bar{a}\tilde{z}} P' \qquad P_2 \xrightarrow{a(\tilde{z})} P''}{P_1 \| P_2 \xrightarrow{\bar{a}\tilde{z}} P' \| P''}$$

Because $(\!|P_1'|\!)_c \simeq (\!|P'|\!)_c$ and $(\!|P_2'|\!)_c \simeq (\!|P''|\!)_c$ we have that $(\!|P'\|P''|\!)_c \simeq (\!|P_1'|\!)_c \| (\!|P_2'|\!)_c$ as required. □

Theorem 6.1. *The encoding* $(\!|\,\mathbf{.}\,|\!) : \mathsf{PA} \to \mathrm{AbC}$ *is uniform.*

Proof. Definition 6.2(1) and 6.2(2) hold by construction. Definition 6.2(4) holds by Lemma 6.1 and Lemma 6.2. Definition 6.2(3) holds easily and as a result of the strong formulation of operational correspondence in Lemma 6.1, Lemma 6.2, this encoding cannot introduce divergence. □

As a result of Theorem 4.3, Theorem 6.1 and of the strong formulations of Lemma 6.1, Lemma 6.2, this encoding is sound and complete with respect to bisimilarity as stated in the following corollaries.

Corollary 6.1 (Soundness w.r.t bisimilarity)**.**

- $(\!|P|\!)_c \approx (\!|Q|\!)_c$ *implies* $P \approx Q$

Corollary 6.2 (Completeness w.r.t bisimilarity)**.**

- $P \approx Q$ *implies* $(\!|P|\!)_c \approx (\!|Q|\!)_c$

7. Related Work

In this section, we report on related works concerning calculi with primitives that either model multiparty interaction or provide interesting ways to establish interaction. Many calculi have been proposed to model interactions in distributed systems. However, the interaction mechanisms used in the Psi-calculus [5], the fusion calculus [6] and the cc-pi calculus [9] are the closest ones to AbC. Below, we concentrate on the relationships between AbC and these calculi and only briefly discuss its relationships with others.

Psi-calculus is an extension of the π-calculus with the aim to serve as a meta-theory for process calculi in general. Once the Ψ-parameters are instantiated to obtain a new calculus, the behavioural theory of the new instance is fully-defined and there is no need to redo all of the proofs from scratch and develop a new theory. The Psi-calculus is heavily influenced by the π-calculus and its main extensions like the Applied π-calculus [29], the fusion calculus [6], and the cc-pi-calculus [9]. However, the Psi-calculus is more expressive in some aspects and provides a compositional and carefully developed operational semantics. This is not the case for the Applied π-calculus, the fusion calculus, and the cc-pi calculus which heavily rely on structural congruence to postulate a lot of properties about the semantics. In Psi, AbC, and π these properties are derived and proved based on the labelled semantics. This makes the proofs much simpler as there is only one inductive definition of transitions.

In Psi-calculus the subjects of communication actions are generalised to data terms and interaction is based on a dynamic notion of channel-equivalence rather than on name-matching. The environment/knowledge is encoded as a special process, named assertion, which influences the behaviour of the process within its scope. The assertion, denoted by $(\!|\Psi|\!)$, can be made local to a specific process by means of name restriction, e.g., $(\nu\tilde{b})(P \mid (\!|\Psi|\!))$ where $\tilde{b}$ is a sequence of names that are bound in Ψ and possibly in P. A local assertion (assertion with local names) is called a frame $\mathcal{F}$ and represents the information that a process make available to other processes operating in parallel.

As mentioned before, Psi-calculus has been heavily influenced by the design choices of π-calculus as evident from the extensive use of name-restriction to hard-code different notions. Everything floats in the structure of processes and a restriction operator (in the π-calculus style) is used to create a local knowledge, to share knowledge, etc. For instance, the environment/knowledge is hard-coded in Psi by relying on name-restriction: The term $(\nu\tilde{b})(P \mid (\!|\Psi|\!))$ represents a process P with environment $(\!|\Psi|\!)$ and with name-restriction used to localise $(\!|\Psi|\!)$ to P. This choice does not distinguish between *knowledge* and *behaviour* and thus affects readability and maintainability. Actually the scope $\nu\tilde{b}$ might fully or partially localise $(\!|\Psi|\!)$ and this makes the notion of local knowledge a fluid concept. In AbC, this can be modelled as $\Gamma_\Psi :_I P$ and by simple syntactic checks, one could easily infer that Γ_Ψ is a local environment for process P and $\Gamma_\Psi \downarrow I$ is the part that is exposed to possible partners (only) at the time of interaction. This clean separation between knowledge and behaviour in AbC avoids dependencies between components and enhances readability, compositional reasoning, and maintainability.

The evolution of knowledge is complicated in Psi. For instance, the process $\overline{M}N.P$ does not contribute any knowledge to the environment at this stage while after executing the output action $\overline{M}N$ the process P might do. For technical reasons, the process $!P$ does not add any knowledge to the environment while a finite number of the same P, running in parallel, adds the knowledge obtained as the composition of frames of $P|\ldots|P$. In AbC, knowledge evolution is cleanly defined through attribute updates, taking place as side-effects of interactions.

Furthermore, Psi-calculus is based on a point-to-point communication model that, unlike AbC's implicit multicast model, does not scale when multiparty coordination is needed. And this is particularly important for open systems where agents may join or leave at any time while systems continue their execution. In [30], we laid the basis for an efficient, correct, and distributed communication infrastructure to manage the interaction of the attribute-based paradigm. There we enable multiple components to coordinate asynchronously while preserving the semantics of AbC.

Broadcast Psi [21] is an extension of Psi with broadcast primitives to generally facilitate both point-to-point and broadcast interactions in a single framework. Unlike Broadcast Psi, AbC is mainly designed to tackle systems which interact based on novel and dynamic group-based approaches. For this reason, AbC primitives provide natural support to modelling applications in the CAS domain. The meta-theory

of AbC is also much simpler in that it avoids the usual technical difficulties when dealing name-restriction, which is made possible without compromising the expressive power of AbC. Note that name-restriction is a core primitive in Psi-like calculi because it is the only way to model knowledge, local interaction, and dynamic communication links. Furthermore, broadcast Psi relies on a notion of connectivity predicates, it uses $M \dot{\prec} K$ to denote that an output prefix $\overline{M}N$ is out-connected to channel K while it uses $K \dot{\succ} M$ to denote that an input prefix $\underline{M}(\lambda\tilde{x})N$ is in-connected to channel K. This way a process may either adapt by choosing to connect/disconnect from a broadcast channel through these predicates (and thus provides an adaptable connectivity of broadcast channels) or adapt by considering its local state. However, it is not easy to program processes that consider both possibilities at the same time. The reason is that if a term M representing a broadcast endpoint adapts based on a local name x, M must contain x in its support, i.e., $x \in n(M)$. M can be connected to other endpoints via the broadcast connectivity relation, but only through a broadcast channel with strictly greater support than both endpoints; that is, only through channels that mention x. Hence this channel cannot be used for communication across the scope boundary of x.

Also modelling reconfiguration in broadcast Psi is not easy; a proposal is put forward in [21] but its scalability is an issue. The proposal models different connectivity configurations using assertions, and relies on generation of assertions tagged by a fresh generation number; only the most recent generation is used. A generation becomes obsolete when composed with an assertion from later generation.

For an example of reconfiguration that should also clarify the different levels of abstraction of Psi-calculi and AbC, consider a topology controller $T = (\!|d \leftarrow\!\!\shortmid \langle 1, \emptyset\rangle|\!) \mid \tau.((\!|d \leftarrow\!\!\shortmid \langle 2, \{K \dot{\succ} M, K \dot{\succ} N\}\rangle|\!) \mid \tau.((\!|d \leftarrow\!\!\shortmid \langle 3, \{K \dot{\succ} M\}\rangle|\!))$. In process $P \mid T$, the process P broadcasts on K while T manages the topology. Initially $\mathcal{F}(T) = \langle 1, \emptyset\rangle$ and sending/reception on K is not possible, if T evolves with $T \xrightarrow{\tau} T'$ then $\mathcal{F}(T') = \langle 2, \{K \dot{\succ} M, K \dot{\succ} N\}\rangle$ and a broadcast on K can be received on both M and N and after $T' \xrightarrow{\tau} T''$, a broadcast can be received only on M, since $\mathcal{F}(T'') = \langle 3, \{K \dot{\succ} M\}\rangle$. Note that the interpretation of $\mathcal{F}(T)$ considers only the latest generation of assertions. To make T local to P, one needs to restrict channel d and thus $(\nu d)(P \mid T)$ can send, receive, and discard messages based on the influence of T. At this point, the reader should be convinced that all of this hard-coding of knowledge can be modelled simply in AbC by a component $\Gamma :_I Q$ where all send and receive operations in Q are guarded by the value of attribute d that can be changed locally with an attribute update; the behaviour of Q is thus parametric to Γ.

The fusion calculus [6] is introduced to simplify the π-calculus and to provide a canonical calculus of concurrency. Interestingly, as fusion simplifies π it gains more expressiveness. It changes the concepts of sending and receiving in the π-calculus to what is called *fusion*. Fusions can be thought of as bubbles that when collide create larger ones. Thus when the transition $\overline{u}vw.P \mid uxy.Q \xrightarrow{\{v=x, w=y\}} P \mid Q$ is executed, a fusion (or a bubble) is created that includes both P and Q and both share the facts that $v = x,\ w = y$. Furthermore as result of interaction a new persistent shared knowledge that relates input to output variables is introduced. This knowledge can be used for further interactions, it evolves at run-time and influences interacting processes. It is easy to see that this is somehow similar to the concept of frame in the Psi-calculus and it is not surprisimng that these two calculi are strongly related. As in Psi, name restriction is used to localise the knowledge to a specific scope. For instance, the transition $\nu x(\overline{u}vw.P \mid \nu y(uxy.Q \mid R) \mid S) \xrightarrow{\{\}} (P \mid (Q \mid R)\,[w/y] \mid S)[v/x]$ clearly shows the complicated evolution of knowledge. Initially x is shared between all running processes while y is limited to a specific scope. However, when specific parts of the system interact they may produce global side effects, thus we have that S is not involved in the interaction but it gets to know that x becomes equivalent to v and has no choice but to accept the new modifications. It is clear that the notions of compositionality and knowledge are fluid in such settings, i.e., there is no clear separation between knowledge and behaviour. The above described behaviour cannot be expressed in AbC and it would be interesting to find out how to accommodate fusions in a cleaner way. It is worth mentioning that fusions (also frames of Psi) facilitate some kind of group communication through shared stores but these stores should be hard-coded in the structure of the specifications using name restriction. Furthermore, group communication through shared memory is not as clean and controlled as the one of message-passing. Clearly AbC only allows shared memory inside a single component and different components may only influence each other by explicit message-passing. This is natural for a distributed system where a single components is programmed by a single person while different components might be programmed by different people. While a fine-grained

control on shared variables can be enforced inside a single component through a "lock-based" mechanism, it is more challenging when considering independent components with dynamic and evolving set of shared variables as in the above mentioned calculi. Thus, it is highly recommended to rely on clear interfaces for interacting with the external world.

