A Behavioural Theory for Interactions in Collective-Adaptive Systems[☆]

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

[a] *IMT School for Advanced Studies, Lucca, Italy*
[b] *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 simple basic observables and context closure. The introduced equivalence is used to study the expressiveness of *AbC* in terms of encoding broadcast channel-based interactions and to establish formal relationships between systems descriptions at different levels of abstraction.

Keywords: Collective-adaptive systems, Attribute-Based Communication, Process calculus, Operational semantics, Behavioral 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 and programming frameworks still handle the interaction between distributed components by relying on their identities; see, e.g., the Actor model [3], or by relying on channel-names as the case with process calculi, e.g., point-to-point [4], multicast with explicit addressing [5], or broadcast [6]. In these formalisms, interactions rely on names or addresses 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. Thus, a change of perspective of how communication can be achieved, while possibly taking into account run-time properties, status, and capabilities of systems, is on demand.

In this article, we study the impact of *Attribute-based Communication*, a novel communication paradigm that permits selecting a group of partners by considering the predicates over the (dynamic) values of the

[☆]This research has been partially supported by the European projects IP 257414 ASCENS and STReP 600708 QUANTICOL, and by the Italian project PRIN 2010LHT4KM CINA.

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

attributes they expose. Communication takes place anonymously in an implicit multicast fashion without a prior agreement between the communicating partners; thanks to anonymity we have that scalability, dynamicity, and open-endedness can be easily achieved. 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.

Sending operations are non-blocking while receiving operations are blocking. This breaks synchronisation dependencies between interacting partners, and permits modelling systems where communicating 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.

Modeling an opportunistic behaviour in channel-based communication, e.g., π-calculus [7], is definitely more challenging. Components should agree on specific names or channels to interact. Channels have no connection with the component attributes, characteristics or knowledge. They are specified as addresses where the exchange should happen. Channel names are static and changing them locally at run-time requires explicit communication and intensive use of scoping mechanisms which do affect programs readability and compositionality.

The attribute-based system is however more than just the parallel composition of interacting partners; it is also parametrised with respect to the environment or the space where system components are executed. The environment has a great impact on how components behave. It introduces a new way of indirect communication, which we refer to as *interdependence*, that leads components to mutually influence each other unintentionally. Unlike classical message passing and IP multicast [5] where the reference address of the group is explicitly included in the message, AbC components are unaware of the existence of each other and they receive messages only if they satisfy the sender's requirements.

Attributes make it easy to encode interesting features of CAS. For instance, awareness can be easily modelled by locally reading the values of the attributes that represent either the component status (e.g., the battery level of a robot) or the external environment (e.g., the external humidity). Also localities of CAS components can be naturally modelled as attributes.

We provide an evidence of the expressive power of AbC by discussing how to use it to encode different communication paradigms and we provide a uniform encoding of a broadcast channel-based process calculus, named $b\pi$-calculus [8], into AbC. We conjecture that the converse is not possible.

An AbC system is rendered as a set of parallel components, each equipped with a set of attributes whose values can be modified by internal actions. The operational semantics of AbC is given in terms of a labeled transition system (LTS) that is also used as the basis for defining a notion of bisimulation-based equivalence for AbC components. We first introduce a context-based reduction barbed congruence by relying on very simple basic observables and then the corresponding extensional labeled bisimilarity. We also show how to use the introduced bisimilarity to prove equivalences of different system properties and to prove correctness of the encoding of $b\pi$-calculus [8], into AbC.

This article is an extended version of the conference paper presented in [9]. Here, we extend our behavioural theory and provide equational laws for it. Moreover, we provide full proofs of all the results introduced there. The scope of this paper is focused on the theoretical aspects of our calculus while aspects concerning programming methodologies can be found in [10]; where 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 the operational semantics of the calculus that is based on two relations, the first relation describes the behaviour of individual components and the other one describes the behaviour of AbC systems. In Section 4 we define a behavioural theory for AbC by introducing a barbed congruence and then an equivalent definition of a labeled bisimulation. Section 5 is used to introduce a number of equational laws.

Section 6 provides an evidence of the expressive power of AbC; there we discuss how the calculus can be used to model other communication paradigms and prove correctness and completeness of an encoding of a message passing calculus into AbC. Finally, in Section 7 we sum up our main contributions, relate our work to closely related literature and list research directions that worth further investigation.

2. Syntax of the *AbC* Calculus

In this section we formally present the syntax of AbC and briefly discuss the intuition behind the different operators we introduce.

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* $\Gamma : \mathcal{A} \rightharpoonup \mathcal{V}$ 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. An *interface* $I \subseteq \mathcal{A}$ consists of a 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 can be composed by using the parallel operator $\|$, e.g., $C_1 \| C_2$ or can be replicated by using the replicating operator !, e.g., $!C$ which can always create a new copy of C. The scope of names say $\tilde{n}$, can be restricted by using the restriction operator $\nu\tilde{n}$. For instance, in a component of the form $C = C_1 \parallel \nu\tilde{n}C_2$, the occurrences of the names $\tilde{n}$ in C_2 are only visible within C_2. The visibility of attribute values can be restricted while the visibility of attribute identifiers is instead never limited. The attribute identifiers are assumed to be known by each component in the system.

(Components)	$C ::= \Gamma :_I P \mid C_1 \| C_2 \mid\ !C \mid \nu\tilde{x}C$
(Processes)	$P ::= \mathsf{0} \mid \Pi(\tilde{x}).U \mid (\tilde{E})@\Pi.U \mid \langle\Pi\rangle P \mid P_1 + P_2 \mid P_1 \vert P_2 \mid K$
(Updates)	$U ::= [a := E]U \mid P$
(Predicates)	$\Pi ::= \mathsf{tt} \mid \mathsf{ff} \mid p(\tilde{E}) \mid \Pi_1 \wedge \Pi_2 \mid \Pi_1 \vee \Pi_2 \mid \neg\Pi$
(Expressions)	$E ::= v \mid x \mid a \mid \mathtt{this}.a \mid op(\tilde{E})$

Table 1: The syntax of the AbC calculus

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 *context aware* $\langle\Pi\rangle P$ process, a *nodeterministic choice* between two processes $P_1 + P_2$, a *parallel composition* of two processes $P_1|P_2$, or a process call with a unique identifier K used in process definition $K \triangleq P$. All of these operators will now be described below. We start by explaining what we mean by expressions and predicates, then we continue by describing the actual operations on processes.

An *expression* E is built from constant values $v \in \mathcal{V}$, variables x, attribute identifiers a, a reference to the value of a ($\mathtt{this}.a$) in the component that is executing the code, or through a standard operators $op(\tilde{E})$[2]. The evaluation of expression E under Γ is denoted by $[\![E]\!]_\Gamma$. The definition of $[\![\cdot]\!]_\Gamma$ is standard, the only interesting cases are $[\![a]\!]_\Gamma = [\![\mathtt{this}.a]\!]_\Gamma = \Gamma(a)$.

A *predicate* Π is built from boolean constants, tt and ff, and from an *atomic predicate* $p(\tilde{E})$ by using standard boolean operators ($\neg$, $\wedge$ and $\vee$). The precise set of atomic predicates is not detailed here; we only assume that it contains basic binary relations like $>$, $<$, $\leq$, $\geq, =$, and the predicates $\in$ and $\notin$. In what follows, we shall use the notation $\{\Pi\}_\Gamma$ to indicate the *closure* of a predicate Π under the attribute environment Γ. The closure is also a predicate Π' obtained from Π by replacing the occurrences of the

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

[2]For the sake of simplicity, we omit the specific syntax of operators used to build expressions and use $\tilde{E}$ to denote sequences of expressions.

expression $\mathtt{this}.a$ with its value $\Gamma(a)$. We also shall use the relation $\simeq$ to denote a semantic equivalence for predicate as defined below.

Definition 2.1 (Predicate Equivalence). *Two predicates are semantically equivalent, written $\Pi_1 \simeq \Pi_2$, iff for every environment Γ, it holds that:*

$$\Gamma \models \Pi_1 \textit{ iff } \Gamma \models \Pi_2$$

This equivalence is decidable because predicates only consider standard boolean expressions and simple constraints on attribute values.

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 a component satisfying predicate Π; the sequence $\tilde{x}$ acts as a placeholder for received values. The constructs νx and $\Pi(\tilde{x})$ act as binders for names, e.g., $\tilde{x}$ in $\Pi(\tilde{x}).U$ and x in $\nu x C$. We will say that a name x is *bound* when it occurs under the scope of an input action or a restriction operator while it is *free* when it is not bound. We use $bn(P)$ and $fn(P)$ to denote the set of bound and free names of P, respectively. We use $fv(P)$ to denote the set of free process variables of P. We use $x,\ y,\ \ldots$ to range over names while $X,\ Y, \ldots$ to range over process variables. Notice that names are used as placeholders for values while process variables are used as placeholders for processes. Our processes are *closed*, i.e. without free process variables ($fv(P) = \emptyset$) because *AbC* components can only exchange values, but not code.

Predicates, used in communication actions, can also refer to names in $\tilde{x}$ and the received values can be used to check whether specific conditions are satisfied. For instance, the action

$$((x = \text{``}\textit{try}\text{''}) \land (\mathtt{this}.\mathsf{id} > \mathsf{id}) \land (\mathtt{this}.\mathsf{round} = z))(x, y, z)$$

can be used to receive a message of the form $(\text{``}\textit{try}\text{''}, c, r)$ where the value received on z is equal to $\mathtt{this}.\mathsf{round}$ and the value of the interface attribute id of the sending component is less than $\mathtt{this}.\mathsf{id}$. Thus, the predicate can be used to check either the received values or the values of the sending component interface. A predicate can also refer to *local* attributes of components. Thus, an action like

$$(\text{``}\textit{try}\text{''}, c, r)@(\mathtt{this}.\mathsf{id} \in \mathsf{N})$$

can be used to send the message $(\text{``}\textit{try}\text{''}, c, r)$ to all components whose attribute N contains $\mathtt{this}.\mathsf{id}$.

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.

The *awareness construct*, $\langle\Pi\rangle P$, is used to trigger new behaviours (i.e., P) when the status of a component is changed (i.e., $\Pi \models \Gamma$). It blocks the execution of P until predicate Π satisfies the attribute environment.

The *parallel operator*, $P|Q$, models the interleaving between co-located processes, i.e., processes residing within the same component.

The *choice operator*, $P + Q$, indicates a nondeterministic choice between P and Q.

Other process operators can be defined as *macros* in *AbC*. Indeed, we will use the following derived operators:

$$\textbf{if } \Pi \textbf{ then } P_1 \textbf{ else } P_2 \triangleq \langle\Pi\rangle P_1 + \langle\neg\Pi\rangle P_2 \tag{1}$$

$$\textbf{set}(a, E)P \triangleq ()@\mathsf{ff}.[a := E]P \tag{2}$$

$$[a_1 := E_1,\ a_2 := E_2,\ \ldots,\ a_n := E_n]P \triangleq [a_1 := E_1][a_2 := E_2]\ldots[a_n := E_n]P \tag{3}$$

$\Gamma \models \mathtt{tt}$ for all Γ

$\Gamma \not\models \mathtt{ff}$ for all Γ

$\Gamma \models p(\tilde{E})$ iff $[\![\tilde{E}]\!]_\Gamma \in [\![p]\!]_\Gamma$

$\Gamma \models \Pi_1 \wedge \Pi_2$ 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

3. AbC Operational Semantics

In this section, we provide an overview of the operational semantics of *AbC* and use fragments of the Distributed Graph Colouring example to show how the semantics rules work. The operational semantics of *AbC* is based on two relations. The transition relation $\longmapsto$ that describes the behavior of single components and the transition relation $\longrightarrow$ that relies on $\longmapsto$ and describes system behaviours.

3.1. Operational semantics of components

We use the transition relation $\longmapsto\ \subseteq$ Comp $\times$ CLAB $\times$ 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:

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

The λ-labels are used to denote *AbC* output $\nu\tilde{x}\Gamma \triangleright \overline{\Pi}(\tilde{v})$ and input $\Gamma \triangleright \Pi(\tilde{v})$ actions. The former contains the sender's predicate Π, that specifies the expected communication partners, the transmitted values $\tilde{v}$, the portion of the sender *attribute environment* Γ that can be perceived by receivers and a possible set of private names $\tilde{x}$. The latter is just the complementary label selected among all the possible ones that the receiver may accept. An output is called "bound" if its label contains a bound name (i.e., if $\tilde{x} \neq \emptyset$) and we shall use $\Gamma \triangleright \overline{\Pi}(\tilde{v})$ for an output label when $\tilde{x} = \emptyset$. 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* labels is crucial to appropriately handle dynamic operators like choice and awareness.

Free names in α are specified as follows:

- $fn(\nu\tilde{x}\Gamma \triangleright \overline{\Pi}(\tilde{v})) = fn(\Gamma \triangleright \Pi(\tilde{v}))\backslash\tilde{x}$
- $fn(\Gamma \triangleright \Pi(\tilde{v})) = fn(\widetilde{\Gamma \triangleright \Pi(\tilde{v})}) = fn(\Gamma) \cup fn(\Pi) \cup \tilde{v}$

The free names of a predicate is the set of names occurring in that predicate except for attribute identifiers. Notice that $\mathtt{this}.a$ is only a reference to the value of the attribute identifier a. Only the output label has bound names:

- $bn(\nu\tilde{x}\Gamma \triangleright \overline{\Pi}(\tilde{v})) = \tilde{x}$.

