Institutions
Markets
Technologies
IMT
INSTITUTE
FOR ADVANCED
STUDIES
LUCCA

#10
2015

**IMT LUCCA CSA TECHNICAL
REPORT SERIES 10
2015**

RA **Computer Science and Applications**

On Expressiveness and Behavioural Theory of Attribute-based Communication

Yehia Abd Alrahman
Rocco De Nicola
Michele Loreti

IMT LUCCA CSA TECHNICAL REPORT SERIES #10/2015

Piazza San Ponziano 6, 55100 Lucca

Research Area
Computer Science and Applications

On Expressiveness and Behavioural Theory of Attribute-based Communication

Yehia Abd Alrahman
IMT Institute for Advanced Studies Lucca

Rocco De Nicola
IMT Institute for Advanced Studies Lucca

Michele Loreti
Università degli Studi di Firenze

On Expressiveness and Behavioural Theory of Attribute-based Communication

Yehia Abd Alrahman and Rocco De Nicola, IMT Institute for Advanced Studies
Michele Loreti, Università degli Studi di Firenze

Attribute-based communication is an interesting alternative to broadcast and binary communication when providing abstract models for the so called Collective Adaptive Systems which consist of a large number of interacting components that dynamically adjust and combine their behavior to achieve specific goals. A basic process calculus, named *AbC*, is introduced whose primary primitive for interaction is *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 interactions among components are dynamically established by taking into account "connections" as determined by predicates over the attributes exposed by components. First, the syntax and the semantics of *AbC* are presented, then expressiveness and effectiveness of the calculus are demonstrated both in terms of the ability to model scenarios featuring collaboration, reconfiguration, and adaptation and of the possibility of encoding a process calculus for *broadcasting channel-based communication* and other communication paradigms. Behavioral equivalences for *AbC* are introduced for establishing formal relationships between different descriptions of the same system.

Categories and Subject Descriptors: D.1.3 [**Programming Techniques**] Distributed programming; F.1.1 [**Theory of Computation**] Models of Computation

General Terms: Languages, Theory

Additional Key Words and Phrases: Attribute-based Communication, Process Calculi, Encoding, Distributed Systems

1. INTRODUCTION

In a world of *Internet of (every)Things* (IoT), of *Systems of Systems* (SoS), and of *Collective Adaptive Systems* (CAS), most of concurrent programming models still rely on communication primitives based on direct (point-to-point), multicast with explicit addressing (i.e. IP multicast [Holbrook and Cheriton 1999]), or broadcast communication. In our view, it is important to consider alternative basic interaction primitives and in this paper we study the impact of a new paradigm that permits selecting groups of partners by considering the (predicates over the) attributes they expose. The findings we report in this paper have been triggered by our interest in CAS, see e.g. [Coronato et al. 2012], and the recent attempts to define appropriate abstractions and linguistic primitives to deal with such systems, see e.g. SCEL [De Nicola et al. 2014] and *AbC* [Abd Alrahman et al. 2015].

CAS consist of large numbers of interacting components which exhibit complex behaviors depending on their attributes, objectives and actions. Decision-making in such systems is complicated and interaction between components may lead to unexpected behaviors. CAS are open, in that components may enter or leave the collective at anytime and might have different (potentially conflicting) objectives; so they need to dynamically adapt to new requirements and contextual conditions. New engineering techniques to address the challenges of developing, integrating, and deploying such systems are needed [Sommerville et al. 2012].

To move towards this goal, in our view, it is important to develop a theoretical foundation for this class of systems that would help in understanding their distinctive features. In this paper, we concentrate our attention on *AbC*, a calculus inspired by SCEL and focusing on a minimal set of primitives that define attribute-based communication. *AbC* systems are represented as sets of parallel components, each is equipped with a set of attributes whose values can be modified by internal actions. Communication actions (both send and receive) are decorated with predicates over attributes that partners have to satisfy to make the

interaction possible. Thus, communication takes place in an implicit multicast fashion, and communication partners are selected by relying on predicates over the attributes exposed in their interfaces. In this way, AbC multicast is different from the usual IP multicast [Holbrook and Cheriton 1999] in the sense that components are unaware of the existence of each other and they receive messages only if they satisfy senders requirements while in IP multicast, the address reference of the group is explicitly included in the message. The semantics for output actions in AbC is non-blocking while input actions are blocking in that they can only take place through synchronization with an available sent message.