The cc-pi calculus [9] generalises fusions like $x = v$ to *named constraints* and adds some of concurrent constraint programming primitives [7, 8] to hande such constraints. In named constraints, variables are regarded as ordinary names in the π-calculus style. Constraints are put in parallel as a special form of a passive process and they can be manipulated by blocking actions like ask, tell, etc. Two processes interact by performing an output $\overline{x}\langle y\rangle$ and an input $x'\langle y'\rangle$ only if the constraint $x = x'$ is entailed by the store (i.e., the constraints in parallel). The result of synchronisation is a new constraint $y = y'$ which is added to the store. Note that the store is based on constraints and is different from the attribute environment in AbC. Furthermore constraints are different from the interaction predicates in AbC where predicates select interaction partners. In fact, constraints may not be directly used to select partners but rather other interactions might be enabled as a result of the changes in the shared store. Like Psi and fusion, cc-pi heavily relies on name restriction to localise constraints and its flexibility come from a sort of shared "memory".

Psi, fusion, and cc-pi gain in expressivity by exploiting a sort of shared memory and adequate scoping mechanisms but this complicates reasoning about their models. In AbC we understand the importance of message-passing and shared memory paradigms. We strongly believe that their combined effects are crucial to model distributed systems more accurately. However, we are more conservative in this direction and limit sharing to local behaviours while keeping message-passing for external communication. Thus AbC components have clear communication interfaces with the external world, this simplifies systems decomposition and reasoning about individual components separately.

Let us now consider in turn some of the other calculi, namely CBS [31], $b\pi$ [11], CPC [32], attribute π-calculus [33], imperative π-calculus [34], Set-Pi [35] and Broadcast Quality Calculus of [36].

The CBS calculus [31] captures the essential features of broadcast communication in a simple and natural way. Whenever a process transmits a value, all processes running in parallel, and ready to perform an input, do catch the broadcast. In [37], an LTS for CBS was proposed where notions of strong and weak labelled bisimilarity relying on a discard relation were defined. Unlike AbC, the communication links in CBS are static and cannot change at run-time.

The $b\pi-$calculus [11] equips $\pi-$calculus [3] with broadcast primitives where only agents listening on a specific channel can receive the broadcast. The authors also proposed an LTS relying on a discard relation and a labelled bisimilarity which is proved to coincide with the reduction barbed congruence when closed under substitutions. Channel mobility in $b\pi$ permits creating dynamic communication links at run-time, but this dynamicity is still limited because it requires processes to agree on a specific channel name to establish the interactions. $b\pi$ has no notion of knowledge representation, this can only be hard-coded as another parallel process, closed under name-restriction. For instance, a term $(\nu\tilde{a})(K_1\| \ldots \|K_n\|P)$ represents a process P with knowledge $K_1\| \ldots \|K_n$ and the scope $(\nu\tilde{a})$ ensures that the knowledge is local to P. The main weakness of this approach, other than being low-level, is the fact that, like in π, every interaction is an LTS transition and there is no distinction between local or global transitions. This implies that a process has to expose a local interaction as a silent transition. Because of interleaving there is always a chance that a process misses an external message because it could not adjust its communication interface instantaneously based on a local interaction, e.g., between P and some K_i. Psi-calculus [5] mitigates this by introducing a notion of *assertion*.

The CPC calculus [32] uses a point-to-point pattern-matching to select partners. This is not just matching equivalence of names, but it also extends to the structure of terms and thus enables a notion of unification that permits a two-way, or symmetric, flow of information. This differentiates CPC from AbC and other standard process calculi which instead rely on complimentary actions (i.e., input/output) to exchange information in one direction at a time.

The attribute π-calculus [33] aims at constraining point-to-point interaction by considering values of communication attributes. A λ-function is associated to each receiving action and communication takes place only if the evaluation of the function with the provided input falls within a predefined set of values.

The imperative π-calculus [34] is a recent extension of the attribute π-calculus with a global store and

with imperative programs used to specify constraints.

The Set-Pi [35] extends the applied π-calculus by a notion of sets of messages and thus allows processes to specify how to store, lookup and manipulate information. While this calculus is convenient to deal with databases of objects, it is as expressive as the applied π because sets can be simulated using private channels.

The broadcast Quality Calculus of [36] deals with the problem of denial-of-service by means of *selective* input actions. It inspects the structure of messages by associating specific contracts to inputs, but does not provide any means to change the input contracts during execution.

AbC combines the lessons learnt from the above mentioned languages and calculi in the sense that *AbC* strives for expressivity while preserving minimality and simplicity. The dynamic settings of attributes and the possibility of inspecting/modifying the environment gives *AbC* greater flexibility and expressivity while keeping models as much natural as possible.

8. Concluding Remarks and Future Works

We have introduced a foundational process calculus, named *AbC*, for modelling interactions in CAS systems by relying on attribute-based communication. We tested the expressive power of *AbC* by discussing how other interaction paradigms, such as group based communication and publish-subscribe, could be modelled in our calculus. Moreover, we used *AbC* as the target of the encoding of a process algebra (inspired by CBS [4] and [11]) and proved the correctness of the encoding up to our equivalence.

For future work, we want to formally study the relation between *AbC* and other closely-related calculi, i.e., Psi calculus [5], Fusion calculus [6], and cc-pi calculus [9]. We would also like to understand the impact of having a π-like restriction operator in *AbC*. We conjecture that this extension will require the introduction of an operator to modify the exposed environment $\Gamma \downarrow I$ of a component at run-time.

We plan to investigate the impact of alternative behavioural relations like testing preorders in terms of equational laws, proof techniques, etc. We also want to consider the challenging problem of verifying collective properties of *AbC* code. As a step in this direction, we developed the ReCiPe framework [38] (an extension and symbolic representation of *AbC* specifications) and we extended LTL to be able to specify collective properties. Another line of research worth investigating is anonymity at the level of attribute identifiers. Clearly, *AbC* achieves dynamicity and openness in the distributed settings, which is an advantage compared to channel-based models. In our model, components are anonymous; however the "name-dependency" challenge arises at another level, that is, the level of attribute environments. In other words, the sender's predicate should be aware of the identifiers of receiver's attributes in order to explicitly use them. For instance, the sending predicate $(loc = (1, 4))$ targets the components at location $(1, 4)$. However, different components might use different identifiers names (i.e., "location") to denote their locations; this requires that there should be an agreement about the attribute identifiers used by the components. For this reason, appropriate mechanisms for handling *attribute directories* together with identifiers matching/correspondence will be considered. These mechanisms will be particularly useful when integrating heterogeneous applications.

Another research direction is introducing a static semantics for *AbC* as a way to discipline the interaction between components. This way we can answer questions regarding deadlock freedom and the fact that messages have the type the receiver expects.

Appendix A. Proofs

Appendix A.1. Proof of Lemma 3.1

We prove each statement separately.

1. **We need to prove that for any λ such that $\lambda = \Gamma' \triangleright \Pi(\tilde{v})$ and $\Pi \doteq \mathsf{ff}$, then $C \xrightarrow{\lambda} C$.** We proceed by induction on the syntax of C.

Base of Induction: $C = \Gamma :_I P$. It is sufficient to prove that $\Gamma :_I P \xmapsto{\widetilde{\Gamma' \triangleright \Pi(\tilde{v})}} \Gamma :_I P$ where $\Pi \doteq \mathsf{ff}$. This can be done by induction on the transition $\Gamma :_I P \xmapsto{\widetilde{\Gamma' \triangleright \Pi(\tilde{v})}} \hat{C}$ where $\Pi \doteq \mathsf{ff}$. We have the following cases.

Case 1: $P = \mathsf{0}$**.** We can only apply rule FZero regardless of Π and we have that $\Gamma :_I \mathsf{0} \xmapsto{\widetilde{\Gamma' \triangleright \Pi(\tilde{v})}} \Gamma :_I \mathsf{0}$ as required.

Case 2: $P = \Pi_1(\tilde{x}).U$**.** We can only apply rule FRcv because $\Gamma \downarrow I \not\models \Pi$ (Notice that $\Pi \doteq \mathsf{ff}$) and we have that $\Gamma :_I \Pi_1(\tilde{x}).U \xmapsto{\widetilde{\Gamma' \triangleright \Pi(\tilde{v})}} \Gamma :_I \Pi_1(\tilde{x}).U$ as required.

Case 3: $P = (\tilde{E})@\Pi_1.U$**.** We can only apply rule FBrd regardless of Π and we have that $\Gamma :_I (\tilde{E})@\Pi_1.U \xmapsto{\widetilde{\Gamma' \triangleright \Pi(\tilde{v})}} \Gamma :_I (\tilde{E})@\Pi_1.U$ as required.

Case 4: $P = \langle \Pi_1 \rangle P$**.** We can either apply rule FAware1 if $\Gamma \not\models \Pi_1$ or rule FAware2 otherwise by the induction hypothesis on the premise $\Gamma :_I P \xmapsto{\widetilde{\Gamma' \triangleright \Pi(\tilde{v})}} \Gamma :_I P$ of rule FAware2 and in both cases we have that $\Gamma :_I \langle \Pi_1 \rangle P \xmapsto{\widetilde{\Gamma' \triangleright \Pi(\tilde{v})}} \Gamma :_I \langle \Pi_1 \rangle P$ as required.

Case 5: $P = P_1 + P_2$**.** We can only apply rule FSum by the induction hypothesis on its premises $\Gamma :_I P_1 \xmapsto{\widetilde{\Gamma' \triangleright \Pi(\tilde{v})}} \Gamma :_I P_1$ and $\Gamma :_I P_2 \xmapsto{\widetilde{\Gamma' \triangleright \Pi(\tilde{v})}} \Gamma :_I P_2$ and we have that $\Gamma :_I P_1 + P_2 \xmapsto{\widetilde{\Gamma' \triangleright \Pi(\tilde{v})}} \Gamma :_I P_1 + P_2$ as required.