The transition relation $\longmapsto$ 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 showing how a component discards undesired input messages.

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:

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

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

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

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

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

Table 3: Operational Semantics of Components (Part 1)

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

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

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

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

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

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

Table 4: Operational Semantics of Components (Part 2)

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

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

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 action; we need another rule (FBrd) to model the fact that all inputs 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 $[\tilde{v}/\tilde{x}]$. The predicate satisfaction relation, $\models$, is reported in Table 2.

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.

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 0 always discards messages.

180 *3.2. Operational semantics of systems*

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

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

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

Rules Comp and FComp depends on relation $\longmapsto$ and 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 perceived 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) governs communication between two parallel components C_1 and C_2: If C_1 sends a message then (C_2 can receive it by applying rule Comp). However, C_2 has also the possibility of discarding the message by applying rule FComp. The ability to receive or discard the message depends on the local behaviour of a component as defined previously.

Rule iRep states that a component replicates itself every time a message is received while rules oRep and fRep ensure that replication is not activated when messages are sent or discarded. Restricting the replication to only blocking input actions is important to avoid generating infinite LTS. Notice that we

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

$$\frac{C_1 \xrightarrow{\Gamma\triangleright\Pi(\tilde{v})} C_1' \quad C_2 \xrightarrow{\Gamma\triangleright\Pi(\tilde{v})} C_2'}{C_1 \parallel C_2 \xrightarrow{\Gamma\triangleright\Pi(\tilde{v})} C_1' \parallel C_2'}\ \textsc{Sync} \qquad \frac{C_1 \xrightarrow{\nu\tilde{x}\Gamma\triangleright\overline{\Pi}(\tilde{v})} C_1' \quad C_2 \xrightarrow{\Gamma\triangleright\Pi(\tilde{v})} C_2' \qquad \tilde{x} \cap fn(C_2) = \emptyset}{C_1 \parallel C_2 \xrightarrow{\nu\tilde{x}\Gamma\triangleright\overline{\Pi}(\tilde{v})} C_1' \parallel C_2'}\ \textsc{ComL}$$

$$\frac{C_1 \xrightarrow{\Gamma\triangleright\Pi(\tilde{v})} C_1' \quad C_2 \xrightarrow{\nu\tilde{x}\Gamma\triangleright\overline{\Pi}(\tilde{v})} C_2' \qquad \tilde{x} \cap fn(C_1) = \emptyset}{C_1 \parallel C_2 \xrightarrow{\nu\tilde{x}\Gamma\triangleright\overline{\Pi}(\tilde{v})} C_1' \parallel C_2'}\ \textsc{ComR} \qquad \frac{\Gamma :_I P \xmapsto{\Gamma\triangleright\Pi(\tilde{v})} \Gamma' :_I P'}{!(\Gamma :_I P) \xrightarrow{\Gamma\triangleright\Pi(\tilde{v})} \Gamma' :_I P' \| !(\Gamma :_I P)}\ \text{i}\textsc{Rep}$$

$$\frac{\Gamma :_I P \xmapsto{\Gamma\triangleright\overline{\Pi}(\tilde{v})} \Gamma' :_I P'}{!(\Gamma :_I P) \xrightarrow{\Gamma\triangleright\overline{\Pi}(\tilde{v})} !(\Gamma' :_I P')}\ \text{o}\textsc{Rep} \qquad \frac{\Gamma :_I P \xmapsto{\widetilde{\Gamma\triangleright\Pi(\tilde{v})}} \Gamma' :_I P'}{!(\Gamma :_I P) \xrightarrow{\Gamma\triangleright\Pi(\tilde{v})} !(\Gamma' :_I P')}\ \text{f}\textsc{Rep}$$

$$\frac{C[y/x] \xrightarrow{\lambda} C' \quad y \notin n(\lambda) \wedge y \notin fn(C)\backslash\{x\}}{\nu x C \xrightarrow{\lambda} \nu y C'}\ \textsc{Res}$$

$$\frac{C \xrightarrow{\nu\tilde{x}\Gamma\triangleright\overline{\Pi}(\tilde{v})} C' \qquad \begin{matrix}(\Pi \blacktriangleright y) \doteq \mathsf{ff} \\ y \in n(\Pi)\end{matrix}}{\nu y C \xrightarrow{\triangleright\overline{\mathsf{ff}}()} \nu y \nu\tilde{x} C'}\ \textsc{Hide1} \qquad \frac{C \xrightarrow{\nu\tilde{x}\Gamma\triangleright\overline{\Pi}(\tilde{v})} C' \qquad \begin{matrix}(\Pi \blacktriangleright y) \not\doteq \mathsf{ff} \\ y \in n(\Pi)\end{matrix}}{\nu y C \xrightarrow{\nu\tilde{x}\Gamma\uparrow y\triangleright\overline{\Pi\blacktriangleright y}(\tilde{v})} \nu y C'}\ \textsc{Hide2}$$

$$\frac{C[y/x] \xrightarrow{\Gamma\triangleright\overline{\Pi}(\tilde{v})} C' \quad \Pi \not\doteq \mathsf{ff} \quad y \in \tilde{v}\backslash n(\Pi) \wedge y \notin fn(C)\backslash\{x\}}{\nu x C \xrightarrow{\nu y\Gamma\triangleright\overline{\Pi}(\tilde{v})} C'}\ \textsc{Open}$$

Table 5: Operational Semantics of Systems

$$\begin{array}{lcl}
\mathsf{tt}\blacktriangleright x & = & \mathsf{tt} \\
\mathsf{ff}\blacktriangleright x & = & \mathsf{ff} \\
(a = m)\blacktriangleright x & = & \begin{cases} \mathsf{ff} & \texttt{if } x = m \\ a = m & \texttt{otherwise} \end{cases} \\
(\Pi_1 \wedge \Pi_2)\blacktriangleright x & = & \Pi_1\blacktriangleright x \ \wedge\ \Pi_2\blacktriangleright x \\
(\Pi_1 \vee \Pi_2)\blacktriangleright x & = & \Pi_1\blacktriangleright x \ \vee\ \Pi_2\blacktriangleright x \\
(\neg\Pi)\blacktriangleright x & = & \neg(\Pi\blacktriangleright x)
\end{array}$$

Table 6: Predicate restriction $\bullet\blacktriangleright x$

also restrict replication to individual components, but not system components because the latter cannot be controlled.

Rule Res states that component $\nu x C$ with a restricted name x can still perform an action with a λ-label as long as x does not occur in the names of the label and component C can perform the same action. If necessary, we allow renaming with conditions to avoid name clashing.

Rules Hide1 and Hide2 are unique to AbC and introduce a new concept that we call predicate restriction "$\bullet\blacktriangleright x$" as reported in Table 6. In process calculi where broadcasting is the basic primitive for communication like CSP [11] and $b\pi$-calculus [7], broadcasting on a private channel is equal to performing an internal action and no other processes can observe the broadcast except for the one that performed it.

For example in $b\pi$-calculus, if we let $P = \nu a(P_1\| \ P_2)\| \ P_3$ where $P_1 = \bar{a}v.Q,\ P_2 = a(x).R$, and $P_3 = b(x)$ then if P_1 broadcasts on a we would have that only P_2 can observe it since P_2 is within the scope of the restriction. P_3 and other processes only observe an internal action, so $P \xrightarrow{\tau} \nu a(Q\|R[v/x])\| \ b(x)$.

This idea is generalised in AbC to what we call predicate restriction "$\bullet\blacktriangleright x$" in the sense that we either hide a part or the whole predicate using the predicate restriction operator "$\bullet\blacktriangleright x$" where x is a restricted name and the "$\bullet$" is replaced with a predicate. If the predicate restriction operator returns ff then we get the usual hiding operator like in CSP and $b\pi$-calculus because the resulting label does not expose the message according to Hide1 rule (i.e., sending with a false predicate). It makes sense to consider any send action on a false predicate (i.e., $()@\mathsf{ff}$) as a τ or silent action. In what follows, we consider any occurrence of $()@\mathsf{ff}$ as a silent action.

If the predicate restriction operator returns something different from ff then the message is exposed with possibly a smaller predicate and the restricted name remains private. Note that any private name in the message values (i.e., $\tilde{x}$) remains private if $(\Pi \blacktriangleright y) \simeq \mathsf{ff}$ as in rule Hide1 otherwise it is not private anymore as in rule Hide2. In other words, messages are sent on a channel that is partially exposed.

It should be noted that also the exposed portion of the attribute environment is affected by predicate restriction since the restricted value might also occur in the exposed public attribute values. We use the function $\Gamma \uparrow v$, reported below, to exclude the attribute with a restricted value v as shown in rule Hide2.

$$\Gamma \uparrow v = \begin{cases} \Gamma\backslash\{x, v\} & \exists x \in dom(\Gamma) : \Gamma(x) = v \\ \Gamma & \text{otherwise} \end{cases}$$

We would like to stress that the predicate restriction operator, that filters the exposure of the communication predicate either partially or completely, is very useful when modelling user-network interaction. The user observes the network as a single node and interacts with it through a public channel and is not aware of how messages are propagated through the network. Networks propagate messages between their nodes through private channels while exposing messages to users through public channels. For instance, if a network sends a message with the predicate ($keyword = \texttt{this}.topic\ \vee\ capability = fwd$) where the name "$fwd$" is restricted then the message is exposed to the user at every node with forwarding capability in the network with this predicate ($keyword = \texttt{this}.topic$). Network nodes observe the whole predicate but they receive the message only because they satisfy the other part of the predicate (i.e., ($capability = fwd$)). In the following Lemma, we prove that the satisfaction of a restricted predicate $\Pi\blacktriangleright x$ by an attribute environment Γ does not depend on the name x that is occurring in Γ.

Lemma 3.1. $\Gamma \models \Pi\blacktriangleright x$ *iff* $\forall v.\ \Gamma[v/x] \models \Pi\blacktriangleright x$ *for any environment* Γ*, predicate* Π*, and name* x*.*

Proof. For the "if" implication, the proof is carried out by induction on the structure of Π and the "only if" implication is straightforward. Let us assume that $\Gamma \models \Pi\blacktriangleright x$:

- if ($\Pi = \mathsf{tt}$): according to Table 6, ($\mathsf{tt}\blacktriangleright x = \mathsf{tt}$) which means that the satisfaction of tt does not depend on x (i.e., $\Gamma \models \mathsf{tt}\blacktriangleright x$ iff $\forall v.\ \Gamma[v/x] \models \mathsf{tt}$). From Table 1, we have that tt is satisfied by all Γ, so it is easy to that if $\Gamma \models \mathsf{tt}\blacktriangleright x$ then $\forall v.\ \Gamma[v/x] \models \mathsf{tt}\blacktriangleright x$ as required.
- if ($\Pi = \mathsf{ff}$): according to Table 6, ($\mathsf{ff}\blacktriangleright x = \mathsf{ff}$) which again means that the satisfaction of ff does not depend on x. From Table 1, we have that ff is not satisfied by any Γ, so this case holds vacuously.
- if ($\Pi = (a = m)\blacktriangleright x$): according to Table 6, we have two cases:
  - if ($x = m$) then $\Pi = \mathsf{ff}$ and by induction hypotheses, the case holds vacuously.
  - if ($x \neq m$) then $\Pi = (a = m)$, according to Table 1, we have that $\Gamma \models (a = m)$ iff $\Gamma(a) = m$. Since $x \neq m$, then $\Gamma(a) = m$ holds for any value of x in Γ and we have that if $\Gamma \models (a = m)\blacktriangleright x$ then $\forall v.\ \Gamma[v/x] \models (a = m)\blacktriangleright x$ as required.
- if ($\Pi = \Pi_1 \wedge \Pi_2$): according to Table 6, $(\Pi_1 \wedge \Pi_2)\blacktriangleright x = (\Pi_1\blacktriangleright x \wedge \Pi_2\blacktriangleright x)$. From Table 1, we have that $\Gamma \models (\Pi_1\blacktriangleright x \wedge \Pi_2\blacktriangleright x)$ iff $\Gamma \models \Pi_1\blacktriangleright x$ and $\Gamma \models \Pi_2\blacktriangleright x$. By induction hypotheses, we have that if ($\Gamma \models \Pi_1\blacktriangleright x$ then $\forall v.\ \Gamma[v/x] \models \Pi_1\blacktriangleright x$) and if ($\Gamma \models \Pi_2\blacktriangleright x$ then $\forall v.\ \Gamma[v/x] \models \Pi_2\blacktriangleright x$).
  $\Gamma \models (\Pi_1\blacktriangleright x \wedge \Pi_2\blacktriangleright x)$ iff $\forall v.(\Gamma[v/x] \models \Pi_1\blacktriangleright x \wedge \Gamma[v/x] \models \Pi_2\blacktriangleright x)$ and now we have that if $\Gamma \models (\Pi_1 \wedge \Pi_2)\blacktriangleright x$ then $\forall v.\ \Gamma[v/x] \models (\Pi_1 \wedge \Pi_2)\blacktriangleright x$ as required.
- if ($\Pi = \Pi_1 \vee \Pi_2$): This case if analogous to the previous one.
- if ($\Pi = \neg\Pi$): According to Table 6, $(\neg\Pi)\blacktriangleright x = \neg(\Pi\blacktriangleright x)$. From Table 1, we have that $\Gamma \models \neg(\Pi\blacktriangleright x)$ iff not $\Gamma \models (\Pi\blacktriangleright x)$. By induction hypotheses, we have that if (not $\Gamma \models \Pi\blacktriangleright x$ then $\forall v.$ not $\Gamma[v/x] \models \Pi\blacktriangleright x$) and now we have that if $\Gamma \models \neg(\Pi)\blacktriangleright x$ then $\forall v.\ \Gamma[v/x] \models \neg(\Pi)\blacktriangleright x$ as required.