Many communication models addressing distributed systems have been introduced so far. Some of the most well-known approaches include: channel-based models (e.g., CCS [Milner 1980], CSP [Hoare 1978], π-calculus [Milner et al. 1992], etc.), group-based models [Agha and Callsen 1993], [Chockler et al. 2001], [Holbrook and Cheriton 1999], and publish/subscribe models [Eugster et al. 2003], [Bass and Nguyen 2002]. The advantage of AbC over channel-based models is that interacting partners are anonymous to each other. They interact by relying on the satisfaction of predicates over the attributes they expose rather than agreeing on channels or names. This makes AbC more suitable for modeling scalable distributed systems as anonymity is a key factor for scalability. Furthermore, the group formation in group-based models like Actorspace [Agha and Callsen 1993] is static in the sense that spaces (i.e., groups) are explicitly specified in advance. while in AbC, groups (i.e., collectives) are dynamically formed and destroyed at the time of interaction. There is no explicit construct for group formation or destruction. On the other hand, the publish/subscribe model is a special case of AbC where publishers send messages tagged with attributes without predicates and only subscribers can check the compatibility of the exposed publishers attributes with their subscriptions.

The point we want to make is that the general concept of attribute-based communication can be used to provide a general unifying framework to encompass such communication models.

Attributes make it easy to encode some interesting aspects; for instance, components localities in CAS can be naturally modeled as attributes, channel-based interaction is simply modeled by exposing a single attribute at the time of interaction, the name of the group is modeled as an attribute, and finally, the publish/subscribe interactions are modeled by having a "tt" predicate (i.e., satisfied by all) at the publisher-side while taking advantage of the possibility for the subscriber to accept only selected inputs.

The version of AbC presented in [Abd Alrahman et al. 2015] is very basic and has a number of limitations, see the discussion in Section 6. In this paper, we majorly reconstruct the calculus, enrich it with behavioral equivalences and assess its expressiveness and effectiveness. More specifically, the main contributions of this paper are:

(1) The introduction of an extended and polyadic version of AbC calculus including name restriction, multithreading, a match operator for acquiring knowledge about both the local status and the external environment, a richer language for defining predicates, and an operator to specify the specific attributes to be exposed during interaction;

(2) the study of the expressive power of the new version of AbC both in terms of the ability of modeling scenarios featuring collaboration, reconfiguration, and adaptation and of the possibility of encoding a process calculus for broadcast channel-based communication ($b\pi$-calculus [Ene and Muntean 2001]) and other communication paradigms;

(3) the definition of behavioral equivalences for AbC by first introducing a context based reduction barbed congruence relation and then the corresponding extensional labelled bisimilarity;

(4) the correctness of the encoding of $b\pi$-calculus [Ene and Muntean 2001] into AbC up to the introduced equivalence.

In the following sections, the main features of the new AbC will be presented in a step-by-step fashion using a running example from the swarm robotics domain described below. A complete AbC model of this scenario

Table I. : The syntax of the AbC calculus and the satisfaction relation $\Gamma \models \Pi$

(Components)	C	$::= \Gamma : P \mid C_1 \| C_2 \mid \nu x C$
(Processes)	P	$::=$
(Inaction)		$\mathsf{0}$
(Input)		$\mid \Pi(\tilde{x}).P$
(Output)		$\mid (\tilde{E})@\Pi \vdash_s .P$
(Update)		$\mid [a := E].P$
(New)		$\mid \texttt{new(x)}P$
(Match)		$\mid \langle \Pi \rangle P$
(Choice)		$\mid P_1 + P_2$
(Parallel)		$\mid P_1 \mid P_2$
(Call)		$\mid K$
(Predicate)	Π	$::= \mathsf{tt} \mid \mathsf{ff} \mid E_1 \bowtie E_2 \mid \Pi_1 \wedge \Pi_2 \mid \ldots$
(Expression)	E	$::= v \mid x \mid a \mid this.a \mid \ldots$

(a)

$\Gamma \models \mathsf{tt}$	for all Γ
$\Gamma \models \mathsf{ff}$	for no Γ
$\Gamma \models a \bowtie v$	iff $\Gamma(a) \bowtie v$
$\Gamma \models \Pi_1 \wedge \Pi_2$	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 $\Gamma \models \Pi$

(b)

is given in Section 4.1.