Case 6: $P_1 | P_2$**.** It is proved in a similar case of Case 5 by applying rule FInt instead.

Case 7: $P = K(\tilde{y})$**.** It is proved by applying rule FRec and by the induction hypothesis on the premise $\Gamma :_I P[\tilde{y}/\tilde{x}] \xmapsto{\widetilde{\Gamma' \triangleright \Pi(\tilde{v})}} \Gamma :_I P[\tilde{y}/\tilde{x}]$ of FRec.

Hence, we have that $\Gamma :_I P \xmapsto{\widetilde{\Gamma' \triangleright \Pi(\tilde{v})}} \Gamma :_I P$ where $\Pi \doteq \mathsf{ff}$. By applying rule FComp, we have that $\Gamma :_I P \xrightarrow{\Gamma' \triangleright \Pi(\tilde{v})} \Gamma :_I P$ when $\Pi \doteq \mathsf{ff}$.

Inductive Hypothesis:. Let us assume that for any C_1 and C_2, and for any λ such that $\lambda = \Gamma' \triangleright \Pi(\tilde{v})$ and $\Pi \doteq \mathsf{ff}$, then $C_i \xrightarrow{\lambda} C_i$

Inductive Step:. We have to consider three cases: $C = C_1 \| C_2$, $C = [\ C_i\ ]^{\triangleright f}$ and $C = [\ C_i\ ]^{\triangleleft f}$.

Case 1: $C = C_1 \| C_2$**.** The statement follows directly from the inductive hypothesis by applying rule Sync:

$$\frac{C_1 \xrightarrow{\lambda} C_1 \quad C_2 \xrightarrow{\lambda} C_2}{C_1 \| C_2 \xrightarrow{\lambda} C_1 \| C_2}$$

Case 2: $C = [\ C_i\ ]^{\triangleright f}$**.** This case follows from the inductive hypothesis ($C_i \xrightarrow{\lambda} C_i$) by applying rule ResO and by observing that if $\Pi \doteq \mathsf{ff}$ then for any Π', $\Pi \wedge \Pi' \doteq \mathsf{ff}$.

Case 3: $C = [\ C_i\ ]^{\triangleleft f}$**.** Exactly in the previous case by considering rule ResI.

2. **We need to prove that if** $C_1 \xrightarrow{\lambda} C_1'$ **and** $\lambda = \tau$**, then** $C_1 \| C \xrightarrow{\tau} C_1' \| C$ **and** $C \| C_1 \xrightarrow{\tau} C \| C_1'$**.** The statement follows directly from the previous point by applying rules ComL and ComR.

3. **We need to prove that if** $C_1 \Rightarrow C_1'$ **then** $C_1 \| C \Rightarrow C_1' \| C$ **and** $C \| C_1 \Rightarrow C \| C_1'$**.** We prove the statement ($C_1 \Rightarrow C_1'$ then $C_1 \| C \Rightarrow C_1' \| C$) and the statement ($C_1 \Rightarrow C_1'$ then $C \| C_1 \Rightarrow C \| C_1'$) can be proved in a symmetric way. The proof proceeds by induction on the length of the derivation $\Rightarrow_w$.

Base case, $w = 0$**:** We have that $C_1 \Rightarrow_0 C_1$ and $C_1 \| C \Rightarrow_0 C_1 \| C$ as required.

Inductive Hypothesis: we assume that $\forall k \leq w : C_1 \Rightarrow_k C_1'$ then $C_1 \| C \Rightarrow_k C_1' \| C$.

Inductive Step: Let $C_1 \Rightarrow_{w+1} C_1'$. By definition of $\Rightarrow_{w+1}$, we have that there exists C_1'' such that $C_1 \xrightarrow{\tau} C_1''$ and $C_1'' \Rightarrow_w C_1'$. By the second statement of this lemma, we have that if $C_1 \xrightarrow{\tau} C_1''$ then $C_1 \| C \xrightarrow{\tau} C_1'' \| C$. Moreover, by the induction hypothesis $C_1'' \| C \Rightarrow_w C_1' \| C$. Hence, we have that $C_1 \Rightarrow C_1'$ then $C_1 \| C \Rightarrow C_1' \| C$ as required.

4. We need to prove that if $C_1 \xRightarrow{\Gamma \triangleright \Pi_1(\tilde{v})} C_1'$ and $\Pi_1 \simeq \Pi_2$ then $C_1 \xRightarrow{\Gamma \triangleright \Pi_2(\tilde{v})} C_1'$ We first need to prove the single-step version of this statement: if $C_1 \xrightarrow{\Gamma \triangleright \Pi_1(\tilde{v})} C_1'$ and $\Pi_1 \simeq \Pi_2$ then $C_1 \xrightarrow{\Gamma \triangleright \Pi_2(\tilde{v})} C_1'$. The proof proceeds by induction on C_1.

Case 1: $C_1 = \Gamma :_I P$**.** We have to proceed by induction on the length of the derivation of the transitions $\Gamma :_I P \xmapsto{\Gamma' \triangleright \Pi_1(\tilde{v})} \Gamma' :_I P'$ and $\Gamma :_I P \xmapsto{\widetilde{\Gamma' \triangleright \Pi_1(\tilde{v})}} \Gamma :_I P$.

- We start by the transition $\Gamma :_I P \xmapsto{\widetilde{\Gamma' \triangleright \Pi_1(\tilde{v})}} \Gamma :_I P$. The cases when rules FBrd, FAware1, FZero are trivial since they refuse regardless of the sender predicate. The other discard rules can be proved by the induction hypothesis on their premises. The interesting case is when rule FRcv is applied. In this case $P = \Pi(\tilde{x}).U$ and we have that $\Gamma :_I \Pi(\tilde{x}).U \xmapsto{\widetilde{\Gamma' \triangleright \Pi_1(\tilde{v})}} \Gamma :_I \Pi(\tilde{x}).U$ only if $\Gamma' \not\models \{\Pi[\tilde{v}/\tilde{x}]\}_\Gamma$ or $\Gamma \downarrow I \not\models \Pi_1$. Now we need to show that $\Gamma \downarrow I \not\models \Pi_2$, but this is immediate from Definition 2.1 and the fact that $\Pi_1 \simeq \Pi_2$ and we have that $\Gamma :_I \Pi(\tilde{x}).U \xmapsto{\widetilde{\Gamma' \triangleright \Pi_2(\tilde{v})}} \Gamma :_I \Pi(\tilde{x}).U$. By applying rule FComp, we have that we have that $\Gamma :_I P \xrightarrow{\Gamma' \triangleright \Pi_1(\tilde{v})} \Gamma :_I P$ implies $\Gamma :_I P \xrightarrow{\Gamma' \triangleright \Pi_2(\tilde{v})} \Gamma :_I P$ such that $\Pi_1 \simeq \Pi_2$ as required.
- Now we prove the transition $\Gamma :_I P \xmapsto{\Gamma' \triangleright \Pi_1(\tilde{v})} \Gamma' :_I P'$. The interesting case is when rule Rcv is applied and other cases are proved by the induction hypothesis on their premises. In this case $P = \Pi(\tilde{x}).U$ and we have that $\Gamma :_I \Pi(\tilde{x}).U \xmapsto{\Gamma' \triangleright \Pi_1(\tilde{v})} \{\!|\Gamma :_I U[\tilde{v}/\tilde{x}]|\!\}$ only if $\Gamma' \models \{\Pi[\tilde{v}/\tilde{x}]\}_\Gamma$ or $\Gamma \downarrow I \models \Pi_1$. Now we need to show that $\Gamma \downarrow I \models \Pi_2$, but this is immediate from Definition 2.1 and the fact that $\Pi_1 \simeq \Pi_2$ and we have that $\Gamma :_I \Pi(\tilde{x}).U \xmapsto{\Gamma' \triangleright \Pi_2(\tilde{v})} \{\!|\Gamma :_I U[\tilde{v}/\tilde{x}]|\!\}$. By applying rule Comp, we have that we have that $\Gamma :_I P \xrightarrow{\Gamma' \triangleright \Pi_1(\tilde{v})} \Gamma' :_I P'$ implies $\Gamma :_I P \xrightarrow{\Gamma' \triangleright \Pi_2(\tilde{v})} \Gamma' :_I P'$ such that $\Pi_1 \simeq \Pi_2$ as required.

Hence, we have that if $\Gamma :_I P \xrightarrow{\Gamma \triangleright \Pi_1(\tilde{v})} \Gamma' :_I P'$ and $\Pi_1 \simeq \Pi_2$ then $\Gamma :_I P \xrightarrow{\Gamma \triangleright \Pi_2(\tilde{v})} \Gamma' :_I P'$ as required.

Case 2: $C = C_3 \| C_4$**.** We can only use rule Sync and by the induction hypothesis on the premises of Sync to prove this case.

Case 3: $C = [\ C_3\ ]^{\triangleleft f}$ **or** $C = [\ C_3\ ]^{\triangleright f}$**.** The statement follows directly from the inductive hypothesis by applying rules ResO and ResI and from the fact that $\Pi_1 \simeq \Pi_2$ implies that $\Pi_1 \wedge \Pi \simeq \Pi_2 \wedge \Pi$ for any predicate Π.

Let $C_1 \xRightarrow{\Gamma \triangleright \Pi_1(\tilde{v})} C_1'$. This means that there exits C_2 and C_2' such that:

$$C_1 \Rightarrow C_2 \xrightarrow{\Gamma \triangleright \Pi_1(\tilde{v})} C_2' \Rightarrow C_1'$$

We have already proved that $C_2 \xrightarrow{\Gamma \triangleright \Pi_2(\tilde{v})} C_2'$ for any $\Pi_2 \simeq \Pi_1$. Hence:

$$C_1 \Rightarrow C_2 \xrightarrow{\Gamma \triangleright \Pi_2(\tilde{v})} C_2' \Rightarrow C_1'$$

from which we can finally infer that $C_1 \xRightarrow{\Gamma \triangleright \Pi_2(\tilde{v})} C_1'$.

5. We prove that if $C_1 \xrightarrow{\tau} C_1'$, then for any f: $[\ C_1\ ]^{\triangleright f} \xrightarrow{\tau} [\ C_1'\ ]^{\triangleright f}$ and $[\ C_1\ ]^{\triangleleft f} \xrightarrow{\tau} [\ C_1'\ ]^{\triangleleft f}$. If $C_1 \xrightarrow{\tau} C_1'$ then there exists Γ, $\tilde{v}$ and $\Pi \simeq \mathsf{ff}$ such that $C_1 \xrightarrow{\Gamma \triangleright \overline{\Pi}(\tilde{v})} C_1'$.