$\square$

Rule Open states that a component has the ability to communicate a private name to other components. This rule is different from the one in π-calculus in the sense that AbC represents multiparty settings. This implies that the scope of the private name x is not expanded to include a group of other components but rather the scope is dissolved. In other words, when a private name is communicated in AbC then the name is not private anymore. Note that, a component that is sending on a false predicate (i.e., $\Pi \triangleq \mathsf{ff}$) cannot open the scope.

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 labeled bisimulation and provide a number of equational laws for it. We also show how bisimulation can be used to prove non trivial properties of systems.

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. So it makes sense to consider only message transmission as an observation. 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 the application of 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*. In this article we use the term "barb" as synonymous with observable, following the works in [12, 13]. In what follows, we shall use the following notations:

- $C \xrightarrow{\tau} C'$ iff $\exists \tilde{x}, \tilde{v}, \Gamma, \text{and } \Pi$ such that $C \xrightarrow{\nu\tilde{x}\Gamma \triangleright \overline{\Pi}(\tilde{v})} C'$ and $\Pi \bumpeq \mathsf{ff}$.
- $\Rightarrow$ denotes $(\xrightarrow{\tau})^*$.
- $\overset{\lambda}{\Rightarrow}$ denotes $\Rightarrow\xrightarrow{\lambda}\Rightarrow$ if $(\lambda \neq \tau)$.
- $\overset{\hat{\lambda}}{\Rightarrow}$ denotes $\Rightarrow$ if $(\lambda = \tau)$ and $\overset{\lambda}{\Rightarrow}$ otherwise.
- $\rightarrowtriangle$ denotes $\xrightarrow{\lambda}$ where λ is an output or $\lambda = \tau$ and $\rightarrowtriangle^*$ is the reflexive transitive closure of $\rightarrowtriangle$.

Definition 4.1 (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:*

$$\mathcal{C}[\bullet] ::= \quad [\bullet] \quad | \quad [\bullet]\|C \quad | \quad C\|[\bullet] \quad | \quad \nu x[\bullet] \quad | \quad ![\bullet]$$

Definition 4.2 (Barbs and Closures)**.**

Barb *Let $C{\downarrow_\Pi}$ mean that component C can send a message with a predicate Π' (i.e., $C \xrightarrow{\nu\tilde{x}\Gamma \triangleright \overline{\Pi'}(\tilde{v})}$ where $\Pi' \bumpeq \Pi$ and $\Pi' \not\bumpeq \mathsf{ff}$). We write $C \Downarrow_\Pi$ if $C \rightarrowtriangle^* C' \downarrow_\Pi$ for some C'*[3]*.*

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 $(C_1, C_2) \in \mathcal{R}$, $C_1 \rightarrowtriangle C_1'$ implies $C_2 \rightarrowtriangle^* C_2'$ 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. 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 [14]. This definition is a slight variant of Milner and Sangiorgi's barbed congruence [13] and it is also known as open barbed bisimilarity [4].

Definition 4.3 (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.*

[3]From now on, we consider the predicate Π to denote only its meaning, not its syntax. In other words, we consider predicates up to semantic equivalence $\bumpeq$.

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 $\rightarrowtriangle^*$ with $\rightarrowtriangle$.

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

- $C_1 \Downarrow_\Pi$ *iff* $C_2 \Downarrow_\Pi$
- $C_1 \rightarrowtriangle^* C_1'$ *implies* $C_2 \rightarrowtriangle^* \cong C_1'$ *where* $\cong C_1'$ *denotes a component that is weakly bisimilar to* C_1'.

Proof. (We prove each statement separately)

- The proof of first item proceeds by induction on the length of the derivation $\rightarrowtriangle^*_n$ where n is the number of derivations. We only prove the "if" implication; the "only if" one follows in a similar way.
  - Base case, $n = 0$: Since $C_1 \cong C_2$, we have that $C_1 \downarrow_\Pi$ implies $C_2 \Downarrow_\Pi$ as indicated in Definition 4.3 and Definition 4.2 respectively. From Definition 4.2, we have that $C_1 \downarrow_\Pi$ if $C_1 \rightarrowtriangle^*_0 C_1' \downarrow_\Pi$ and $C_1 \equiv C_1'$ for some C_1' and $n = 0$. In other words, $C_1 \Downarrow^0_\Pi$ implies $C_2 \Downarrow_\Pi$ as required.
  - Suppose that $\forall k \leq n$: $C_1 \Downarrow^k_\Pi$ implies $C_2 \Downarrow_\Pi$ where $C_1 \Downarrow^k_\Pi$ denotes $C_1 \rightarrowtriangle^*_k C_1' \downarrow_\Pi$. It is sufficient to prove the claim for $k + 1$. Now, we have that $C_1 \rightarrowtriangle^*_{k+1} C_1' \downarrow_\Pi = C_1 \rightarrowtriangle C_1'' \rightarrowtriangle^*_k C_1' \downarrow_\Pi$ for some C_1''. Since $C_1 \cong C_2$ and from Definition 4.3 and Definition 4.2, we have that $C_1 \rightarrowtriangle C_1''$ implies $C_2 \rightarrowtriangle^* C_2''$ for some C_2'' such that $C_1'' \cong C_2''$. Since $C_1'' \rightarrowtriangle^*_k C_1' \downarrow_\Pi$ and from induction hypothesis, we have that $C_1'' \Downarrow^k_\Pi$ implies $C_2'' \Downarrow_\Pi$. As a result, we have that $C_1 \Downarrow_\Pi$ implies $C_2 \Downarrow_\Pi$ as required.
- Also the proof of second item proceeds by induction on the length of the derivation $\rightarrowtriangle^*_n$ where n is the number of derivations.
  - Base case, $n = 1$: Since $C_1 \cong C_2$ and from Definition 4.3 and Definition 4.2, we have that $C_1 \rightarrowtriangle C_1'$ implies $C_2 \rightarrowtriangle^* C_2'$ for some C_2' such that $C_1' \cong C_2'$. In other words, $C_1 \rightarrowtriangle^*_1 C_1'$ implies $C_2 \rightarrowtriangle^* \cong C_1'$ as required.
  - Suppose that $\forall k \leq n$: $C_1 \rightarrowtriangle^*_k C_1'$ implies $C_2 \rightarrowtriangle^* \cong C_1'$. It is sufficient to prove the claim for $k + 1$. Now, we have that $C_1 \rightarrowtriangle^*_{k+1} C_1' = C_1 \rightarrowtriangle C_1'' \rightarrowtriangle^*_k C_1'$ for some C_1''. Since $C_1 \cong C_2$, we have that $C_1 \rightarrowtriangle C_1''$ implies $C_2 \rightarrowtriangle^* C_2''$ for some C_2'' such that $C_1'' \cong C_2''$. Since $C_1'' \rightarrowtriangle^*_k C_1'$ and by induction hypothesis, we have that $C_1'' \rightarrowtriangle^*_k C_1'$ implies $C_2'' \rightarrowtriangle^* \cong C_1'$.
    As a result, we have that $C_1 \rightarrowtriangle^* C_1'$ implies $C_2 \rightarrowtriangle^* \cong C_1'$ as required.

□

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.

Definition 4.4 (Weak Bisimulation). *A symmetric binary relation $\mathcal{R}$ over the set of* AbC*-components is a weak bisimulation if for every action λ, whenever $(C_1, C_2) \in \mathcal{R}$ and λ is of the form τ, $\Gamma \triangleright \Pi(\tilde{v})$, or $(\nu\tilde{x}\Gamma \triangleright \overline{\Pi}(\tilde{v})$ with $\Pi \not\simeq \mathsf{ff}$), it holds that:*

$$C_1 \xrightarrow{\lambda} C_1' \text{ implies } C_2 \overset{\hat{\lambda}}{\Rightarrow} C_2' \text{ and } (C_1', C_2') \in \mathcal{R}$$

where every predicate Π occurring in λ is matched by its semantics meaning in $\hat{\lambda}$. 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 $\overset{\hat{\lambda}}{\Rightarrow}$ is replaced by $\xrightarrow{\lambda}$. It is easy to prove that $\sim$ and $\approx$ are equivalence relations by relying on the classical arguments of [15]. However, our bisimilarities enjoy a much more interesting property: closure under any external context.

The following Lemma is used to prove that a component with a restricted name does not need any renaming when performing a τ action. This lemma will be used to prove Lemma 4.4 in the following.

Lemma 4.2. *$C[y/x] \Rightarrow C'$ implies $\nu xC \Rightarrow \nu yC'$ such that $y \notin fn(C)\backslash\{x\}$.*

Proof. The proof proceeds by induction on the length of the derivation $\Rightarrow_n$ where n is the number of derivations.

- Base Case, $n = 0$:
  $C[y/x] \equiv_\alpha C'$ which implies that $\nu xC \equiv_\alpha \nu yC[y/x]$ where $\equiv_\alpha$ is the structural congruence under α-conversion.

- Suppose that $\forall k \leq n$: $\quad C[y/x] \Rightarrow_k C'$ implies $\nu xC \Rightarrow_k \nu yC'$
  if $C[y/x] \Rightarrow_{k\,+\,1} C'$, then we have that $C[y/x] \Rightarrow_k C'' \xrightarrow{\tau} C'$ for some C''. By induction hypothesis we have that $\nu xC \Rightarrow_k \nu yC''$ and $C'' \xrightarrow{\tau} C'$ which means that $\nu yC'' \xrightarrow{\tau} \nu yC'$.

  In other words, $C'' \xrightarrow{\tau} C'$ implies $C''[y/y] \xrightarrow{\tau} C'$. Now we can apply Res rule. Since $y \notin fn(C'')\backslash\{y\}$ and $y \notin n(\tau)$, we have that $\nu yC'' \xrightarrow{\tau} \nu yC'$ and we have that $\nu xC \Rightarrow \nu yC'$ as required.

□

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

Lemma 4.3 ($\approx$ 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 4.4 ($\approx$ is preserved by name restriction)**.** *If C_1 and C_2 are two components, we have that $C_1 \approx C_2$ implies $\nu xC_1 \;\approx\; \nu xC_2$ for all names x.*

Lemma 4.5 ($\approx$ are preserved by replication)**.** *If $\Gamma :_I P$ and $\Gamma' :_{I'} Q$ are two components we have that $\Gamma :_I P \approx \Gamma' :_{I'} Q$ implies $!(\Gamma :_I P) \;\approx\; !(\Gamma' :_{I'} Q)$.*

As an immediate consequence of Lemma 4.3, Lemma 4.4, and Lemma 4.5, 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 $C_1 \xrightarrow{\nu\tilde{x}\Gamma\triangleright\overline{\Pi}(\tilde{v})}$ for an output label $\nu\tilde{x}\Gamma\triangleright\overline{\Pi}(\tilde{v})$ with $\Pi \not\simeq \mathsf{ff}$. As $(C_1 \approx C_2)$, we have that also $C_2 \xRightarrow{\nu\tilde{x}\Gamma\triangleright\overline{\Pi}(\tilde{v})}$ and $C_2 \Downarrow_\Pi$.

- (Reduction-closure): $C_1 \mathrel{-\!\triangleright} C_1'$ means that either $C_1 \xrightarrow{\tau} C_1'$ or $C_1 \xrightarrow{\nu\tilde{x}\Gamma\triangleright\overline{\Pi}(\tilde{v})} C_1'$. As $(C_1 \approx C_2)$, then there exists C_2' such that either $C_2 \Rightarrow C_2'$ or $C_2 \xRightarrow{\nu\tilde{x}\Gamma\triangleright\overline{\Pi}(\tilde{v})} C_2'$ with $(C_1' \approx C_2')$. So $C_2 \mathrel{-\!\triangleright}^* C_2'$.

- (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 4.3, Lemma 4.4, and Lemma 4.5, we have that $\mathcal{C}[C_1] \approx \mathcal{C}[C_2]$.

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

Finally, we prove that bisimilarity is more than a proof technique, it rather represents a complete characterisation of the reduction barbed congruence.

Lemma 4.6 (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 $\mathcal{R} = \{(C_1, C_2) \mid C_1 \cong C_2\}$ is a weak bisimulation.