A Swarm Robotics Scenario We consider a scenario where a swarm of robots spreads throughout a given disaster area. The goal is to locate and rescue possible victims. All robots playing the same role execute the same code. This code defines the functional behavior and a set of adaptation mechanisms regulating the interactions among robots and their environments. All robots initially play the explorer role to search for victims in the environment. Once a robot finds a victim, it changes its role to "rescuer" and communicates victim's information to nearby explorers. The collective (i.e., the swarm) starts forming in preparation for the rescuing procedure. As soon as another robot receives victim's information, it changes its role to "helping" and moves to join the rescuers-collective. The rescuing procedure starts only when the collective formation is complete. During exploration, in case of critical battery level, a robot stops moving and enters the power saving mode until it is recharged. It is worth mentioning that some of the robot attributes are considered as the projection of the robot internal state that is monitored by sensors and actuators.

The rest of the paper is organised as follows: Section 2 and Section 3 formally introduce the syntax and the semantics of AbC calculus respectively. Section 4 provides evidences of the expressive power of AbC both in terms of the application scenario and of encodings of other communication primitives. Section 5 studies the behavioral theory of AbC. Finally, Section 6 discusses and surveys related work, while Section 7 concludes by touching upon directions for future work. Due to space limitations, most proofs have been omitted; they can be found in the accompanying file.

2. THE ABC CALCULUS

The syntax of the AbC calculus is reported in Table I (a). The top-level entities of the calculus are *components* (C), a component consists of either a process P associated with an *attribute environment* Γ, denoted $\Gamma : P$, or a parallel composition $C_1 \| C_2$ of two components. The *attribute environment* $\Gamma : \mathcal{A} \twoheadrightarrow \mathcal{V}$ is a map from attribute identifiers $a \in \mathcal{A}$ to values $v \in \mathcal{V}$, where values can be also names. No restriction is enforced on the attributes of different components, in the sense that different components may have equal or different sets of attributes depending on the system being modeled. It is also possible to restrict the scope of a name say n, by using the name restriction operator νn. For instance, the name n in $C_1 \parallel \nu n C_2$ is only visible within compo-

nent C_2. Basically, this operator plays the same role of *begin* ... *end* block in sequential programming languages.

Running example (step 1/6) The robotics scenario can be modeled in *AbC* as follows:

$$Robot_1 \| \ldots \| Robot_n$$

Each robot is modeled as an *AbC* component ($Robot_i$) of the following form $(\Gamma_i : P_R)$. These components execute in parallel and interact to achieve a specific goal. The attribute *environment* Γ_i specifies a set of attributes for each robot. For instance, attribute *role* can take different values like "*explorer*", "*helping*", "*charger*", or "*rescuer*" according to the current state of the robot.

A *process* is either the inactive process $\mathsf{0}$, an action-prefixed process $\bullet.P$ (where "$\bullet$" is replaced with an action), a name restricted process $\mathtt{new}(x)P$, a match process $\langle\Pi\rangle P$, a choice between two processes $P_1 + P_2$, a parallel composition between two processes $P_1|P_2$, or a recursive call K. We assume that each process has a unique process definition $K \triangleq P$. The construct $\mathtt{new}(x)$ has exactly the same meaning of νx, the only difference is that the former works at the process level while the latter works at the component level. The (Match) construct is used to check the status of a component or its environment. It is different from the one in π-calculus in that it models event-based actions. This construct $\langle\Pi\rangle P$ blocks the execution of process P until the evaluation of the predicate Π under the local environment Γ equals to tt (i.e., $[\![\Pi]\!]_\Gamma = \mathsf{tt}$). The (Parallel) operator allows co-located processes (i.e., residing within the same component) to interact through message-passing rather than just by sharing information via the local environment. In this way, informations could be selectively shared rather than offered to all co-located processes. In what follows, We shall use the notation $[\![\Pi]\!]_\Gamma$ (resp. $[\![E]\!]_\Gamma$) to indicate the evaluation of a predicate Π (resp. an expression E) under the local environment Γ.

Running example (step 2/6) The process P_R running on a robot has the following form:

$$P_R \triangleq (\langle\Pi\rangle a_1.P_1 + a_2.P_2)|P_3$$

This means that the behavior of P_R is a parallel composition of two subprocesses where the one on the left-hand side of "|" can either perform a_1 and continue as P_1 if the evaluation of Π under the local environment equals to tt or perform a_2 and continue as P_2.

There are three kinds of *actions*. The attribute-based input $\Pi(\tilde{x})$ binds to the sequence $\tilde{x}$ the corresponding received values from any process whose attributes satisfy the predicate Π; the attribute-based output $(\tilde{E})@\Pi \vdash_s$ evaluates the sequence of expressions $\tilde{E}$ under the local environment Γ and then broadcasts the result of the evaluation $[\![\tilde{E}]\!]_\Gamma$ to all processes whose attributes satisfy the predicate Π. Set s specifies the exposed attributes names for interaction. This means that a process can control the visibility of its attributes while broadcasting messages to other processes.