By applying rule ResO we have that for any f, $[\ C_1\ ]^{\triangleright f} \xrightarrow{\Gamma \triangleright \overline{\Pi \wedge \Pi'}(\tilde{v})} [\ C_1'\ ]^{\triangleright f}$ where $f(\Gamma, \tilde{v}) = \Pi'$. Since, $\Pi \simeq \mathsf{ff}$ we have that $\Pi \wedge \Pi' \simeq \mathsf{ff} \wedge \Pi' \simeq \mathsf{ff}$. Hence, $[\ C_1\ ]^{\triangleright f} \xrightarrow{\tau} [\ C_1'\ ]^{\triangleright f}$

We can also apply rule ResIPass to prove that $[\ C_1\ ]^{\triangleleft f} \xrightarrow{\Gamma \triangleright \overline{\Pi}(\tilde{v})} [\ C_1'\ ]^{\triangleleft f}$. That is $[\ C_1\ ]^{\triangleleft f} \xrightarrow{\tau} [\ C_1'\ ]^{\triangleleft f}$

6. We prove that if $C_1 \Rightarrow C_1'$, then for any f: $[\ C_1\ ]^{\triangleright f} \Rightarrow [\ C_1'\ ]^{\triangleright f}$ and $[\ C_1\ ]^{\triangleleft f} \Rightarrow [\ C_1'\ ]^{\triangleleft f}$. We prove the statement by induction on $C_1 \Rightarrow_w C_1'$.

Base case, $w = 0$: We have that $C_1 \Rightarrow_0 C_1$ while both $[\ C_1\ ]^{\triangleright f} \Rightarrow_0 [\ C_1\ ]^{\triangleright f}$ and $[\ C_1\ ]^{\triangleleft f} \Rightarrow_0 [\ C_1\ ]^{\triangleleft f}$ as required.

Inductive Hypothesis: we assume that $\forall k \leq w : C_1 \Rightarrow_k C_1'$ then: $[\ C_1\ ]^{\triangleright f} \Rightarrow_k [\ C_1'\ ]^{\triangleright f}$ and $[\ C_1\ ]^{\triangleleft f} \Rightarrow_k [\ C_1'\ ]^{\triangleleft f}$.

Inductive Step: Let $C_1 \Rightarrow_{w+1} C_1'$. By definition of $\Rightarrow_{w+1}$, we have that there exists C_1'' such that $C_1 \xrightarrow{\tau} C_1''$ and $C_1'' \Rightarrow_w C_1'$. By **item 5** of this Lemma, and by Inductive Hypothesis we have that for any f:

- $[\ C_1\ ]^{\triangleright f} \xrightarrow{\tau} [\ C_1''\ ]^{\triangleright f}$ and $[\ C_1''\ ]^{\triangleright f} \Rightarrow_w [\ C_1'\ ]^{\triangleright f}$;
- $[\ C_1\ ]^{\triangleleft f} \xrightarrow{\tau} [\ C_1''\ ]^{\triangleleft f}$ and $[\ C_1''\ ]^{\triangleleft f} \Rightarrow_w [\ C_1'\ ]^{\triangleleft f}$.

From the two above we have that $[\ C_1\ ]^{\triangleright f} \Rightarrow_{w+1} [\ C_1'\ ]^{\triangleright f}$ and $[\ C_1\ ]^{\triangleleft f} \Rightarrow_{w+1} [\ C_1'\ ]^{\triangleleft f}$.

Appendix A.2. Proof of Lemma 4.2

It is sufficient to prove that the relation $\mathcal{R} = \{(C_1 \| C, C_2 \| C) | \forall C_1, C_2, C \in \mathrm{Comp} : C_1 \approx C_2\}$ is a weak bisimulation. First of all we can observe that $\mathcal{R}$ is *symmetric*. It is easy to see that if $(C_1 \| C, C_2 \| C) \in \mathcal{R}$ then $(C_2 \| C, C_1 \| C) \in \mathcal{R}$. We have now to prove that for each $(C_1 \| C, C_2 \| C) \in \mathcal{R}$ and for each λ_1 such that $bn(\lambda_1) \cap fn(C_1, C_2) = \emptyset$:

$$C_1 \| C \xrightarrow{\lambda_1} C_3 \text{ implies } \exists \lambda_2 : \lambda_1 \simeq \lambda_2 \text{ such that } C_2 \| C \stackrel{\widehat{\lambda_2}}{\Longrightarrow} C_4 \text{ and } (C_3, C_4) \in \mathcal{R}.$$

We can observe that the transition $C_1 \| C \xrightarrow{\lambda_1} C_3$ can be derived by using one of rule among Sync, ComL, and ComR. The following cases can be distinguished:

Case 1: rule Sync is applied. In this case $\lambda_1 = \Gamma \triangleright \Pi_1(\tilde{v})$, $C_1 \xrightarrow{\Gamma \triangleright \Pi_1(\tilde{v})} C_1'$, $C \xrightarrow{\Gamma \triangleright \Pi_1(\tilde{v})} C'$ and $C_3 = C_1' \| C'$. Since $C_1 \approx C_2$, we have that $C_2 \stackrel{\lambda_2}{\Longrightarrow} C_2'$, with $\lambda_1 \simeq \lambda_2 = \Gamma \triangleright \Pi_2(\tilde{v})$ and $C_1' \approx C_2'$. This implies that there exists C_2'' and C_3''' such that $C_2 \Rightarrow C_2'' \xrightarrow{\lambda_2} C_2''' \Rightarrow C_2'$. Moreover, since $\lambda_1 \simeq \lambda_2$, by Lemma 3.1, we have that $C \xrightarrow{\lambda_2} C'$. By using again Lemma 3.1, we have that $C_2 \| C \Rightarrow C_2'' \| C$. We can apply rule Sync to prove that $C_2'' \| C \xrightarrow{\lambda_2} C_2''' \| C'$. Finally, as before, we have that $C_2''' \| C' \Rightarrow C_2' \| C'$. The statement follows by observing that $(C_1' \| C', C_2' \| C') \in \mathcal{R}$.

Case 2: rule ComL is applied. We can distinguish two cases: $\lambda_1 = \tau$, $\lambda_1 \neq \tau$. If λ_1 is a *silent* transition ($\lambda_1 = \tau$), we have that $\lambda_1 = \Gamma \triangleright \overline{\Pi_1}(\tilde{v})$ with $\Pi_1 \simeq \mathsf{ff}$. Moreover, $C_1 \xrightarrow{\lambda_1} C_1'$ and $C_3 = C_1' \| C$ (since $\Pi_1 \simeq \mathsf{ff}$, $C \xrightarrow{\Gamma \triangleright \Pi_1(\tilde{v})} C$). From the fact that $C_1 \approx C_2$, we have that $C_2 \Rightarrow C_2'$ and $C_1' \approx C_2'$. Moreover, by Lemma 3.1, we have that $C_2 \| C \Rightarrow C_2' \| C$. It is easy to observe that $(C_1' \| C, C_2' \| C) \in \mathcal{R}$.

If $\lambda_1 \neq \tau$, then $\lambda_1 = \Gamma \triangleright \overline{\Pi_1}(\tilde{v})$. We have that $C_1 \xrightarrow{\Gamma \triangleright \overline{\Pi_1}(\tilde{v})} C_1'$, $C \xrightarrow{\Gamma \triangleright \Pi_1(\tilde{v})} C'$ and $C_3 = C_1' \| C'$. In this case the statement follows similarly to **Case 1**. Indeed, $C_2 \xrightarrow{\Gamma \triangleright \overline{\Pi_2}(\tilde{v})} C_2'$, with $\Gamma \triangleright \overline{\Pi_1}(\tilde{v}) \simeq \Gamma \triangleright \overline{\Pi_2}(\tilde{v})$ and $C_1' \approx C_2'$. Moreover, by Lemma 3.1, $C \xrightarrow{\Gamma \triangleright \Pi_2(\tilde{v})} C'$. Hence, $C_2 \| C \xrightarrow{\Gamma \triangleright \overline{\Pi_2}(\tilde{v})} C_2' \| C'$ and $(C_1' \| C', C_2' \| C') \in \mathcal{R}$.

Case 3: rule ComR is applied. We can distinguish two cases: $\lambda_1 = \tau$, $\lambda_1 \neq \tau$. By Lemma 3.1 we have that $C \xrightarrow{\lambda_1} C'$ and $C_3 = C_1 \| C'$. Similarly, $C_2 \| C \stackrel{\lambda_1}{\Longrightarrow} C_2 \| C'$ and $(C_1 \| C', C_2 \| C') \in \mathcal{R}$. If $\lambda_1 \neq \tau$ we have that $\lambda_1 = \Gamma \rhd \overline{\Pi_1}(\tilde{v})$, $C_1 \xrightarrow{\Gamma \rhd \Pi_1(\tilde{v})} C_1'$, $C \xrightarrow{\Gamma \rhd \overline{\Pi_1}(\tilde{v})} C'$, and $C_3 = C_1' \| C'$. Like in the previous cases, by using the fact that $C_1 \approx C_2$ and by Lemma 3.1, we have that $C_2 \xrightarrow{\Gamma \rhd \Pi_1(\tilde{v})} C_2'$ and $C_1' \approx C_2'$, and that $C_2 \| C' \stackrel{\lambda_1}{\Longrightarrow} C_2' \| C'$. The statement follows by observing that also in this case $(C_1' \| C', C_2' \| C') \in \mathcal{R}$.

The strong case of bisimulation ($\sim$) follows in a similar way.

Proof of Lemma 4.3. We prove the lemma case by case. We start by the output restriction operator and we follow up with the input restriction one. For the first case, It is sufficient to prove that the relation

$$\mathcal{R} = \{([\ C_1\ ]^{\rhd f}, [\ C_2\ ]^{\rhd f}) | \text{ for all functions } f \text{ and all } C_1, C_2 \in \text{Comp} : C_1 \approx C_2\}$$

is a weak bisimulation. First of all we can observe that $\mathcal{R}$ is *symmetric*. We have now to prove that for each $([\ C_1\ ]^{\rhd f}, [\ C_2\ ]^{\rhd f}) \in \mathcal{R}$ and for each λ_1:

$$[\ C_1\ ]^{\rhd f} \xrightarrow{\lambda_1} C_3 \text{ implies } \exists \lambda_2 : \lambda_1 \backsimeq \lambda_2 \text{ such that } [\ C_2\ ]^{\rhd f} \stackrel{\widehat{\lambda}_2}{\Longrightarrow} C_4 \text{ and } (C_3, C_4) \in \mathcal{R}.$$

We can observe that the transition $[\ C_1\ ]^{\rhd f} \xrightarrow{\lambda_1} C_3$ can be derived either by using rule ResO or by using ResOPass.