1. Suppose that $C_1 \xrightarrow{\nu\tilde{x}\Gamma\triangleright\overline{\Pi}(\tilde{v})} C_1'$ for some fresh names $\tilde{x}$, exposed environment Γ, sending predicate Π and a sequence of values $\tilde{v}$ where $\Pi \not\simeq \mathsf{ff}$. We build up a context to mimic the effect of this transition. Our context has the following form:

   $$\mathcal{C}[\bullet] \triangleq [\bullet] \parallel \prod_{i\in I}(\Gamma_i :_{int_i} \mathsf{tt}(\tilde{x}_i).\langle \tilde{x}_i = \tilde{v}\rangle(\tilde{x}_i,\ a_i)@(in = a_i).0 \parallel \Gamma_i' :_{int_i'} (y = a_i)(\tilde{x}_i,\ y).(\tilde{x}_i,\ b_i)@(out = b_i).0)$$

   where $|\tilde{x_i}| = |\tilde{v}|$ for $i \in I$ and $\Gamma_i' \downarrow int_i' \models (in = a_i)$, and the names a_i and b_i for $i \in I$ are fresh. We use the notation $\langle \tilde{x}_i = \tilde{v}\rangle$ to denote $\langle (x_{i,1} = v_1) \wedge (x_{i,2} = v_2) \wedge \cdots \wedge (x_{i,n} = v_n)\rangle$ where $n = |\tilde{x_i}|$ and $\prod_{i\in I} C_i$ to denote the parallel composition of all components C_i, for $i \in I$. To be able to mimic the effects of the transition $C_1 \xrightarrow{\nu\tilde{x}\Gamma\triangleright\overline{\Pi}(\tilde{v})} C_1'$ by the above context, we need to assume that $\forall i \in I :\ \Gamma_i \models \Pi$. Intuitively, the existence of a barb on $(in = a_i)$ indicates that the action has not yet happened, whereas the presence of a barb on $(out = b_i)$ together with the absence of the barb on $(in = a_i)$ ensures that the action happened.

   As $\cong$ is context-closed, $C_1 \cong C_2$ implies $\mathcal{C}[C_1] \cong \mathcal{C}[C_2]$. Since $C_1 \xrightarrow{\nu\tilde{x}\Gamma\triangleright\overline{\Pi}(\tilde{v})} C_1'$, it follows that:

   $$\mathcal{C}[C_1] \ \rightarrowtriangle^* \ C_1' \parallel \prod_{i\in I}(\Gamma_i :_{int_i} 0 \parallel \Gamma_i' :_{int_i'} (\tilde{v},\ b_i)@(out = b_i).0) \ = \ \hat{C}_1$$

   with $\hat{C}_1 \not\Downarrow_{(in=a_i)}$ and $\hat{C}_1 \Downarrow_{(out=b_i)}$.
   The reduction sequence above must be matched by a corresponding reduction sequence $\mathcal{C}[C_2] \ \rightarrowtriangle^* \hat{C}_2 \cong \hat{C}_1$ with $\hat{C}_2 \not\Downarrow_{(in=a_i)}$ and $\hat{C}_2 \Downarrow_{(out=b_i)}$. By Lemma 4.1 and the conditions on the barbs, we get the structure of the above reduction sequence as follows:

   $$\mathcal{C}[C_1] \ \rightarrowtriangle^* \ C_2' \parallel \prod_{i\in I}(\Gamma_i :_{int_i} 0 \parallel \Gamma_i' :_{int_i'} (\tilde{v},\ b_i)@(out = b_i).0) \ \cong \ \hat{C}_1$$

   This implies that $C_2 \xRightarrow{\nu\tilde{x}\Gamma\triangleright\overline{\Pi}(\tilde{v})} C_2'$. Reduction barbed congruence is preserved by name restriction because it is context closed. We have that $\nu\tilde{a}\nu\tilde{b}\hat{C}_1 \cong \nu\tilde{a}\nu\tilde{b}\hat{C}_2$. By using labeled bisimilarity, Theorem 4.1, the third item of Lemma 5.2 (to be proved in the next section), and because the names a_i and b_i for $i \in I$ are fresh, we have that

   $$\nu\tilde{a}\nu\tilde{b}\hat{C}_1 \cong C_1' \parallel \nu\tilde{a}\nu\tilde{b}(\prod_{i\in I}(\Gamma_i :_{int_i} 0 \parallel \Gamma_i' :_{int_i'} (\tilde{v},\ b_i)@(out = b_i).0))$$

   $$\nu\tilde{a}\nu\tilde{b}\hat{C}_2 \cong C_2' \parallel \nu\tilde{a}\nu\tilde{b}(\prod_{i\in I}(\Gamma_i :_{int_i} 0 \parallel \Gamma_i' :_{int_i'} (\tilde{v},\ b_i)@(out = b_i).0))$$

   By using labeled bisimilarity and Theorem 4.1, we can prove that

   $$\nu\tilde{a}\nu\tilde{b}(\prod_{i\in I}(\Gamma_i :_{int_i} 0 \parallel \Gamma_i' :_{int_i'} (\tilde{v},\ b_i)@(out = b_i).0)) \cong 0$$

   and we have that $C_1' \cong C_2'$ as required.

2. Suppose that $C_1 \xrightarrow{\Gamma\triangleright\Pi(\tilde{v})} C_1'$ for some Γ, Π and a sequence of values $\tilde{v}$. Remember that input actions cannot be observed externally, but a context sending the message $\Gamma \triangleright \overline{\Pi}(\tilde{v})$ associated with this input action could. We build up the following context to mimic the effect of this transition.

$$\mathcal{C}[\bullet] \triangleq [\bullet] \parallel \Gamma' :_{int} (\tilde{v})@\Pi.(\tilde{v})@(in = a).(\tilde{v})@(out = b).0$$

where $\Gamma' \downarrow int = \Gamma$ and the names a and b are fresh. As $\cong$ is context-closed, $C_1 \cong C_2$ implies $\mathcal{C}[C_1] \cong \mathcal{C}[C_2]$. Since $C_1 \xrightarrow{\Gamma\triangleright\Pi(\tilde{v})} C_1'$, it follows that:

$$\mathcal{C}[C_1] \ \rightharpoondown^* \ C_1' \parallel \Gamma' :_{int} (\tilde{v})@(out = b).0 \ = \ \hat{C}_1$$

with $\hat{C}_1 \not\Downarrow_{(in=a)}$ and $\hat{C}_1 \Downarrow_{(out=b)}$.
The reduction sequence above must be matched by a corresponding reduction sequence $\mathcal{C}[C_2] \ \rightharpoondown^* \hat{C}_2 \cong \hat{C}_1$ with $\hat{C}_2 \not\Downarrow_{(in=a)}$ and $\hat{C}_2 \Downarrow_{(out=b)}$. By Lemma 4.1, we have that:

$$\mathcal{C}[C_2] \ \rightharpoondown^* \ C_2' \parallel \Gamma' :_{int} (\tilde{v})@(out = b) \cong \ \hat{C}_1$$

This implies that $C_2 \xRightarrow{\Gamma\triangleright\Pi(\tilde{v})} C_2'$. However, this does not ensure that C_2 has really received the message. We can only conclude that C_2 has either received or discarded the message. Reduction barbed congruence is preserved by name restriction, so we have that $\nu a \nu b \hat{C}_1 \cong \nu a \nu b \hat{C}_2$ and from which we can easily conclude $C_1' \cong C_2'$ as required.

3. Suppose that $C_1 \xrightarrow{\tau} C_1'$. This case is straightforward.

□

Theorem 4.2 (Characterisation). *Bisimilarity and reduction barbed congruence coincide.*

Proof. As a direct consequence of Theorem 4.1 and Lemma 4.6, 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), presented in Definition 4.1. 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. *Let* $\Gamma :_I P \approx \Gamma :_I Q$*, then*

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

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 $(\mathtt{tt})(w)$.
3. The statement, $\Gamma :_I P|R \not\approx \Gamma :_I Q|R$ for some process R, holds easily 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 sequence of operational rules for the awareness operator.
5. The last statement holds easily with 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 [4]. 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 bisimilarity. The reason is related to the fact that input transitions cannot be observed. Below we explain the issue with a concrete example.

Remark 5.2. $\Gamma :_I P \approx \Gamma :_I Q$ *does not imply* $\Gamma :_I P + R \approx \Gamma :_I Q + R$ *for every process* R

Proof. Let $C_1 = \Gamma :_I \Pi_1(x).0$, $C_2 = \Gamma :_I \Pi_2(x).0$, and $R = (v)@\Pi.0$. 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 $\Gamma :_I 0$ while component C_2 applies rule FComp and stays unchanged. Both transitions carry the same label and again $\Gamma :_I 0$ and $\Gamma :_I \Pi_2(x).0$ are equivalent for a similar reason. An external observer cannot distinguish them.

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

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

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

$$\Gamma :_I \Pi_1(x).0 + R \xrightarrow{\Gamma'\triangleright\Pi_3(w)} \Gamma :_I 0 \not\xrightarrow{\Gamma\downarrow I\triangleright\overline{\Pi}(v)}$$

while

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

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. In practice this is not a problem since mixed choice (i.e., nondeterministic choice between input and output actions) is very hard to be implemented. □

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

Lemma 5.1 (Parallel composition).

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

Lemma 5.2 (Name restriction).

- $\nu x C \;\approx C \;\; \textit{if}\; x \notin fn(C)$
- $\nu x \nu y C \approx \nu y \nu x C \quad \textit{if} \quad x \neq y$
- $\nu x C_1 \;\|\; C_2 \approx \nu x(C_1 \;\|\; C_2) \;\textit{if}\; x \notin fn(C_2)$

The proof of the last statement is reported in Appendix A.

Lemma 5.3 (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 5.4 (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 5.5 (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 5.6 (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. □

$$\begin{tikzcd}
\Gamma :_I \Pi(x).P \arrow[rr, dotted, dash, bend left, "\Gamma{\downarrow}I{\models}\Pi' \qquad \Gamma'{\models}\Pi_2"] \arrow[d, bend right, "\Gamma'\triangleright\overline{\Pi'}(v)"'] \arrow[dr, bend left, "\Gamma'\triangleright\overline{\Pi'}(w)"] & & \Gamma :_I \Pi_1(x).P + \Pi_2(x).P \arrow[d, bend right, "\Gamma'\triangleright\overline{\Pi'}(w)"'] \arrow[dr, bend left, "\Gamma'\triangleright\overline{\Pi'}(v)"] & \\
\Gamma :_I P[v/x] & \Gamma :_I P[w/x] \arrow[r, dotted, dash, bend left] & \Gamma :_I P[w/x] & \Gamma :_I P[v/x]
\end{tikzcd}$$

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

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* $\Pi \simeq \Pi_1 \vee \Pi_2$.

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* $\Gamma_1 \downarrow I = \Gamma_2 \downarrow I'$ *and* $[\![E_1]\!]_{\Gamma_2} = [\![E_2]\!]_{\Gamma_1}$.

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 communication 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 name restriction in *AbC*. The next example is very simple, but the intuition behind it will be used later in a more elaborated scenario.

Example 5.3. *Let* $C_1 = \Gamma :_I (v)@\Pi_1.P$ *and* $C_2 = \Gamma :_I (v)@\Pi.P$ *where* $\Pi \simeq \Pi_1 \vee \Pi_2$*, it holds that:*

$$\nu x C_1 \approx \nu x C_2 \quad \textit{if and only if} \quad \Pi_2 \blacktriangleright x = \mathsf{ff}$$

Clearly $\nu x C_1$ can apply rule Res *and evolves to $C_1' = \nu x \Gamma :_I P$ with a transition label $\Gamma \downarrow I \triangleright \overline{\Pi_1}(v)$ while $\nu x C_2$ can apply rule* Hide2 *and evolves $C_2' = \nu x \Gamma :_I P \approx C_1'$ with a transition label $\Gamma \downarrow I \triangleright \overline{\Pi \blacktriangleright x}(v)$. From Table 6, section 3 we have that $(\Pi \blacktriangleright x) = (\mathsf{ff} \vee \Pi_1) \simeq \Pi_1$. Now it is easy to see that components $\nu x C_1$ and $\nu x C_2$ are bisimilar. The hiding mechanism in* AbC *where a predicate can be partially exposed is very useful in describing collective behaviour with a global point of view.*

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

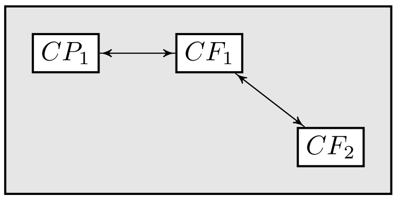

Figure 2: The system with assumptions about the network topology

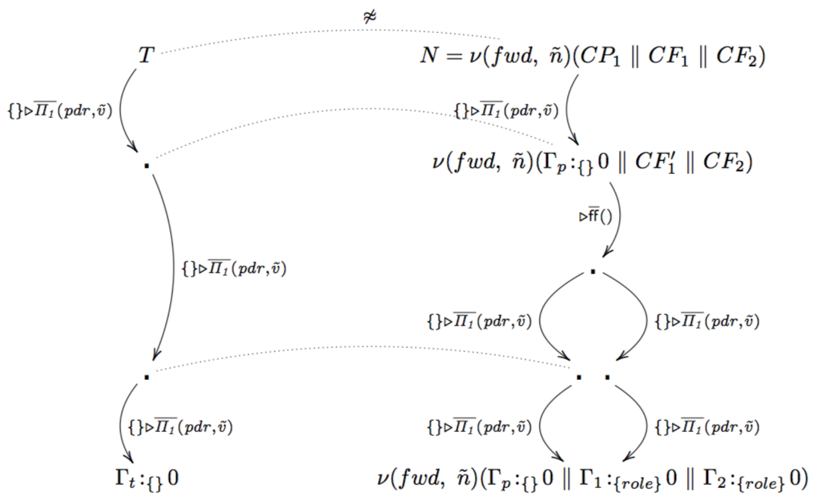
$\not\approx$
T
$N = \nu(fwd,\ \tilde{n})(CP_1 \parallel CF_1 \parallel CF_2)$
$\{\} \triangleright \overline{\Pi_1}(pdr, \tilde{v})$
$\{\} \triangleright \overline{\Pi_1}(pdr, \tilde{v})$
$\nu(fwd,\ \tilde{n})(\Gamma_p :_{\{\}} 0 \parallel CF_1' \parallel CF_2)$
$\triangleright \overline{\mathsf{ff}}()$
$\{\} \triangleright \overline{\Pi_1}(pdr, \tilde{v})$
$\{\} \triangleright \overline{\Pi_1}(pdr, \tilde{v})$
$\{\} \triangleright \overline{\Pi_1}(pdr, \tilde{v})$
$\{\} \triangleright \overline{\Pi_1}(pdr, \tilde{v})$
$\{\} \triangleright \overline{\Pi_1}(pdr, \tilde{v})$
$\{\} \triangleright \overline{\Pi_1}(pdr, \tilde{v})$
$\Gamma_t :_{\{\}} 0$
$\nu(fwd,\ \tilde{n})(\Gamma_p :_{\{\}} 0 \parallel \Gamma_1 :_{\{role\}} 0 \parallel \Gamma_2 :_{\{role\}} 0)$