When rule ResO is used, $\lambda_1 = \Gamma \rhd \overline{\Pi_1 \wedge \Pi}(\tilde{v})$ where $f(\Gamma, \tilde{v}) = \Pi$ and $C_1 \xrightarrow{\Gamma \rhd \overline{\Pi_1}(\tilde{v})} C_1'$. Since $C_1 \approx C_2$, we have that $C_2 \xrightarrow{\Gamma \rhd \overline{\Pi_2}(\tilde{v})} C_2'$, with $\Gamma \rhd \overline{\Pi_1}(\tilde{v}) \backsimeq \Gamma \rhd \overline{\Pi_2}(\tilde{v})$ and $C_1' \approx C_2'$. From the latter we have that, by definition of $\mathcal{R}$, $([\ C_1'\ ]^{\rhd f}, [\ C_2'\ ]^{\rhd f}) \in \mathcal{R}$.

We can now proceed by considering two cases: $\Pi_1 \backsimeq \mathsf{ff}$, $\Pi_1 \not\backsimeq \mathsf{ff}$.

If $\Pi_1 \backsimeq \mathsf{ff}$, we have that $C_2 \Rightarrow C_2'$. Directly from *Item 6* of Lemma 3.1 we have that $[\ C_2\ ]^{\rhd f} \Rightarrow [\ C_2'\ ]^{\rhd f}$.

If $\Pi_1 \not\backsimeq \mathsf{ff}$, we have that there exists C_2'' and C_3'''' such that $C_2 \Rightarrow C_2'' \xrightarrow{\Gamma \rhd \overline{\Pi_2}(\tilde{v})} C_2'''' \Rightarrow C_2'$. We have that:

- From *Item 6* of Lemma 3.1, $[\ C_2\ ]^{\rhd f} \Rightarrow [\ C_2''\ ]^{\rhd f}$;
- By applying rule ResO, $[\ C_2''\ ]^{\rhd f} \xrightarrow{\lambda_2} [\ C_2''''\ ]^{\rhd f}$ where $\lambda_1 \backsimeq \lambda_2 = \Gamma \rhd \overline{\Pi_2 \wedge \Pi}(\tilde{v})$;
- From *Item 6* of Lemma 3.1, $[\ C_2''''\ ]^{\rhd f} \Rightarrow [\ C_2'\ ]^{\rhd f}$.

That is, $[\ C_2\ ]^{\rhd f} \stackrel{\lambda_2}{\Longrightarrow} [\ C_2'\ ]^{\rhd f}$.

When rule ResOPass is used, we have that $\lambda_1 = \Gamma \rhd \Pi_1(\tilde{v})$, $C_1 \xrightarrow{\lambda_1} C_1'$ and $C_3 = [\ C_1'\ ]^{\rhd f}$. Since $C_1 \approx C_2$, we have that $C_2 \xrightarrow{\Gamma \rhd \Pi_2(\tilde{v})} C_2'$ and $C_1' \approx C_2'$ (and $([\ C_1'\ ]^{\lhd f}, [\ C_2'\ ]^{\lhd f}) \in \mathcal{R}$).

Hence we have that, $C_2 \Rightarrow C_2'' \xrightarrow{\Gamma \rhd \Pi_2(\tilde{v})} C_2'''' \Rightarrow C_2'$

By using Lemma 3.1, and by applying rule RuleOPass, we have that $[\ C_2\ ]^{\rhd f} \xrightarrow{\Gamma \rhd \Pi_2(\tilde{v})} [\ C_2'\ ]^{\rhd f}$, and the statement follows.

For the second case, It is sufficient to prove that the relation

$$\mathcal{R} = \{([\ C_1\ ]^{\lhd f}, [\ C_2\ ]^{\lhd f}) | \text{ for all functions } f \text{ and all } C_1, C_2 \in \text{Comp} : C_1 \approx C_2\}$$

is a weak bisimulation. First of all we can observe that $\mathcal{R}$ is *symmetric*. We have now to prove that for each $([\ C_1\ ]^{\lhd f}, [\ C_2\ ]^{\lhd f}) \in \mathcal{R}$ and for each λ_1:

$$[\ C_1\ ]^{\lhd f} \xrightarrow{\lambda_1} C_3 \text{ implies } \exists \lambda_2 : \lambda_1 \backsimeq \lambda_2 \text{ such that } [\ C_2\ ]^{\lhd f} \stackrel{\widehat{\lambda}_2}{\Longrightarrow} C_4 \text{ and } (C_3, C_4) \in \mathcal{R}.$$

We can observe that the transition $[\ C_1\ ]^{\triangleleft f} \xrightarrow{\lambda_1} C_3$ can be derived by using either rule ResI or ResIPass.

If rule ResI is used, we have that $\lambda_1 = \Gamma \triangleright \Pi_1(\tilde{v})$ where $f(\Gamma, \tilde{v}) = \Pi$ and $C_1 \xrightarrow{\Gamma \triangleright (\Pi_1 \wedge \Pi)(\tilde{v})} C_1'$. Since $C_1 \approx C_2$, we have that $C_2 \xRightarrow{\Gamma \triangleright (\Pi_2 \wedge \Pi)(\tilde{v})} C_2'$, with $\Pi_1 \doteq \Pi_2$ and $C_1' \approx C_2'$. This implies that there exists C_2'' and C_3''' such that $C_2 \Rightarrow C_2'' \xrightarrow{\Gamma \triangleright (\Pi_2 \wedge \Pi)(\tilde{v})} C_2''' \Rightarrow C_2'$. By using again Lemma 3.1, we have that $[\ C_2\ ]^{\triangleleft f} \Rightarrow [\ C_2''\ ]^{\triangleleft f}$. We can apply rule ResI to prove that $[\ C_2''\ ]^{\triangleleft f} \xrightarrow{\lambda_2} [\ C_2'''\ ]^{\triangleleft f}$ with $\lambda_2 = \Gamma \triangleright \Pi_2(\tilde{v})$. Finally, as before, we have that $[\ C_2'''\ ]^{\triangleleft f} \Rightarrow [\ C_2'\ ]^{\triangleleft f}$. The statement follows by observing that $([\ C_1'\ ]^{\triangleleft f}, [\ C_2'\ ]^{\triangleleft f}) \in \mathcal{R}$.

When rule ResIPass is used, we have that $\lambda_1 = \Gamma \triangleright \overline{\Pi_1}(\tilde{v})$, $C_1 \xrightarrow{\lambda_1} C_1'$ and $C_3 = [\ C_1'\ ]^{\triangleleft f}$. Since $C_1 \approx C_2$, we have that $C_2 \xRightarrow{\Gamma \triangleright \overline{\Pi_2}(\tilde{v})} C_2'$ and $C_1' \approx C_2'$. By using Lemma 3.1, and by applying rule RuleIPass, we have that $[\ C_2\ ]^{\triangleleft f} \xRightarrow{\Gamma \triangleright \overline{\Pi_2}(\tilde{v})} [\ C_2'\ ]^{\triangleleft f}$. This case follows directly from the fact that $C_1' \approx C_2'$ and $([\ C_1'\ ]^{\triangleleft f}, [\ C_2'\ ]^{\triangleleft f}) \in \mathcal{R}$).

The strong case of bisimulation ($\sim$) follows in a similar way. □

Appendix B. Detailed proofs about the encoding

Proof of Lemma 6.1. We have the following cases:

- if $P \triangleq \mathtt{nil}$: This case is immediate $(\!|\mathtt{nil}|\!)_c \triangleq [0]$
- if $P \triangleq \tau.G$: We have that $\tau.G \xrightarrow{\tau} G$ and it is translated to $(\!|\tau.G|\!)_c \triangleq [()@\mathsf{ff}.(\!|G|\!)_p]$. We can only apply rule Comp to mimic this transition.

$$\frac{[()@\mathsf{ff}.(\!|G|\!)_p] \xrightarrow{\{\}\triangleright \overline{\mathsf{ff}}()} [(\!|G|\!)_p]}{[()@\mathsf{ff}.(\!|G|\!)_p] \xrightarrow{\{\}\triangleright \overline{\mathsf{ff}}()} [(\!|G|\!)_p]}$$

  From Table 6, we have that $(\!|G|\!)_c = [(\!|G|\!)_p]$ as required. Notice that sending on a false predicate is not observable (i.e., a silent move).

- if $P \triangleq a(\tilde{x}).G$: We have that $a(\tilde{x}).G \xrightarrow{a(\tilde{z})} G[\tilde{z}/\tilde{x}]$ and it is translated to $(\!|a(\tilde{x}).Q|\!)_c \triangleq [\Pi(y, \tilde{x}).(\!|G|\!)_p]]$ where $\Pi = (y = a)$. We can only apply rule Comp to mimic this transition.

$$\frac{[\Pi(y, \tilde{x}).(\!|G|\!)_p] \xrightarrow{\{\}\triangleright \mathsf{tt}(a,\ \tilde{z})} [(\!|G|\!)_p[a/y,\ \tilde{z}/\tilde{x}]]}{[\Pi(y, \tilde{x}).(\!|G|\!)_p] \xrightarrow{\{\}\triangleright \mathsf{tt}(a,\ \tilde{z})} [(\!|G|\!)_p[a/y,\ \tilde{z}/\tilde{x}]]}$$

  From Table 6, It is not hard to see that: $(\!|G[\tilde{z}/\tilde{x}]|\!)_c \simeq [(\!|G|\!)_p[a/y,\ \tilde{z}/\tilde{x}]] \simeq [(\!|G|\!)_p[\tilde{z}/\tilde{x}]]$ since $y \notin n((\!|G|\!)_p])$.