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

Example 5.4. *We consider two types of components, a provider component $CP = \Gamma_p :_I P$ and a forwarder component $CF = \Gamma_i :_{I'} F$ where the behaviour of processes P and F is defined below.*

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

$$\begin{aligned} F \triangleq\ & (x = pdr)(x,\ \tilde{y}).(\textbf{this}.grp,\ x,\ \tilde{y})@(role = fwd).(x,\ \tilde{y})@\Pi_1.0 \\ & + \\ & (x = \textbf{this}.grp \vee x = \textbf{this}.nbr)(x,\ y,\ \tilde{z}).(y,\ \tilde{z})@\Pi_1.0 \end{aligned}$$

Process P sends an advertisement message to all components that either satisfy predicate Π_1 where $\Pi_1 = (role = client)$ or have a forwarder role (i.e., $(role = fwd)$). Process F may receive an ad from a provider, then it first appends its group id (i.e., $\textbf{this}.grp$) to the message and sends it to nearby forwarders. Process F continues by sending the ad to nearby clients. Alternatively process F may receive a message from one member of its group (i.e., the forwarder that shares the same group id) or from a neighbour forwarder from another group ($x = \textbf{this}.nbr$) and then it will propagate the message to nearby clients. The scenario is simplified to allow at most two hops from the provider. The communication links between providers and forwarders are private (i.e., the name "fwd" and all group and neighbour ids (i.e., $\tilde{n}$) are private names) to avoid interference with other components running in parallel.

The goal of the provider component is to ensure that its advertising message reaches all clients across the network.

To prove if the above specification guarantees this property[4]*, 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 provider*

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

$CP_1 = \Gamma_p :_{\{\}} P$ and two forwarders $CF_1 = \Gamma_1 :_{\{role\}} F$ and $CF_2 = \Gamma_2 :_{\{role\}} F$. Notice that the interface of a provider is empty while the interface for forwarders 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:

$$\Gamma_p = \{(grp, n),\ (role, pdr)\}, \quad \Gamma_1 = \{(grp, n),\ (role, fwd),\ (nbr, n')\}$$
$$\Gamma_2 = \{(grp, n),\ (role, fwd),\ (nbr, n'')\}$$

The full system is represented by the component N as defined below:

$$N = \nu(fwd,\ \tilde{n})(CP_1 \parallel CF_1 \parallel CF_2)$$

The behavior of N without any interventions from other providers is reported on the right side of Figure 6. The provider component CP_1 initiates the interaction by sending an advertisement to nearby clients and forwarders and evolves to $\Gamma_p :_{\{\}} \mathsf{0}$. Forwarder CF_1 receives the message and evolves to CF_1'. The overall system N applies rule Hide2 *and evolves to $\nu(fwd,\ \tilde{n})(\Gamma_p :_{\{\}} \mathsf{0} \parallel CF_1' \parallel CF_2)$ with the label $\{\} \triangleright \overline{(\Pi_1 \lor (role = fwd))\blacktriangleright fwd}(pdr, \tilde{v})$ which is equivalent to $\{\} \triangleright \overline{\Pi_1}(pdr, \tilde{v})$ according to Table 6, section 3. The forwarder CF_1' adds its group id to the message and sends it secretly to nearby forwarders, in our case this is CF_2. The overall system applies rule* Hide1 *and evolves to $\nu(fwd,\ \tilde{n})(\Gamma_p :_{\{\}} \mathsf{0} \parallel CF_1'' \parallel CF_2')$ with the label $\triangleright\overline{\mathsf{ff}}()$. This message is private and is perceived externally as a τ-move. The overall system terminates after emitting the ad, $\{\} \triangleright \overline{\Pi_1}(pdr, \tilde{v})$, two more times, one from CF_1'' and the other from CF_2'. By applying the rule* Res *twice, the system evolves to $\nu(fwd,\ \tilde{n})(\Gamma_p :_{\{\}} \mathsf{0} \parallel \Gamma_1 :_{\{role\}} \mathsf{0} \parallel \Gamma_2 :_{\{role\}} \mathsf{0})$.*

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, defined below, which is able to send the same message three times sequentially and then terminates.

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

Figure 6 shows that system N weakly simulates component T, but they are not bisimilar, i.e., $T \not\approx N$. This is because forwarders are initially prepared to accept messages from any component with a provider role. For instance if we put another provider, say $CP_2 = \Gamma_h :_{\{\}} (\mathbf{this}.role,\ \tilde{w})@(\mathsf{tt}).0$ where $\Gamma_h(role) = pdr$, there is a possibility that CF_1 first receives a message from CP_2 and the system evolves as follows:

$$N \| CP_2 \xrightarrow{\{\}\triangleright\overline{\mathsf{tt}}(pdr,\tilde{w})} \xrightarrow{\{\}\triangleright\overline{\Pi_1}(pdr,\tilde{v})} \xrightarrow{\{\}\triangleright\overline{\Pi_1}(pdr,\tilde{w})} \xrightarrow{\{\}\triangleright\overline{\Pi_1}(pdr,\tilde{w})}$$

while

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

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 shown below:

$$P' \triangleq\ (\mathbf{this}.grp,\ \mathbf{this}.role,\ \tilde{v})@(role = fwd)\ .(\mathbf{this}.role,\ \tilde{v})@\ \Pi_1\ .0$$

$$F' \triangleq\ (x = \mathbf{this}.grp \land y = pdr)(x,\ y,\ \tilde{z}).\ (\mathbf{this}.grp,\ y,\ \tilde{z})@(role = fwd).(y,\ \tilde{z})@\Pi_1.0$$

Now for components $\hat{CP}_1 = \Gamma_p :_{\{\}} P'$, $\hat{CF}_1 = \Gamma_1 :_{\{role\}} F'$, $\hat{CF}_2 = \Gamma_2 :_{\{role\}} F'$, and system $\hat{N}$ where $\hat{N} = \nu(fwd,\ \tilde{n})(\hat{CP}_1 \parallel \hat{CF}_1 \parallel \hat{CF}_2)$, we have that $T \approx \hat{N}$. The interference is avoided by isolating process F' from the external world and now it can only receive messages from its group members with a provider role, in our case this is $\hat{CP}_1$. To allow $\hat{CP}_1$ and $\hat{CF}_1$ to interact, process P' is adapted so that it first sends a secret message to its group and then continues by sending a public message to nearby clients.

6. Encoding channel-based interaction

In this section, we provide evidences of the expressive power of the AbC calculus by discussing how interaction patterns can be naturally expressed in AbC and advocate the use of attribute-based communication as a unifying framework to encompass different communication models. First we hint how *group-based* [16, 17, 5] and *publish/subscribe-based* [18, 19] interaction patterns can be naturally rendered in AbC[5], then we concentrate on the encoding of the broadcast variant of $\pi-$calculus in AbC.

In the group-based model, when an agent wants to send a message, it attaches the group name/id in the message and only member agents 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.

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 modeling of the topic-based publish/subscribe model [19] into AbC can be accomplished by allowing publishers to broadcast messages with "$\mathtt{tt}$" predicates (i.e., satisfied by all subscribers) and only subscribers can check the compatibility of the exposed publishers attributes with their subscriptions.

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

In contrast to more classical process calculi, where senders and receivers have to agree on an explicit channel or name, AbC relies on the satisfaction of predicates over attributes or communicated values for establishing an interaction. Attribute values in AbC can be modified by means of internal actions. Changing attributes values makes it possible to have opportunistic interactions among components in the sense that an attribute update might provide new opportunities of interaction, because the selection of interaction partners depends on predicates over the attributes they expose. Changing the values of these attributes implies changing the set of possible partners and this makes it quite natural to model adaptivity in AbC. Offering this possibility in channel-based process calculi is not easy.

Indeed, we would like to argue that finding an encoding in channel-based process calculi of the following simple AbC system is very difficult if not impossible:

$$\begin{aligned}&\Gamma_1 :_{\{b\}} (msg)@(\mathtt{tt}) \|\\ &\qquad \Gamma_2 :_{I'} (()@\mathsf{ff}.[\mathtt{this}.a := 5]P \mid (b \leq \mathtt{this}.a)(x).Q)\end{aligned}$$

If we assume that initially $\Gamma_1(b) = 3$ and $\Gamma_2(a) = 2$, we have that changing the value of the local attribute a to "5" by the first process in the second component gives it an opportunity of receiving the message "msg" from the process in the first component. One would argue that using restriction to hide local communication and bound input/output actions would be sufficient to encode such kind of behaviors in channel-based process calculi. However, this is not the case because bound input/output actions can engage in communication only when they are instantiated with concrete channel names. In the example above, the input action of the second process of the second component is always enabled. This means that before the update, an input is available on the predicate $y \leq 2$ and after the update it is available on the predicate $y \leq 5$.

6.1. Encoding bπ-calculus into AbC

We consider now, in some details, the issue of encoding one-to-many *channel-based interaction* in the AbC calculus. It may seem tempting to model a channel name as an attribute in AbC, however it turns out not to be the case. The reason is that in channel-based communication, a channel, where the exchange happens, is instantly enabled at the time of interaction and is disabled afterwards. This feature is not present

[5]Further details about encoding these two communication paradigms can be found in [10].

(Component Level)

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

$(\!|\nu\tilde{x}P|\!)_c \triangleq \nu\tilde{x}(\!|P|\!)_c$

(Process Level)

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

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

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

$(\!|(rec\ A\langle\tilde{x}\rangle).G)\langle\tilde{y}\rangle|\!)_p \triangleq (A(\tilde{x}) \triangleq (\!|G|\!)_p)$
$\quad \texttt{where} \quad fn((\!|G|\!)_p) \subseteq \{\tilde{x}\}$

$(\!|G_1 + G_2|\!)_p \triangleq (\!|G_1|\!)_p + (\!|G_2|\!)_p$

Table 7: Encoding bπ-calculus into AbC

in AbC since attributes are persistent in the attribute environment and cannot be disabled at any time (i.e., attribute values are always available to be checked against sender predicates). However, this is not a problem because we can exploit the fact that the receiving predicates in AbC can check the values in the received message. The key idea is to use structured messages to select communication partners where the name of the channel is rendered as the first element in the message; receivers only accept messages with attached channels that match their receiving channels. Actually, attributes do not play any role in such interaction so we assume components with empty environments and interfaces i.e., $\emptyset :_{\emptyset} P$.

To show feasibility of the approach just outlined, we encoded the $b\pi$-calculus [8] into AbC. We consider $b\pi$ a good representative of channel-based process calculi because it uses broadcast instead of binary communication as a basic primitive for interaction which makes it a sort of variant of value-passing CBS [6]. Furthermore, channels in $b\pi$-calculus can be communicated like in the point-to-point π-calculus [7] which is considered as one of the richest paradigms introduced for concurrency so far.

Based on a separation result presented in [20], it has been proven that $b\pi$-calculus and π-calculus are incomparable in the sense that there does not exist any uniform, parallel-preserving translation from $b\pi$-calculus into π-calculus up to any "reasonable" equivalence. On the other hand, in π-calculus a process can non-deterministically choose the communication partner while in $b\pi$-calculus it cannot. Proving the existence of a uniform and parallel-preserving encoding of $b\pi$-calculus into AbC up to some reasonable equivalence ensures at least the same separation results between AbC and π-calculus.

We consider a two-level syntax of $b\pi$-calculus (i.e., we consider only static contexts [15]) as shown below.

$P ::= G \ \mid P_1\|P_2 \ \mid \nu x P$

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

Dealing with the one level $b\pi$-syntax would not add any difficulty concerning channel encoding; only the encoding of parallel composition and name restriction occurring under a prefix or a choice would be slightly more intricate. As reported in Table 7, the encoding of a $b\pi$-calculus 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., $\emptyset :_{\emptyset} \mathsf{G}$. Notice that $(\!|G|\!)_c$ encodes a $b\pi$-sequential process while $(\!|P|\!)_c$ encodes the parallel composition of $b\pi$-sequential processes. The channel is rendered as the first element in the sequence of values. For instance, in the output action $(a,\tilde{x})@(a = a)$, 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.

Notice that the predicate $(a = a)$ is satisfied by any Γ, however including the channel name in the predicate is crucial to encode name restriction correctly.

6.2. Correctness of the encoding

In this section, we provide the correctness proof of the encoding presented in Section 6.1. We start by defining the properties that we would like our encoding to preserve. Basically, when translating a term from $b\pi$-calculus into AbC, we would like 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 the observable behavior (i.e., barbs) and divergence. Moreover, the encoding has to translate output (input) actions of $b\pi$-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 should be complete (i.e., every computation of the source term can be mimicked by its translation) and it should be sound (i.e., every computation of a translated term corresponds to some computation of its source term).