- if $P \triangleq \bar{a}\tilde{x}.G$: The proof is similar to the previous case but by applying an output transition instead.
- The fail rules for **nil**, τ, input and output are proved in a similar way but with applying FComp instead.
- if $P \triangleq ((rec\ A\langle\tilde{x}\rangle).P)\langle\tilde{y}\rangle)$: This case is trivial.
- if $P \triangleq G_1 + G_2$: We have that either $G_1 + G_2 \xrightarrow{\alpha} G_1'$ or $G_1 + G_2 \xrightarrow{\alpha} G_2'$. We only consider the first case with $G_1 \xrightarrow{\alpha} G_1'$ and the other case follows in a similar way. This process is translated to $(\!|G_1 + G_2|\!)_c \triangleq [(\!|G_1|\!)_p + (\!|G_2|\!)_p]$. By applying the induction hypothesis on the premise $G_1 \xrightarrow{\alpha} G_1'$, we have that $(\!|G_1|\!)_c \xrightarrow{(\!|\alpha|\!)} \simeq (\!|G_1'|\!)_c$. By rule Comp or FComp we have that $(\!|G_1|\!)_c \xrightarrow{(\!|\alpha|\!)} \simeq (\!|G_1'|\!)_c$ if $(\!|G_1|\!)_c \xmapsto{(\!|\alpha|\!)} \simeq (\!|G_1'|\!)_c$. We can apply either rule Comp or rule FComp (i.e., when discarding) to mimic

this transition depending on the performed action. We consider the case of Comp only and the other case follows in a similar way.

$$\frac{[(\!|G_1|\!)_p] \overset{(\!|\alpha|\!)}{\longmapsto} [(\!|G_1'|\!)_p]}{\dfrac{[(\!|G_1|\!)_p + (\!|G_2|\!)_p] \overset{(\!|\alpha|\!)}{\longmapsto} [(\!|G_1'|\!)_p]}{[(\!|G_1|\!)_p + (\!|G_2|\!)_p] \overset{(\!|\alpha|\!)}{\longrightarrow} [(\!|G_1'|\!)_p]}}$$

Again $(\!|G_1'|\!)_c \simeq [\ (\!|G_1'|\!)_p]$

- if $P \triangleq P_1 \| P_2$: This process is translated to $(\!|\ P_1 \| P_2\ |\!)_c \triangleq (\!|P_1|\!)_c \| (\!|P_2|\!)_c$. We have different cases depending on the performed action in deriving the transition $P_1 \| P_2 \xrightarrow{\alpha} \hat{P}$.

  - $P_1 \| P_2 \xrightarrow{\bar{a}\tilde{z}} P_1' \| P_2'$: We have two cases, either $P_1 \xrightarrow{\bar{a}\tilde{z}} P_1'$ and $P_2 \xrightarrow{a(\tilde{z})} P_2'$ or $P_2 \xrightarrow{\bar{a}\tilde{z}} P_2'$ and $P_1 \xrightarrow{a(\tilde{z})} P_1'$. We only consider the first case and the other case follows in the same way. By applying the induction hypothesis on the premises $P_1 \xrightarrow{\bar{a}\tilde{z}} P_1'$ and $P_2 \xrightarrow{a(\tilde{z})} P_2'$, we have that $(\!|P_1|\!)_c \xrightarrow{(\!|\overline{a}\tilde{z}|\!)} \simeq (\!|P_1'|\!)_c$ and $(\!|P_2|\!)_c \xrightarrow{(\!|a(\tilde{z})|\!)} \simeq (\!|P_2'|\!)_c$. We can apply rule ComL.

    $$\frac{(\!|P_1|\!)_c \xrightarrow{\{\}\triangleright\overline{\mathsf{tt}}(a,\ \tilde{z})} (\!|P_1'|\!)_c \quad (\!|P_2'|\!)_c \xrightarrow{\{\}\triangleright\mathsf{tt}(a,\ \tilde{z})} (\!|P_2'|\!)_c}{(\!|P_1|\!)_c \parallel (\!|P_2|\!)_c \xrightarrow{\{\}\triangleright\overline{\mathsf{tt}}(a,\ \tilde{z})} (\!|P_1'|\!)_c \parallel (\!|P_2'|\!)_c}$$

    Again we have that: $(\!|P_1' \| P_2'|\!)_c \simeq (\!|P_1'|\!)_c \| (\!|P_2'|\!)_c$.

  - $P_1 \| P_2 \xrightarrow{a(\tilde{z})} P_1' \| P_2'$: By applying the induction hypothesis on the premises $P_1 \xrightarrow{a(\tilde{z})} P_1'$ and $P_2 \xrightarrow{a(\tilde{z})} P_2'$, we have that $(\!|P_1|\!)_c \xrightarrow{(\!|a(\tilde{z})|\!)} \simeq (\!|P_1'|\!)_c$ and $(\!|P_2|\!)_c \xrightarrow{(\!|a(\tilde{z})|\!)} \simeq (\!|P_2'|\!)_c$. We only can apply Sync to mimic this transition.

    $$\frac{(\!|P_1|\!)_c \xrightarrow{\{\}\triangleright\mathsf{tt}(a,\ \tilde{z})} (\!|P_1'|\!)_c \qquad (\!|P_2'|\!)_c \xrightarrow{\{\}\triangleright\mathsf{tt}(a,\ \tilde{z})} (\!|P_2'|\!)_c}{(\!|P_1|\!)_c \parallel (\!|P_2|\!)_c \xrightarrow{\{\}\triangleright\mathsf{tt}(a,\ \tilde{z})} (\!|P_1'|\!)_c \parallel (\!|P_2'|\!)_c}$$

    Again we have that: $(\!|P_1' \| P_2'|\!)_c \simeq (\!|P_1'|\!)_c \| (\!|P_2'|\!)_c$.

  - $P_1 \| P_2 \xrightarrow{\alpha} P_1' \| P_2$ if $P_1 \xrightarrow{\alpha} P_1'$ and $P_2 \xrightarrow{\alpha:}$ or $P_1 \| P_2 \xrightarrow{\alpha} P_1 \| P_2'$ if $P_2 \xrightarrow{\alpha} P_2'$ and $P_1 \xrightarrow{\alpha:}$. We consider only the first case and by applying the induction hypothesis on the premises $P_1 \xrightarrow{\alpha} P_1'$ and $P_2 \xrightarrow{\alpha:}$, we have that $(\!|P_1|\!)_c \xrightarrow{(\!|\alpha|\!)} \simeq (\!|P_1'|\!)_c$ and $(\!|P_2|\!)_c \xrightarrow{(\!|\alpha:|\!)} \simeq (\!|P_2|\!)_c$. We have many cases depending on the performed action:

    1. if $\alpha = \tau$ then $P_1 \| P_2 \xrightarrow{\tau} P_1' \| P_2$ with $P_1 \xrightarrow{\tau} P_1'$ and $P_2 \xrightarrow{\tau:}$. We can apply ComL and FComp to mimic this transition. Note that role FComp can only be applied if $(\!|P_2|\!)_c$ can discard which is the case by the induction hypothesis.

    $$\frac{(\!|P_1|\!)_c \xrightarrow{\{\}\triangleright\overline{\mathsf{ff}}()} (\!|P_1'|\!)_c \quad \dfrac{(\!|P_2|\!)_c \overset{\widetilde{\{\}\triangleright\mathsf{ff}()}}{\longmapsto} (\!|P_2|\!)_c}{(\!|P_2|\!)_c \xrightarrow{\{\}\triangleright\mathsf{ff}()} (\!|P_2|\!)_c}}{(\!|P_1|\!)_c \parallel (\!|P_2|\!)_c \xrightarrow{\{\}\triangleright\overline{\mathsf{ff}}()} (\!|P_1'|\!)_c \parallel (\!|P_2|\!)_c}$$

    and again we have that: $(\!|P_1' \| P_2|\!)_c \simeq (\!|P_1'|\!)_c \| (\!|P_2|\!)_c$.

    2. if $\alpha = a(\tilde{z})$ then $P_1 \| P_2 \xrightarrow{a(\tilde{z})} P_1' \| P_2$ with $P_1 \xrightarrow{a(\tilde{z})} P_1'$ and $P_2 \xrightarrow{a(\tilde{z}):}$. We can apply Sync and FComp to mimic this transition.

$$\frac{(\!|P_1|\!)_c \xrightarrow{\{\}\triangleright \mathtt{tt}(a,\ \tilde{z})} (\!|P_1'|\!)_c \quad \dfrac{(\!|P_2|\!)_c \xmapsto{\widetilde{\{\}\triangleright \mathtt{tt}(a,\ \tilde{z})}} (\!|P_2|\!)_c}{(\!|P_2|\!)_c \xrightarrow{\{\}\triangleright \mathtt{tt}(a,\ \tilde{z})} (\!|P_2|\!)_c}}{(\!|P_1|\!)_c \parallel (\!|P_2|\!)_c \xrightarrow{\{\}\triangleright \mathtt{tt}(a,\ \tilde{z})} (\!|P_1'|\!)_c \parallel (\!|P_2|\!)_c}$$

Again we have that: $(\!|P_1' \| P_2|\!)_c \simeq (\!|P_1'|\!)_c \| (\!|P_2|\!)_c$.

3. if $\alpha = \bar{a}\tilde{z}$ then $P_1 \| P_2 \xrightarrow{\bar{a}\tilde{z}} P_1' \| P_2$ with $P_1 \xrightarrow{\bar{a}\tilde{z}} P_1'$ and $P_2 \xrightarrow{\bar{a}\tilde{z}:}$. We can apply ComL and FComp. There is also the symmetric case for rule ComL.

$$\frac{[(\!|P_1|\!)_c \xrightarrow{\{\}\triangleright \overline{\mathtt{tt}}(a,\ \tilde{z})} (\!|P_1'|\!)_c \quad \dfrac{(\!|P_2|\!)_c \xmapsto{\widetilde{\{\}\triangleright \mathtt{tt}(a,\ \tilde{z})}} (\!|P_2|\!)_c}{(\!|P_2|\!)_c \xrightarrow{\{\}\triangleright \mathtt{tt}(a,\ \tilde{z})} (\!|P_2|\!)_c}}{(\!|P_1|\!)_c \parallel (\!|P_2|\!)_c \xrightarrow{\{\}\triangleright \overline{\mathtt{tt}}(a,\ \tilde{z})} (\!|P_1'|\!)_c \parallel (\!|P_2|\!)_c}$$

□

Proof of Lemma 6.2. We prove the important cases:

- if $(\!|P|\!)_c \triangleq (\!|\mathtt{nil}|\!)_c$: This case is immediate $[0] \simeq (\!|\mathtt{nil}|\!)_c$
- if $(\!|P|\!)_c \triangleq (\!|\tau.G|\!)_c$: We have that $(\!|\tau.G|\!)_c \xrightarrow{\{\}\triangleright(\mathtt{ff})} (\!|G|\!)_c$. We can apply rule (1) of Table 4 [11] and we have that $\tau.G \xrightarrow{\tau} G$ as required.
- if $(\!|P|\!)_c \triangleq (\!|a(\tilde{x}).G|\!)_c$: We have that $(\!|a(\tilde{x}).G|\!)_c \xrightarrow{\{\}\triangleright \mathtt{tt}(a,\tilde{z})} (\!|G|\!)_c[\tilde{z}/\tilde{x}]$. We can apply rule (2) of Table 4 [11] and we have that $a(\tilde{x}).G \xrightarrow{a(\tilde{z})} G[\tilde{z}/\tilde{x}]$ where $\{\}\triangleright \mathtt{tt}(a,\tilde{z}) = (\!|a(\tilde{z})|\!)$ and $(\!|G[\tilde{z}/\tilde{x}]|\!)_c \simeq (\!|G|\!)_c[\tilde{z}/\tilde{x}]$ as required.
- if $(\!|P|\!)_c \triangleq (\!|\bar{a}\tilde{z}.G|\!)_c$: We have that $(\!|\bar{a}\tilde{z}.G|\!)_c \xrightarrow{\{\}\triangleright \overline{\mathtt{tt}}(a,\tilde{z})} (\!|G|\!)_c$. We can apply rule (3) of Table 4 [11] and we have that $\bar{a}(\tilde{z}).G \xrightarrow{a(\tilde{z})} G$ where $\{\} \triangleright \overline{\mathtt{tt}}(a,\tilde{z}) = (\!|\bar{a}\tilde{z}|\!)$ as required.
- if $(\!|P|\!)_c \triangleq (\!|G_1 + G_2|\!)_c$: We have that $(\!|G_1 + G_2|\!)_c \xrightarrow{(\!|\alpha|\!)} (\!|G_1'|\!)_c \vee (\!|G_2'|\!)_c$. This can be proved rule (7) Table 4 [11] and by the induction hypothesis on its premises.
- if $(\!|P|\!)_c \triangleq (\!|P_1 \| P_2|\!)_c$: We have that different cases depending on the performed action $(\!|\alpha|\!)$ when $(\!|P_1 \| P_2|\!)_c \xrightarrow{(\!|\alpha|\!)} (\!|P_1' \| P_2'|\!)_c$ is executed.
  - if $(\!|\alpha|\!) = \{\} \triangleright \overline{\mathtt{tt}}(a,\tilde{z})$. We have different cases depending on whether the message is received or discarded
    - We have two cases with actual reception where either $(\!|P_1|\!)_c \xrightarrow{(\!|\bar{a}\tilde{z}|\!)} (\!|P_1'|\!)_c$ and $(\!|P_2|\!)_c \xrightarrow{(\!|a(\tilde{z})|\!)} (\!|P_2'|\!)_c$ or $(\!|P_1|\!)_c \xrightarrow{(\!|a(\tilde{z})|\!)} (\!|P_1'|\!)_c$ and $(\!|P_2|\!)_c \xrightarrow{(\!|\bar{a}\tilde{z}|\!)} (\!|P_2'|\!)_c$. We consider the first case and the other one is symmetrical. By the induction hypothesis on the transitions $(\!|P_1|\!)_c \xrightarrow{(\!|\bar{a}\tilde{z}|\!)} (\!|P_1'|\!)_c$ and $(\!|P_2|\!)_c \xrightarrow{(\!|a(\tilde{z})|\!)} (\!|P_2'|\!)_c$ we have that there exist P' and P'' such that $P_1 \xrightarrow{\bar{a}\tilde{z}} P'$ and $P_2 \xrightarrow{a(\tilde{z})} P''$ where $(\!|P_1'|\!)_c \simeq (\!|P'|\!)_c$ and $(\!|P_2'|\!)_c \simeq (\!|P''|\!)_c$. We can apply rule (11) Table 4 [11]:

      $$\frac{P_1 \xrightarrow{\bar{a}\tilde{z}} P' \qquad P_2 \xrightarrow{a(\tilde{z})} P''}{P_1 \| P_2 \xrightarrow{\bar{a}\tilde{z}} P' \| P''}$$

      Because $(\!|P_1'|\!)_c \simeq (\!|P'|\!)_c$ and $(\!|P_2'|\!)_c \simeq (\!|P''|\!)_c$ we have that $(\!|P' \| P''|\!)_c \simeq (\!|P_1' \| P_2'|\!)_c$ as required.

* We have two cases with discarding where either $(\!|P_1|\!)_c \xrightarrow{(\!|\bar{a}\tilde{z}|\!)} (\!|P_1'|\!)_c$ and $(\!|P_2|\!)_c \xrightarrow{(\!|\bar{a}\tilde{z}:|\!)} (\!|P_2|\!)_c$ or $(\!|P_1|\!)_c \xrightarrow{(\!|\bar{a}\tilde{z}:|\!)} (\!|P_1|\!)_c$ and $(\!|P_2|\!)_c \xrightarrow{(\!|\bar{a}\tilde{z}|\!)} (\!|P_2'|\!)_c$. We consider the first case and the other one is symmetrical. By the induction hypothesis on the transitions $(\!|P_1|\!)_c \xrightarrow{(\!|\bar{a}\tilde{z}|\!)} (\!|P_1'|\!)_c$ and $(\!|P_2|\!)_c \xrightarrow{(\!|\bar{a}\tilde{z}:|\!)} (\!|P_2|\!)_c$ we have that there exist P' and P'' such that $P_1 \xrightarrow{\bar{a}\tilde{z}} P'$ and $P_2 \xrightarrow{\bar{a}\tilde{z}:} P''$ where $(\!|P_1'|\!)_c \simeq (\!|P'|\!)_c$ and $(\!|P_2|\!)_c \simeq (\!|P''|\!)_c$. We can apply rule (12) Table 4 [11]:

$$\frac{P_1 \xrightarrow{\bar{a}\tilde{z}} P' \qquad P_2 \xrightarrow{\bar{a}\tilde{z}:} P''}{P_1 \| P_2 \xrightarrow{\bar{a}\tilde{z}} P' \| P''}$$

Because $(\!|P_1'|\!)_c \simeq (\!|P'|\!)_c$ and $(\!|P_2|\!)_c \simeq (\!|P''|\!)_c$ we have that $(\!|P' \| P''|\!)_c \simeq (\!|P_1' \| P_2|\!)_c$ as required.

– if $(\!|\alpha|\!) = \{\} \triangleright \mathsf{tt}(a, \tilde{z})$. We have different cases depending on whether the message is received or discarded.

* We have that $(\!|P_1|\!)_c \xrightarrow{\{\}\triangleright \mathsf{tt}(a,\tilde{z})} (\!|P_1'|\!)_c$ and $(\!|P_2|\!)_c \xrightarrow{\{\}\triangleright \mathsf{tt}(a,\tilde{z})} (\!|P_2'|\!)_c$. By the induction hypothesis on the transitions $(\!|P_1|\!)_c \xrightarrow{\{\}\triangleright \mathsf{tt}(a,\tilde{z})} (\!|P_1'|\!)_c$ and $(\!|P_2|\!)_c \xrightarrow{\{\}\triangleright \mathsf{tt}(a,\tilde{z})} (\!|P_2'|\!)_c$ we have that there exist P' and P'' such that $P_1 \xrightarrow{a(\tilde{z})} P'$ and $P_2 \xrightarrow{a(\tilde{z})} P''$ where $(\!|P_1'|\!)_c \simeq (\!|P'|\!)_c$ and $(\!|P_2'|\!)_c \simeq (\!|P''|\!)_c$. We can apply rule (10) Table 4 [11]:

$$\frac{P_1 \xrightarrow{a(\tilde{z})} P' \qquad P_2 \xrightarrow{a(\tilde{z})} P''}{P_1 \| P_2 \xrightarrow{a(\tilde{z})} P' \| P''}$$

Because $(\!|P_1'|\!)_c \simeq (\!|P'|\!)_c$ and $(\!|P_2'|\!)_c \simeq (\!|P''|\!)_c$ we have that $(\!|P' \| P''|\!)_c \simeq (\!|P_1' \| P_2'|\!)_c$ as required.

* We have three cases with discarding where either $(\!|P_1|\!)_c \xrightarrow{(\!|a(\tilde{z})|\!)} (\!|P_1'|\!)_c$ and $(\!|P_2|\!)_c \xrightarrow{(\!|a(\tilde{z}):|\!)} (\!|P_2|\!)_c$ or $(\!|P_1|\!)_c \xrightarrow{(\!|a(\tilde{z}):|\!)} (\!|P_1|\!)_c$ and $(\!|P_2|\!)_c \xrightarrow{(\!|a(\tilde{z})|\!)} (\!|P_2'|\!)_c$ or $(\!|P_1|\!)_c \xrightarrow{(\!|a(\tilde{z}):|\!)} (\!|P_1|\!)_c$ and $(\!|P_2|\!)_c \xrightarrow{(\!|a(\tilde{z}):|\!)} (\!|P_2|\!)_c$. We consider the first case and the other ones are symmetrical. By the induction hypothesis on the transitions $(\!|P_1|\!)_c \xrightarrow{(\!|a(\tilde{z})|\!)} (\!|P_1'|\!)_c$ and $(\!|P_2|\!)_c \xrightarrow{(\!|a(\tilde{z}):|\!)} (\!|P_2|\!)_c$ we have that there exist P' and P'' such that $P_1 \xrightarrow{a(\tilde{z})} P'$ and $P_2 \xrightarrow{a(\tilde{z}):} P''$ where $(\!|P_1'|\!)_c \simeq (\!|P'|\!)_c$ and $(\!|P_2|\!)_c \simeq (\!|P''|\!)_c$. We can apply rule (12) Table 4 [11]:

$$\frac{P_1 \xrightarrow{a(\tilde{z})} P' \qquad P_2 \xrightarrow{a(\tilde{z}):} P''}{P_1 \| P_2 \xrightarrow{\bar{a}\tilde{z}} P' \| P''}$$

Because $(\!|P_1'|\!)_c \simeq (\!|P'|\!)_c$ and $(\!|P_2|\!)_c \simeq (\!|P''|\!)_c$ we have that $(\!|P' \| P''|\!)_c \simeq (\!|P_1' \| P_2|\!)_c$ as required.