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

Definition 6.2 (Uniform Encoding)**.** *An encoding $\llparenthesis\ \centerdot\ \rrparenthesis : \mathcal{L}_1 \to \mathcal{L}_2$ is uniform if it enjoys the following properties:*

1. *(Homomorphic w.r.t. parallel composition): $\llparenthesis\ P \| Q\ \rrparenthesis \triangleq \llparenthesis\ P\ \rrparenthesis \| \llparenthesis\ Q\ \rrparenthesis$*
2. *(Name invariance): $\llparenthesis\ P\sigma\ \rrparenthesis \ \triangleq\ \llparenthesis\ P\ \rrparenthesis\sigma$, for any permutation of names σ.*
3. *(Faithfulness): $P \Downarrow_1$ iff $\llparenthesis\ P\ \rrparenthesis \Downarrow_2; P \Uparrow_1$ iff $\llparenthesis\ P\ \rrparenthesis \Uparrow_2$*
4. *Operational correspondence*
   1. *(Operational completeness): if $P \rightarrowtriangle_1 P'$ then $\llparenthesis\ P\ \rrparenthesis \rightarrowtriangle^{*}{}_2 \ \simeq_2\ \llparenthesis\ P'\ \rrparenthesis$ where $\simeq$ is the strong barbed equivalence of $\mathcal{L}_2$.*
   2. *(Operational soundness): if $\llparenthesis\ P\ \rrparenthesis \rightarrowtriangle_2 Q$ then there exists a P' such that $P \rightarrowtriangle^{*}{}_1 P'$ and $Q \rightarrowtriangle^{*}{}_2\ \simeq_2\ \llparenthesis\ P'\ \rrparenthesis$, where $\simeq$ is the strong barbed equivalence of $\mathcal{L}_2$.*

Lemma 6.1 (Operational Completeness)**.** *If $P \rightarrowtriangle_{b\pi} P'$ then $\llparenthesis P \rrparenthesis_c \rightarrowtriangle^{*} \simeq \llparenthesis P' \rrparenthesis_c$.*

Now we provide a sketch of the proof of the operational completeness and we report its full details in the Appendix B.

Proof. (Sketch) The proof proceeds by induction on the shortest transition of $\rightarrowtriangle_{b\pi}$. We have several cases depending on the structure of the term P. We only consider the case of parallel composition when communication happens: $P_1 \| P_2 \xrightarrow{\nu\tilde{y}\bar{a}\tilde{z}} P_1' \| P_2'$. By applying induction hypotheses on the premises $P_1 \xrightarrow{\nu\tilde{y}\bar{a}\tilde{z}} P_1'$ and $P_2 \xrightarrow{a(\tilde{z})} P_2'$, we have that $\llparenthesis\ P_1\ \rrparenthesis_c \rightarrowtriangle^{*} \simeq \llparenthesis\ P_1'\ \rrparenthesis_c$ and $\llparenthesis\ P_2\ \rrparenthesis_c \rightarrowtriangle^{*} \simeq \llparenthesis\ P_2'\ \rrparenthesis_c$. We can apply rule ComL.

$$\frac{[\llparenthesis P_1 \rrparenthesis_p] \xrightarrow{\nu\tilde{y}\{\}\triangleright\overline{a=a}(a,\tilde{z})} [\llparenthesis P_1' \rrparenthesis_p] \qquad [\llparenthesis P_2' \rrparenthesis_p] \xrightarrow{\{\}\triangleright a=a(a,\tilde{z})} [\llparenthesis P_2' \rrparenthesis_p]}{[\llparenthesis P_1 \rrparenthesis_p] \ \|\ [\llparenthesis P_2 \rrparenthesis_p] \xrightarrow{\nu\tilde{y}\{\}\triangleright\overline{a=a}(a,\tilde{z})} [\llparenthesis P_1' \rrparenthesis_p] \ \|\ [\llparenthesis P_2' \rrparenthesis_p]}$$

Now, it is easy to see that: $\llparenthesis P_1' \| P_2' \rrparenthesis_c \simeq [\llparenthesis P_1' \rrparenthesis_p] \| \ [\llparenthesis P_2' \rrparenthesis_p]$. Notice that the $b\pi$ term and its encoding have the same observable behavior i.e., $P_1 \| P_2 \downarrow_a$ and $\llparenthesis P_1 \| P_2 \rrparenthesis_c \downarrow_{(a=a)}$. □

Lemma 6.2 (Operational Soundness)**.** *If $\llparenthesis P \rrparenthesis_c \rightarrowtriangle Q$ then $\exists P'$ such that $P \rightarrowtriangle^{*}{}_{b\pi} P'$ and $Q \rightarrowtriangle^{*}\ \simeq\ \llparenthesis P' \rrparenthesis_c$.*

Proof. The proof holds immediately due to the fact that every encoded $b\pi$-term (i.e., $\llparenthesis\ P\ \rrparenthesis_c$) has exactly one possible transition which matches the original $b\pi$-term (i.e., P). □

The idea that we can mimic each transition of $b\pi$-calculus by exactly one transition in AbC implies that soundness and completeness of the operational correspondence can be even proved in a stronger way as in corollary 1 and 2.

Corollary 6.1 (Strong Completeness)**.** *if $P \rightarrowtriangle_{b\pi} P'$ then $\exists Q$ such that $Q \equiv (\!| P' |\!)_c$ and $(\!| P |\!)_c \rightarrowtriangle Q$.*

Corollary 6.2 (Strong Soundness)**.** *if $(\!| P |\!)_c \rightarrowtriangle Q$ then $Q \equiv (\!| P' |\!)_c$ and $P \rightarrowtriangle_{b\pi} P'$*

Theorem 6.1. *The encoding $(\!| \cdot |\!) : b\pi \rightarrow \mathrm{AbC}$ is uniform.*

Proof. Definition 6.2(1) and 6.2(2) hold by construction. Definition 6.2(4) holds by Lemma 6.1, Lemma 6.2, Corollary 6.1, and Corollary 6.2 respectively. Definition 6.2(3) holds easily and as a result of the proof of Lemma 6.1 and the strong formulation of operational correspondence in Corollary 6.1, and Corollary 6.2, this encoding preserves the observable behavior and cannot introduce divergence. $\square$

As a result of Theorem 4.2, Theorem 6.1 and of the strong formulations of Corollary 6.1, and Corollary 6.2, this encoding is sound and complete with respect to bisimilarity as stated in the following corollaries.

Corollary 6.3 (Soundness w.r.t bisimilarity)**.**

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

Corollary 6.4 (Completeness w.r.t bisimilarity)**.**

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

7. Concluding Remarks, Related and Future Works

We have introduced a foundational process calculus, named AbC, for modeling CAS systems by relying on attribute-based communication. We tested the expressive power of AbC by using it as the target of the encoding of $b\pi$-calculus, a rich calculus relying on channel-based communication. We also discussed how other interaction paradigms, such as group based communication and publish-subscribe, could be modelled. We defined behavioral equivalences for AbC and finally we proved the correctness of the encoding of $b\pi$ in AbC up to our equivalence.

Now we would like to briefly discuss related works concerning languages and calculi with primitives that either model multiparty interaction or enjoy specific properties.

AbC is inspired by the SCEL language [21, 22] that was designed to support programming of autonomic computing systems [23]. Compared with SCEL, the knowledge representation in AbC is abstract and is not designed for detailed reasoning during the model evolution. This reflects the different objectives of SCEL and AbC. While SCEL focuses on programming issues, AbC concentrates on a minimal set of primitives to study attribute-based communication.

Many calculi that aim at providing tools for specifying and reasoning about communicating systems have been proposed: CBS [24] 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 input catch the broadcast. In [25], an LTS for CBS was proposed where notions of strong and weak labeled bisimilarity relying on a discard relation were defined.

The $b\pi-$calculus [8] equips $\pi-$calculus [7] 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 labeled bisimilarity which is proved to coincide with the reduction barbed congruence when closed under substitutions. The CPC calculus [26] relies on pattern-matching. Input and output prefixes are generalized to patterns whose unification enables a two-way, or symmetric, flow of information and partners are selected by matching inputs with outputs and testing for equality. The attribute π-calculus [27] aims at constraining interaction by considering values of communication attributes. A λ-function is associated to each receiving action and communication takes place only if the result of the evaluation of the function with the provided input falls within a predefined set of values. The imperative π-calculus [28] is a recent extension of the attribute π-calculus with a global store and with imperative programs used to specify constraints. The broadcast Quality Calculus of [29] 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 mean to change the input contracts during execution.

AbC combines the learnt lessons 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.

We plan to investigate the impact of alternative behavioral relations like testing preorders in terms of equational laws, proof techniques, etc. We want to devise an appropriate notion of temporal logic that can be used to specify, verify, and monitor collective adaptive case studies, modeled in AbC. Actually since CAS components usually operate in an open and changing environment, the spatial and temporal dimensions are strictly correlated and influence each other. So we would like to investigate the impact of spatio-temporal logic approaches in the context of AbC models. One promising approach is presented in [30].

Another line of research is to investigate 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 to establish a static semantics for AbC as a way to discipline the interaction between components. This way we can answer questions regarding deadlock freedom and if the message payload is of the expected type of the receiver.

Appendix A. Detailed Proofs concerning bisimulation properties

Proof of Lemma 4.3. It is sufficient to prove that the relation $\mathcal{R} = \{(C_1 \| C, C_2 \| C) |$ for all C such that $(C_1 \approx C_2)\}$ is a weak bisimulation. Depending on the last rule applied to derive the transition $C_1 \| C \xrightarrow{\lambda} \hat{C}$, we have several cases.

- Assume that $C_1 \| C \xrightarrow{\tau} \hat{C}$: Then the last applied rule is ComL or its symmetrical counterpart ComR. Remember that $\tau = \Gamma \triangleright \overline{\Pi}(v)$ such that $\Pi \simeq \mathsf{ff}$.
  - If ComL is applied then $\hat{C} = C_1' \| C$ and $C_1 \xrightarrow{\tau} C_1'$. Since $C_1 \approx C_2$ then there exists C_2' such that $C_2 \Rightarrow C_2'$ and $(C_1' \approx C_2')$. By applying rule ComL several times, we have that $C_2 \| C \Rightarrow C_2' \| C$ and $(C_1' \| C, C_2' \| C) \in \mathcal{R}$
  - If the symmetrical counterpart of ComR is applied then $\hat{C} = C_1 \| C'$ and $C \xrightarrow{\tau} C'$. So it is immediate to have that $C_2 \| C \Rightarrow C_2 \| C'$ and $(C_1 \| C', C_2 \| C') \in \mathcal{R}$

- Assume that $C_1 \| C \xrightarrow{\nu\tilde{x}\Gamma\triangleright\overline{\Pi}(\tilde{v})} \hat{C}$ with $\hat{x} \cap fn(C) = \emptyset$ and $\Pi \not\simeq \mathsf{ff}$, then the last applied rule is ComL or its symmetrical counterpart ComR.
  - If ComL is applied then $\hat{C} = C_1' \| C'$, $C_1 \xrightarrow{\nu\tilde{x}\Gamma\triangleright\overline{\Pi}(\tilde{v})} C_1'$ and $C \xrightarrow{\Gamma\triangleright\Pi(\tilde{v})} C'$. Since $C_1 \approx C_2$ then there exists C_2' such that $C_2 \xRightarrow{\nu\tilde{x}\Gamma\triangleright\overline{\Pi}(\tilde{v})} C_2'$ and $(C_1' \approx C_2')$. By several applications of rule ComL, we have that $C_2 \| C \xRightarrow{\nu\tilde{x}\Gamma\triangleright\overline{\Pi}(\tilde{v})} C_2' \| C'$ and $(C_1' \| C', C_2' \| C') \in \mathcal{R}$

- If the symmetrical counterpart of ComL is applied then $\hat{C} = C_1' \| C'$, $C_1 \xrightarrow{\Gamma \triangleright \Pi(\tilde{v})} C_1'$ and $C \xrightarrow{\nu\tilde{x}\Gamma \triangleright \overline{\Pi}(\tilde{v})} C'$. So it is immediate to have that $C_2 \| C \xRightarrow{\nu\tilde{x}\Gamma \triangleright \overline{\Pi}(\tilde{v})} C_2' \| C'$ and $(C_1' \| C', C_2' \| C') \in \mathcal{R}$

- $C_1 \| C \xrightarrow{\Gamma \triangleright \Pi(\tilde{v})} \hat{C}$, then the last applied rule is Sync and $\hat{C} = C_1' \| C'$, $C_1 \xrightarrow{\Gamma \triangleright \Pi(\tilde{v})} C_1'$, and $C \xrightarrow{\Gamma \triangleright \Pi(\tilde{v})} C'$. Since $C_1 \approx C_2$ then there exists C_2' such that $C_2 \xRightarrow{\Gamma \triangleright \Pi(\tilde{v})} C_2'$ and $(C_1' \approx C_2')$. By an application of rule Sync and several application of ComL, we have that $C_2 \| C \xRightarrow{\Gamma \triangleright \Pi(\tilde{v})} C_2' \| C'$ and $(C_1' \| C', C_2' \| C') \in \mathcal{R}$.