– if $(\!|\alpha|\!) = \{\} \triangleright \overline{\mathsf{ff}}()$. We have two cases where either $(\!|P_1|\!)_c \xrightarrow{(\!|\tau|\!)} (\!|P_1'|\!)_c$ and $(\!|P_2|\!)_c \xrightarrow{(\!|\tau:|\!)} (\!|P_2|\!)_c$ or $(\!|P_1|\!)_c \xrightarrow{(\!|\tau:|\!)} (\!|P_1|\!)_c$ and $(\!|P_2|\!)_c \xrightarrow{(\!|\tau|\!)} (\!|P_2'|\!)_c$. We consider the first case and the other one is symmetrical. By the induction hypothesis on the transitions $(\!|P_1|\!)_c \xrightarrow{(\!|\tau|\!)} (\!|P_1'|\!)_c$ and $(\!|P_2|\!)_c \xrightarrow{(\!|\tau:|\!)} (\!|P_2|\!)_c$ we have that there exist P' and P'' such that $P_1 \xrightarrow{\tau} P'$ and $P_2 \xrightarrow{\tau:} P''$ where $(\!|P_1'|\!)_c \simeq (\!|P'|\!)_c$ and $(\!|P_2|\!)_c \simeq (\!|P''|\!)_c$. We can apply rule (12) Table 4 [11]:

$$\frac{P_1 \xrightarrow{\tau} P' \qquad P_2 \xrightarrow{\tau:} P''}{P_1 \| P_2 \xrightarrow{\tau} P' \| P''}$$

Because $(\!|P_1'|\!)_c \simeq (\!|P'|\!)_c$ and $(\!|P_2'|\!)_c \simeq (\!|P''|\!)_c$ we have that $(\!|P' \| P''|\!)_c \simeq (\!|P_1' \| P_2'|\!)_c$ as required.

□

References

[1] A. Ferscha, Collective adaptive systems, in: Adjunct Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2015 ACM International Symposium on Wearable Computers, ACM, 2015, pp. 893–895.

[2] I. Sommerville, D. Cliff, R. Calinescu, J. Keen, T. Kelly, M. Kwiatkowska, J. Mcdermid, R. Paige, Large-scale complex it systems, Communications of the ACM 55 (7) (2012) 71–77.

[3] R. Milner, J. Parrow, D. Walker, A calculus of mobile processes, ii, Information and computation 100 (1) (1992) 41–77.

[4] K. Prasad, A calculus of broadcasting systems, in: TAPSOFT'91, Springer, 1991, pp. 338–358.

[5] J. Bengtson, M. Johansson, J. Parrow, B. Victor, Psi-calculi: a framework for mobile processes with nominal data and logic, Logical Methods in Computer Science 7 (1). doi:10.2168/LMCS-7(1:11)2011.
URL https://doi.org/10.2168/LMCS-7(1:11)2011

[6] L. Wischik, P. Gardner, Explicit fusions, Theor. Comput. Sci. 340 (3) (2005) 606–630. doi:10.1016/j.tcs.2005.03.017.
URL https://doi.org/10.1016/j.tcs.2005.03.017

[7] V. A. Saraswat, M. C. Rinard, Concurrent constraint programming, in: Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages, San Francisco, California, USA, January 1990, 1990, pp. 232–245. doi:10.1145/96709.96733.
URL https://doi.org/10.1145/96709.96733

[8] D. R. Gilbert, C. Palamidessi, Concurrent constraint programming with process mobility, in: Computational Logic - CL 2000, First International Conference, London, UK, 24-28 July, 2000, Proceedings, 2000, pp. 463–477. doi:10.1007/3-540-44957-4_31.
URL https://doi.org/10.1007/3-540-44957-4_31

[9] M. G. Buscemi, U. Montanari, Cc-pi: A constraint-based language for specifying service level agreements, in: Programming Languages and Systems, 16th European Symposium on Programming, ESOP 2007, Held as Part of the Joint European Conferences on Theory and Practics of Software, ETAPS 2007, Braga, Portugal, March 24 - April 1, 2007, Proceedings, 2007, pp. 18–32. doi:10.1007/978-3-540-71316-6_3.
URL https://doi.org/10.1007/978-3-540-71316-6_3

[10] G. Agha, Actors: A Model of Concurrent Computation in Distributed Systems, MIT Press, Cambridge, MA, USA, 1986.

[11] C. Ene, T. Muntean, A broadcast-based calculus for communicating systems, in: Parallel and Distributed Processing Symposium, International, Vol. 3, IEEE Computer Society, 2001, pp. 30149b–30149b.

[12] Y. A. Alrahman, R. De Nicola, M. Loreti, On the power of attribute-based communication, in: Formal Techniques for Distributed Objects, Components, and Systems - 36th IFIP International Conference, FORTE, Springer, 2016, pp. 1–18, full technical report can be found on http://arxiv.org/abs/1602.05635. doi:10.1007/978-3-319-39570-8_1.

[13] Y. A. Alrahman, R. De Nicola, M. Loreti, Programming the Interactions of Collective Adaptive Systems by Relying on Attribute-based Communication, ArXiv e-printsarXiv:1711.06092.

[14] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, R. Peralta, Computation in networks of passively mobile finite-state sensors, Distributed Computing 18 (4) (2006) 235–253. doi:10.1007/s00446-005-0138-3.
URL https://doi.org/10.1007/s00446-005-0138-3

[15] M. Boreale, R. De Nicola, R. Pugliese, Basic observables for processes, Inf. Comput. 149 (1) (1999) 77–98.

[16] R. Milner, D. Sangiorgi, Barbed bisimulation, in: Automata, Languages and Programming, Springer, 1992, pp. 685–695.

[17] K. Honda, N. Yoshida, On reduction-based process semantics, Theoretical Computer Science 151 (2) (1995) 437–486.

[18] D. Sangiorgi, D. Walker, The pi-calculus: a Theory of Mobile Processes, Cambridge university press, 2003.

[19] R. Milner, Communication and concurrency, Prentice-Hall, Inc., 1989.

[20] R. Milner, A calculus of communicating systems, Springer, 1980. doi:10.1007/3-540-10235-3.
URL https://doi.org/10.1007/3-540-10235-3

[21] J. Borgström, S. Huang, M. Johansson, P. Raabjerg, B. Victor, J. Å. Pohjola, J. Parrow, Broadcast psi-calculi with an application to wireless protocols, Software and System Modeling 14 (1) (2015) 201–216. doi:10.1007/s10270-013-0375-z.
URL https://doi.org/10.1007/s10270-013-0375-z

[22] G. Agha, C. J. Callsen, ActorSpace: an open distributed programming paradigm, Vol. 28, ACM, 1993.

[23] G. V. Chockler, I. Keidar, R. Vitenberg, Group communication specifications: a comprehensive study, ACM Computing (CSUR) 33 (4) (2001) 427–469.

[24] H. W. Holbrook, D. R. Cheriton, Ip multicast channels: Express support for large-scale single-source applications, in: ACM SIGCOMM Computer Communication Review, Vol. 29, ACM, 1999, pp. 65–78.

[25] M. A. Bass, F. T. Nguyen, Unified publish and subscribe paradigm for local and remote publishing destinations, uS Patent 6,405,266 (Jun. 11 2002).

[26] P. T. Eugster, P. A. Felber, R. Guerraoui, A.-M. Kermarrec, The many faces of publish/subscribe, ACM Comput. Surv. 35 (2) (2003) 114–131. doi:10.1145/857076.857078.
URL http://doi.acm.org/10.1145/857076.857078

[27] T. Given-Wilson, Expressiveness via intensionality and concurrency, in: Theoretical Aspects of Computing - ICTAC 2014 - 11th International Colloquium, Bucharest, Romania, September 17-19, 2014. Proceedings, 2014, pp. 206–223. doi:10.1007/978-3-319-10882-7_13.
URL https://doi.org/10.1007/978-3-319-10882-7_13

[28] D. Gorla, Towards a unified approach to encodability and separation results for process calculi, Inf. Comput. 208 (9) (2010) 1031–1053. doi:10.1016/j.ic.2010.05.002.
URL https://doi.org/10.1016/j.ic.2010.05.002

[29] M. Abadi, B. Blanchet, C. Fournet, The applied pi calculus: Mobile values, new names, and secure communication, J. ACM 65 (1) (2018) 1:1–1:41. doi:10.1145/3127586.
URL https://doi.org/10.1145/3127586

[30] Y. A. Alrahman, R. De Nicola, G. Garbi, M. Loreti, A distributed coordination infrastructure for attribute-based interaction, in: Formal Techniques for Distributed Objects, Components, and Systems - 38th IFIP WG 6.1 International Conference, FORTE 2018, Held as Part of the 13th International Federated Conference on Distributed Computing Techniques, DisCoTec 2018, Madrid, Spain, June 18-21, 2018, Proceedings, 2018, pp. 1–20. doi:10.1007/978-3-319-92612-4_1.
URL https://doi.org/10.1007/978-3-319-92612-4_1

[31] K. V. Prasad, A calculus of broadcasting systems, Science of Computer Programming 25 (2) (1995) 285–327.

[32] T. Given-Wilson, D. Gorla, B. Jay, Concurrent pattern calculus, in: Theoretical Computer Science, Springer, 2010, pp. 244–258.

[33] M. John, C. Lhoussaine, J. Niehren, A. M. Uhrmacher, The attributed pi-calculus with priorities, in: Transactions on Computational Systems Biology XII, Springer, 2010, pp. 13–76.

[34] M. John, C. Lhoussaine, J. Niehren, Dynamic compartments in the imperative π-calculus, in: Computational Methods in Systems Biology, Springer, 2009, pp. 235–250.

[35] A. Bruni, S. Mödersheim, F. Nielson, H. R. Nielson, Set-pi: Set membership p-calculus, in: 2015 IEEE 28th Computer Security Foundations Symposium, 2015, pp. 185–198. doi:10.1109/CSF.2015.20.

[36] R. Vigo, F. Nielson, H. Riis Nielson, Broadcast, Denial-of-Service, and Secure Communication, in: 10th International Conference on integrated Formal Methods (iFM'13), Vol. 7940 of LNCS, 2013, pp. 410–427.

[37] K. Prasad, A calculus of value broadcasts, in: PARLE'93 Parallel Architectures and Languages Europe, Springer, 1993, pp. 391–402.

[38] Y. A. Alrahman, G. Perelli, N. Piterman, A computational framework for adaptive systems and its verification, CoRR abs/1906.10793. arXiv:1906.10793.
URL http://arxiv.org/abs/1906.10793