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

Proof of Lemma 4.4. It is sufficient to prove that the relation $\mathcal{R} = \{(C, B) |\ C = \nu x C_1,\ B = \nu x C_2$ with $(C_1 \approx C_2)\}$ is a weak bisimulation. We have several cases depending on the performed action in deriving the transition $C \xrightarrow{\lambda} \hat{C}$.

- If $(\lambda = \tau)$ then only rule Res is applied. if Res is applied, then $C_1[y/x] \xrightarrow{\tau} C_1'$ and $\hat{C} = \nu y C_1'$. As $(C_1 \approx C_2)$, We have that $C_2[y/x] \Rightarrow C_2'$ with $(C_1' \approx C_2')$. By Lemma 4.2 and several applications of Res, we have that $B \Rightarrow \nu y C_2'$ and $(\nu y C_1', \nu y C_2') \in \mathcal{R}$.

- If $(\lambda = \nu\tilde{y}\Gamma \triangleright \overline{\Pi}(\tilde{v}))$ then either rule Open, Res, Hide1 or Hide2 is applied.

  - If Open is applied, then $x \in (\tilde{v} \cup \tilde{y}) \backslash n(\Pi)$ and $C_1[z/x] \xrightarrow{\Gamma \triangleright \overline{\Pi}(\tilde{v})} C_1'$ with $\hat{C} = C_1'$. As $(C_1 \approx C_2)$, we have that $C_2[z/x] \xRightarrow{\Gamma \triangleright \overline{\Pi}(\tilde{v})} C_2'$ with $(C_1' \approx C_2')$. By Lemma 4.2, an application of Open, and several applications of Res, we have that $B \xRightarrow{\nu\tilde{y}\Gamma \triangleright \overline{\Pi}(\tilde{v})} C_2'$ and $(C_1', C_2') \in \mathcal{R}$.
  - If Res is applied, then $C_1[z/x] \xrightarrow{\nu\tilde{y}\Gamma \triangleright \overline{\Pi}(\tilde{v})} C_1'$ and $\hat{C} = \nu z C_1'$. As $(C_1 \approx C_2)$, we have that $C_2[z/x] \xRightarrow{\nu\tilde{y}\Gamma \triangleright \overline{\Pi}(\tilde{v})} C_2'$ with $(C_1' \approx C_2')$. By Lemma 4.2 and several applications of Res, we have that $B \xRightarrow{\nu\tilde{y}\Gamma \triangleright \overline{\Pi}(\tilde{v})} \nu z C_2'$ and $(\nu z C_1', \nu z C_2') \in \mathcal{R}$
  - If Hide1 is applied, then $C_1 \xrightarrow{\nu\tilde{y}\Gamma \triangleright \overline{\Pi}(\tilde{v})} C_1'$ and $C \xrightarrow{\triangleright \mathsf{ff}()} \hat{C} = \nu x \nu \tilde{y} C_1'$. As $(C_1 \approx C_2)$, we have that $C_2 \xRightarrow{\nu\tilde{y}\Gamma \triangleright \overline{\Pi}(\tilde{v})} C_2'$ with $(C_1' \approx C_2')$. By Lemma 4.2, an application of Hide1, and several applications of Res, we have that $B \xRightarrow{\triangleright \mathsf{ff}()} \nu x \nu \tilde{y} C_2'$ and $(\nu x \nu \tilde{y} C_1', \nu x \nu \tilde{y} C_2') \in \mathcal{R}$
  - If Hide2 is applied, then $C_1 \xrightarrow{\nu\tilde{y}\Gamma \triangleright \overline{\Pi}(\tilde{v})} C_1'$ and $C \xrightarrow{\nu\tilde{y}\Gamma \uparrow x \triangleright \overline{\Pi} \blacktriangleright x(\tilde{v})} \hat{C} = \nu x C_1'$. As $(C_1 \approx C_2)$, we have that $C_2 \xRightarrow{\nu\tilde{y}\Gamma \triangleright \overline{\Pi}(\tilde{v})} C_2'$ with $(C_1' \approx C_2')$. By Lemma 4.2, an application of Hide2, and several applications of Res, we have that $B \xRightarrow{\nu\tilde{y}\Gamma \uparrow x \triangleright \overline{\Pi} \blacktriangleright x(\tilde{v})} \nu x C_2'$ and $(\nu x C_1', \nu x C_2') \in \mathcal{R}$

- If $(\lambda = \Gamma \triangleright \Pi(\tilde{v}))$ then $x \notin n(\gamma)$ and only rule Res is applied. So we have that $C_1[y/x] \xrightarrow{\Gamma \triangleright \Pi(\tilde{v})} C_1'$ and $\hat{C} = \nu y C_1'$. As $(C_1 \approx C_2)$, we have that $C_2[y/x] \xRightarrow{\Gamma \triangleright \Pi(\tilde{v})} C_2'$ with $(C_1' \approx C_2')$. By Lemma 4.2 and several applications of Res, we have that $B \xRightarrow{\Gamma \triangleright \Pi(\tilde{v})} \nu y C_2'$ and $(\nu y C_1', \nu y C_2') \in \mathcal{R}$

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

Proof of Lemma 4.5. Given that $C_1 \approx C_2$, we have that there is a weak bisimulation relation, say $\mathcal{R}$, that relate them where $\mathcal{R} = \{(C_1, C_1) \mid C_1 \approx C_2\}$. The closure of $\mathcal{R}$ under parallel composition is also a weak bisimulation as shown by Lemma 4.3. We will use the notation $\mathcal{R}_{\|}$ to denote the closure of $\mathcal{R}$ under parallel composition. Now it is sufficient to prove that the relation $\mathcal{R}' = \{(!C_1, !C_2)\} \bigcup \{(C' \| !C_1, C'' \| !C_2) |\ (C', C'') \in \mathcal{R}_{\|})\}$ is a weak bisimulation. The proof follows easily by applying rules iRep, oRep, and fRep. The strong case of bisimulation ($\sim$) follows in a similar way. □

Proof of Lemma 5.2. We only prove the last statement and the other statements are straightforward. To prove that $\nu x C_1 \parallel C_2 \approx \nu x(C_1 \parallel C_2)$ if $x \notin fn(C_2)$, it is sufficient to prove that the relation $\mathcal{R} = \{(\nu x C_1 \parallel C_2, \nu x(C_1 \parallel C_2)) \mid x \notin fn(C_2)\}$ is a weak bisimulation. We do a case analysis on the transition $\nu x C_1 \parallel C_2 \xrightarrow{\lambda} \hat{C}$.

We omit the trivial cases when C_2 takes a step.

- Case ($\lambda = \nu x \Gamma \triangleright \overline{\Pi}(\tilde{v})$) where $x \in \tilde{v}$: We can only apply rule ComL and we have that $\nu x C_1 \parallel C_2 \xrightarrow{\nu x \Gamma \triangleright \overline{\Pi}(\tilde{v})} C_1' \parallel C_2' = \hat{C}'$. On the other hand $\nu x(C_1 \parallel C_2)$ evolves to $C_1' \parallel C_2'$ by Lemma 4.2, an application of Open, and several applications of Res and we have that $C_1' \parallel C_2' \approx \hat{C}'$.
- Case ($\lambda = \nu y \Gamma \triangleright \overline{\Pi}(\tilde{v})$) where $x \neq y \wedge x \notin \lambda$: We can only apply rule ComL and we have that $\nu x C_1 \parallel C_2 \xrightarrow{\nu y \Gamma \triangleright \overline{\Pi}(\tilde{v})} \nu z C_1' \parallel C_2'$ if $\nu x C_1 \xrightarrow{\nu y \Gamma \triangleright \overline{\Pi}(\tilde{v})} \nu z C_1'$. On the other hand $\nu x(C_1 \parallel C_2)$ evolves to $\nu z(C_1' \parallel C_2')$ by Lemma 4.2 and several applications of Res.
- Case ($\lambda = \Gamma \triangleright \Pi(\tilde{v})$): With rule Sync, we have that $\nu x C_1 \| C_2 \xrightarrow{\Gamma \triangleright \Pi(\tilde{v})} \nu y C_1' \parallel C_2'$, while $\nu x(C_1 \parallel C_2)$ evolves to $\nu y(C_1' \parallel C_2')$ by Lemma 4.2 and several applications of Res.
- Case ($\lambda = \tau$): $\nu x C_1 \parallel C_2$: We can only apply rule ComL and evolves to $\nu x \nu \tilde{y} C_1' \parallel C_2$ if $\nu x C_1 \xrightarrow{\nu \tilde{y} \overline{\Pi \blacktriangleright x} \tilde{v}} \nu x \nu \tilde{y} C_1'$ where $\Pi \blacktriangleright x \simeq \mathsf{ff}$ or evolves to $\nu y C_1'$ if $\nu x C_1 \xrightarrow{\tau} \nu y C_1'$. On the other hand $\nu x(C_1 \parallel C_2)$ evolves to $\nu x \nu \tilde{y}(C_1' \parallel C_2)$ by Lemma 4.2, an application of Hide1, and several applications of Res, since $C_1 \xRightarrow{\nu \tilde{y} \Gamma \triangleright \overline{\Pi}(\tilde{v})} C_1'$ where $\Pi \blacktriangleright x \simeq \mathsf{ff}$ or it evolves to $\nu y(C_1' \parallel C_2)$ if $\nu x(C_1 \parallel C_2) \xrightarrow{\tau} \nu y(C_1' \parallel C_2)$ by applying rule Res.
- Case ($\lambda = \nu \tilde{y} \Gamma \uparrow x \triangleright \overline{\Pi \blacktriangleright x}(\tilde{v})$) where where $\Pi \blacktriangleright x \not\simeq \mathsf{ff}$ and $x \in n(\Pi)$: $\nu x C_1 \parallel C_2$ can only apply rule ComL and evolves to $\nu x C_1' \parallel C_2'$ if $\nu x C_1 \xrightarrow{\nu \tilde{y} \Gamma \uparrow x \triangleright \overline{\Pi \blacktriangleright x}(\tilde{v})} \nu x C_1'$. On the other hand $\nu x(C_1 \parallel C_2)$ evolves to $\nu x(C_1' \parallel C_2')$ by Lemma 4.2, an application of Hide2, and several applications of Res, since $C_1 \xRightarrow{\nu \tilde{y} \Gamma \triangleright \overline{\Pi}(\tilde{v})} C_1'$. By induction hypotheses we have that $(\nu x C_1 \parallel C_2, \nu x(C_1 \parallel C_2)) \in \mathcal{R}$ as required.

□

Appendix B. Detailed proofs about the encoding

of Lemma 6.1. The proof proceeds by induction on the shortest transition of $\rightarrow_{b\pi}$. We have several cases depending on the structure of the term P.

- 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.

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

Now it is not hard to see that $(\!|\ G\ |\!)_c \simeq [(\!|G|\!)_p]$. They are even structural congruent. 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.

$$\frac{[\Pi(y,\tilde{x}).\llparenthesis G\rrparenthesis_p] \xmapsto{\{\}\triangleright a=a(a,\ \tilde{z})} [\llparenthesis G\rrparenthesis_p[a/y,\ \tilde{z}/\tilde{x}]]}{[\Pi(y,\tilde{x}).\llparenthesis G\rrparenthesis_p] \xrightarrow{\{\}\triangleright a=a(a,\ \tilde{z})} [\llparenthesis G\rrparenthesis_p[a/y,\ \tilde{z}/\tilde{x}]]}$$

It is not hard to see that: $\llparenthesis G[\tilde{z}/\tilde{x}]\rrparenthesis_c \ \simeq\ [\llparenthesis G\rrparenthesis_p[a/y,\ \tilde{z}/\tilde{x}]] \simeq [\llparenthesis G\rrparenthesis_p[\tilde{z}/\tilde{x}]]$ since $y \notin n(\llparenthesis G\rrparenthesis_p])$.

- 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 $P \triangleq \nu xQ$: We have that either $\nu xQ \xrightarrow{\gamma} \nu xQ'$, $\nu xQ \xrightarrow{\tau} \nu x\nu\tilde{y}Q'$ or $\nu xQ \xrightarrow{\nu x\nu\tilde{y}\bar{a}\tilde{z}} Q'$ and it is translated to $\llparenthesis \nu xQ\rrparenthesis_c \triangleq \nu x[\llparenthesis Q\rrparenthesis_p]$. We prove each case independently.
  - Case $\nu xQ \xrightarrow{\gamma} \nu xQ'$ if $Q \xrightarrow{\gamma} Q'$: By applying induction hypotheses on the premise $Q \xrightarrow{\gamma} Q'$, we have that $\llparenthesis Q\rrparenthesis_c \twoheadrightarrow^* \simeq \llparenthesis Q'\rrparenthesis_c$. We can only use rule Res to mimic this transition depending on the performed action.
$$\frac{[\llparenthesis Q\rrparenthesis_p[y/x]] \xrightarrow{\gamma} [\llparenthesis Q'\rrparenthesis_p[y/x]]}{\nu x[\llparenthesis Q\rrparenthesis_p] \xrightarrow{\gamma} \nu y[\llparenthesis Q'\rrparenthesis_p[y/x]]}$$
And we have that $\llparenthesis \nu xQ'\rrparenthesis_c \simeq \nu y[\llparenthesis Q'\rrparenthesis_p[y/x]]$ as required.
  - Case $\nu aQ \xrightarrow{\tau} \nu a\nu\tilde{y}Q'$ if $Q \xrightarrow{\nu\tilde{y}\bar{a}\tilde{z}} Q'$: By applying induction hypotheses on the premise $Q \xrightarrow{\nu\tilde{y}\bar{a}\tilde{z}} Q'$, we have that $\llparenthesis Q\rrparenthesis_c \twoheadrightarrow^* \simeq \llparenthesis Q'\rrparenthesis_c$. We can only use Hide1 to mimic this transition.
$$\frac{[\llparenthesis Q\rrparenthesis_p] \xrightarrow{\nu\tilde{y}\{\}\triangleright\overline{a=a}(a,\ \tilde{z})} [\llparenthesis Q'\rrparenthesis_p]}{\nu a[\llparenthesis Q\rrparenthesis_p] \xrightarrow{\nu\tilde{y}\overline{\mathsf{ff}}(a,\ \tilde{z})} \nu a\nu\tilde{y}[\llparenthesis Q'\rrparenthesis_p]}$$
We have that $\llparenthesis \nu a\nu\tilde{y}Q'\rrparenthesis_c \simeq \nu x\nu\tilde{y}[\llparenthesis Q'\rrparenthesis_p]$ as required.
  - Case $\nu xQ \xrightarrow{\nu x\nu\tilde{y}\bar{a}\tilde{z}} Q'$: follows in a similar way using rule Open .
  - Case $\nu xQ \xrightarrow{\alpha:}$: is similar to the case with Res rule.
- 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 $\llparenthesis G_1 + G_2\rrparenthesis_c \triangleq [\llparenthesis G_1\rrparenthesis_p + \llparenthesis G_2\rrparenthesis_p]$. By applying induction hypotheses on the premise $G_1 \xrightarrow{\alpha} G_1'$, we have that $\llparenthesis\ G_1\ \rrparenthesis_c \twoheadrightarrow^* \simeq \llparenthesis\ G_1'\ \rrparenthesis_c$. 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.
$$\frac{\dfrac{[\llparenthesis G_1\rrparenthesis_p] \xmapsto{\lambda} [\llparenthesis G_1'\rrparenthesis_p]}{[\llparenthesis G_1\rrparenthesis_p + \llparenthesis G_2\rrparenthesis_p] \xmapsto{\lambda} [\llparenthesis G_1'\rrparenthesis_p]}}{[\llparenthesis G_1\rrparenthesis_p + \llparenthesis G_2\rrparenthesis_p] \xrightarrow{\lambda} [\llparenthesis G_1'\rrparenthesis_p]}$$
Again $\llparenthesis G_1'\rrparenthesis_c \simeq [\ \llparenthesis G_1'\rrparenthesis_p]$
- if $P \triangleq P_1\|P_2$: This process is translated to $\llparenthesis\ P_1\|P_2\ \rrparenthesis_c \triangleq [\llparenthesis\ P_1\ \rrparenthesis_p]\|[\llparenthesis\ P_2\ \rrparenthesis_p]$. We have four cases depending on the performed action in deriving the transition $P_1\|P_2 \xrightarrow{\alpha} \hat{P}$.

– $P_1\|P_2 \xrightarrow{\nu\tilde{y}\bar{a}\tilde{x}} P_1'\|P_2'$: We have two cases, either $P_1 \xrightarrow{\nu\tilde{y}\bar{a}\tilde{x}} P_1'$ and $P_2 \xrightarrow{a(\tilde{x})} P_2'$ or $P_2 \xrightarrow{\nu\tilde{y}\bar{a}\tilde{x}} P_2'$ and $P_1 \xrightarrow{a(\tilde{x})} P_1'$. We only consider the first case and the other case follows in the same way. By applying induction hypotheses on the premises $P_1 \xrightarrow{\nu\tilde{y}\bar{a}\tilde{x}} P_1'$ and $P_2 \xrightarrow{a(\tilde{x})} P_2'$, we have that $\llparenthesis P_1\rrparenthesis_c \mathrel{-\!\!\triangleright}^* \simeq \llparenthesis P_1'\rrparenthesis_c$ and $\llparenthesis P_2\rrparenthesis_c \mathrel{-\!\!\triangleright}^* \simeq \llparenthesis P_2'\rrparenthesis_c$. We only can apply ComL.

$$\frac{[\llparenthesis P_1\rrparenthesis_p] \xrightarrow{\nu\tilde{y}\{\}\triangleright\overline{a=a}(a,\ \tilde{x})} [\llparenthesis P_1'\rrparenthesis_p] \quad [\llparenthesis P_2'\rrparenthesis_p] \xrightarrow{\{\}\triangleright a=a(a,\ \tilde{x})} [\llparenthesis P_2'\rrparenthesis_p]}{[\llparenthesis P_1\rrparenthesis_p] \parallel [\llparenthesis P_2\rrparenthesis_p] \xrightarrow{\nu\tilde{y}\{\}\triangleright\overline{a=a}(a,\ \tilde{x})} [\llparenthesis P_1'\rrparenthesis_p] \parallel [\llparenthesis P_2'\rrparenthesis_p]}$$

Again we have that: $\llparenthesis P_1'\|P_2'\rrparenthesis_c \simeq [\llparenthesis P_1'\rrparenthesis_p]\|\ [\llparenthesis P_2'\rrparenthesis_p]$. Notice that the $b\pi$ term and its encoding have the same observable behavior i.e., $P_1\|P_2 \downarrow_a$ and $\llparenthesis P_1\|P_2\rrparenthesis_c \downarrow_{(a=a)}$.

– $P_1\|P_2 \xrightarrow{a(\tilde{x})} P_1'\|P_2'$: By applying induction hypotheses on the premises $P_1 \xrightarrow{a(\tilde{x})} P_1'$ and $P_2 \xrightarrow{a(\tilde{x})} P_2'$, we have that $\llparenthesis P_1\rrparenthesis_c \mathrel{-\!\!\triangleright}^* \simeq \llparenthesis P_1'\rrparenthesis_c$ and $\llparenthesis P_2\rrparenthesis_c \mathrel{-\!\!\triangleright}^* \simeq \llparenthesis P_2'\rrparenthesis_c$. We only can apply Sync to mimic this transition.

$$\frac{[\llparenthesis P_1\rrparenthesis_p] \xrightarrow{\{\}\triangleright a=a(a,\ \tilde{x})} [\llparenthesis P_1'\rrparenthesis_p] \qquad [\llparenthesis P_2'\rrparenthesis_p] \xrightarrow{\{\}\triangleright a=a(a,\ \tilde{x})} [\llparenthesis P_2'\rrparenthesis_p]}{[\llparenthesis P_1\rrparenthesis_p] \parallel [\llparenthesis P_2\rrparenthesis_p] \xrightarrow{\{\}\triangleright a=a(a,\ \tilde{x})} [\llparenthesis P_1'\rrparenthesis_p] \parallel [\llparenthesis P_2'\rrparenthesis_p]}$$

Again we have that: $\llparenthesis P_1'\|P_2'\rrparenthesis_c \simeq [\llparenthesis P_1'\rrparenthesis_p]\|\ [\llparenthesis P_2'\rrparenthesis_p]$.

– $P_1\|P_2 \xrightarrow{\alpha} P_1'\|P_2$ if $P_1 \xrightarrow{\alpha} P_1'$ and $P_2 \xrightarrow{sub(\alpha):}$ or $P_1\|P_2 \xrightarrow{\alpha} P_1\|P_2'$ if $P_2 \xrightarrow{\alpha} P_2'$ and $P_1 \xrightarrow{sub(\alpha):}$. we consider only the first case and by applying induction hypotheses on the premises $P_1 \xrightarrow{\alpha} P_1'$ and $P_2 \xrightarrow{sub(\alpha):}$, we have that $\llparenthesis P_1\rrparenthesis_c \mathrel{-\!\!\triangleright}^* \simeq \llparenthesis P_1'\rrparenthesis_c$ and $\llparenthesis P_2\rrparenthesis_c \mathrel{-\!\!\triangleright}^* \simeq \llparenthesis P_2\rrparenthesis_c$. 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 $P_2 \xrightarrow{sub(\tau):}$. We can apply ComL to mimic this transition.

$$\frac{[\llparenthesis P_1\rrparenthesis_p] \xrightarrow{\{\}\triangleright\overline{\mathsf{ff}}()} [\llparenthesis P_1'\rrparenthesis_p] \quad \dfrac{[\llparenthesis P_2\rrparenthesis_p] \xmapsto{\widetilde{\{\}\triangleright\mathsf{ff}()}} [\llparenthesis P_2\rrparenthesis_p]}{[\llparenthesis P_2\rrparenthesis_p] \xrightarrow{\{\}\triangleright\mathsf{ff}()} [\llparenthesis P_2\rrparenthesis_p]}}{[\llparenthesis P_1\rrparenthesis_p] \parallel [\llparenthesis P_2\rrparenthesis_p] \xrightarrow{\{\}\triangleright\overline{\mathsf{ff}}()} [\llparenthesis P_1'\rrparenthesis_p] \parallel [\llparenthesis P_2\rrparenthesis_p]}$$

and again we have that: $\llparenthesis P_1'\|P_2\rrparenthesis_c \simeq [\llparenthesis P_1'\rrparenthesis_p]\|\ [\llparenthesis P_2\rrparenthesis_p]$.

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

$$\frac{[\llparenthesis P_1\rrparenthesis_p] \xrightarrow{\{\}\triangleright a=a(a,\ \tilde{x})} [\llparenthesis P_1'\rrparenthesis_p] \quad \dfrac{[\llparenthesis P_2\rrparenthesis_p] \xmapsto{\{\}\triangleright \widetilde{a=a(a},\ \tilde{x})} [\llparenthesis P_2\rrparenthesis_p]}{[\llparenthesis P_2\rrparenthesis_p] \xrightarrow{\{\}\triangleright a=a(a,\ \tilde{x})} [\llparenthesis P_2\rrparenthesis_p]}}{[\llparenthesis P_1\rrparenthesis_p] \parallel [\llparenthesis P_2\rrparenthesis_p] \xrightarrow{\{\}\triangleright a=a(a,\ \tilde{x})} [\llparenthesis P_1'\rrparenthesis_p] \parallel [\llparenthesis P_2\rrparenthesis_p]}$$

Again we have that: $\llparenthesis P_1'\|P_2\rrparenthesis_c \simeq [\llparenthesis P_1'\rrparenthesis_p]\|\ [\llparenthesis P_2\rrparenthesis_p]$.

3. if $\alpha = \nu\tilde{y}\bar{a}\tilde{x}$ then $P_1\|P_2 \xrightarrow{\nu\tilde{y}\bar{a}\tilde{x}} P_1'\|P_2$ with $P_1 \xrightarrow{\nu\tilde{y}\bar{a}\tilde{x}} P_1'$ and $P_2 \xrightarrow{a:}$. We can apply ComL.

$$\frac{[\llparenthesis P_1\rrparenthesis_p] \xrightarrow{\nu\tilde{y}\{\}\triangleright\overline{a=a}(a,\ \tilde{x})} [\llparenthesis P_1'\rrparenthesis_p] \quad \dfrac{[\llparenthesis P_2\rrparenthesis_p] \xmapsto{\{\}\triangleright \widetilde{a=a(a},\ \tilde{x})} [\llparenthesis P_2\rrparenthesis_p]}{[\llparenthesis P_2\rrparenthesis_p] \xrightarrow{\{\}\triangleright a=a(a,\ \tilde{x})} [\llparenthesis P_2\rrparenthesis_p]}}{[\llparenthesis P_1\rrparenthesis_p] \parallel [\llparenthesis P_2\rrparenthesis_p] \xrightarrow{\nu\tilde{y}\{\}\triangleright\overline{a=a}(a,\ \tilde{x})} [\llparenthesis P_1'\rrparenthesis_p] \parallel [\llparenthesis P_2\rrparenthesis_p]}$$

Again we have that: $(\!|P'_1\|P_2|\!)_c \simeq [(\!|P'_1|\!)_p]\|\ [(\!|P_2|\!)_p]$. Notice that the $b\pi$ term and its encoding have the same observable behavior i.e., $P_1\|P_2 \downarrow_a$ and $(\!|P_1\|P_2|\!)_c \downarrow_{(a=a)}$.

$\square$

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

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

[5] 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.

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

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

[8] 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.

[9] 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`.

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

[11] C. A. R. Hoare, Communicating sequential processes, Communications of the ACM 21 (8) (1978) 666–677.

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

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

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

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

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

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

[18] 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).

[19] 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`

[20] C. Ene, T. Muntean, Expressiveness of point-to-point versus broadcast communications, in: Fundamentals of Computation Theory, Springer, 1999, pp. 258–268.

[21] R. De Nicola, G. Ferrari, M. Loreti, R. Pugliese, A language-based approach to autonomic computing, in: Formal Methods for Components and Objects, Springer, 2013, pp. 25–48.

[22] R. De Nicola, M. Loreti, R. Pugliese, F. Tiezzi, A formal approach to autonomic systems programming: the scel language, ACM Transactions on Autonomous and Adaptive Systems (2014) 1–29.

[23] J. W. Sanders, G. Smith, Formal ensemble engineering, in: Software-Intensive Systems and New Computing Paradigms, Springer, 2008, pp. 132–138.

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

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

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

[27] 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.

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

[29] 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.

[30] L. Nenzi, L. Bortolussi, Specifying and monitoring properties of stochastic spatio-temporal systems in signal temporal logic, in: 8th International Conference on Performance Evaluation Methodologies and Tools, VALUETOOLS 2014, Bratislava, Slovakia, December 9-11, 2014, Springer, 2014. `doi:10.4108/icst.valuetools.2014.258183`.