The update $[a := E]$ sets the value of an attribute a to the evaluation of the expression E in the local environment. A *predicate* Π either checks the value of an attribute by using some binary operator $\bowtie$ or is the propositional combination of predicates. Predicate tt is satisfied by all attributes and is used for full broadcast while ff is not satisfied by any attribute and in some situations is used to refer to an internal computation.

An *Expression* E is either a constant value $v \in \mathcal{V}$, a variable x, an attribute name a, or a reference to a local attribute name *this.a*. The properties of *self-awareness* and *context-awareness* that are typical for CAS are made possible by allowing to read the values of local attributes referenced by a special name *this* (i.e., *this.a*). These values either represent the current status of a component (i.e., *self-awareness*) or they represent the external environment (i.e.,*context-awareness*). It is worth mentioning that the language of predicates permits evaluating expressions locally. Expressions within predicates contain also variable names and via predicates it is possible to check not only the attributes of the sender but also whether the sent

values satisfy specific conditions. For instance, component: $\Gamma{:}(x > 2 \wedge role = helping)(x)$ receives a value from a component with a "*helping*" role only if it is greater than 2.

We assume that our processes are *closed* (i.e., no free process variables), and free names can be used whenever needed. The constructs νx, $\texttt{new}(x)$, and $\Pi(\tilde{x})$ act as binders for names (i.e., in $\nu x P$, $\texttt{new}(x)P$, and $\Pi(\tilde{x}).P$, x and $\tilde{x}$ are bound). We use the notation $bn(P)$ to denote the set of bound names of P. The free names of P are those that do not occur in the scope of any binder and are denoted by $fn(P)$. The set of names of P is denoted by $n(P)$. The notions of bound and free names is applied equally to components, but in this case free names also includes all attribute values that do not occur in the scope of any binder.

Running example (step 3/6) By specifying the predicate Π and the actions a_1 and a_2 respectively, the process P_R becomes:

$$
\begin{aligned}
&P_R \triangleq \\
&(\langle \texttt{this}.victimPerceived = \mathsf{tt} \rangle \\
&\quad [\texttt{this}.state := stop].P_1 \\
&+ \\
&(\texttt{this}.id, qry)@(role = rescuer \vee role = helping)\vdash_{\{role\}}.P_2 \\
&) \mid P_3
\end{aligned}
$$

The subprocess on the left-hand side of "|" allows the robot to recognize the presence of a victim by locally reading the value of an attribute controlled by its sensors or to help other robots to rescue a victim by sending queries for information about the victim from robots whose role is either "*rescuer*" or "*helping*". If the sensors recognize the presence of a victim and the value of "*victimPerceived*" becomes "tt", the robot updates its "*state*" to "*stop*" which triggers the actuators to stop movement and the process continues as P_1, otherwise the robot sends an query for information about the victim. This query contains the robot identity "*this.id*" and a special name "*qry*" to indicate the request type. The attribute "*role*" is the only exposed attribute for interaction.

3. ABC OPERATIONAL SEMANTICS

In this section, we define the operational semantics of *AbC*; it is defined in two steps. First, the transition relation $\overset{\cdot}{\longmapsto}$. that describes the behavior of *AbC* processes is introduced. Second, this relation is used to define the transition relation $\overset{\cdot}{\longrightarrow}$ that describes the behavior of *AbC* components.

All these transition relations are defined as a label transition system. A transition system is a triple (N, ℓ, R) where N is either a process or a component, ℓ is a transition label, and R is a relation that associates each process or component with another process or component.

3.1 Operational semantics of processes

We use the transition relation $\longmapsto \subseteq Proc \times PLAB \times Proc$ to define the behavior of a process where $Proc$ denotes a process and $PLAB$ is the set of transition labels α which are generated by the following grammar:

$$
\begin{aligned}
&\lambda ::= \nu\tilde{x}\overline{\Gamma{:}(\tilde{v})@\Pi} \quad | \quad \Gamma{:}(\tilde{v})@\Pi \quad | \quad [a := v] \quad | \quad \tau \\
&\alpha ::= \lambda \quad | \quad \widetilde{\Gamma{:}(\tilde{v})@\Pi} \\
&\beta ::= [a := v] \quad | \quad \tau
\end{aligned}
$$

The first three λ-labels are used to denote *AbC* output, input, and update actions respectively while the τ-label denotes an output action with a false predicate. Both output and input labels include the exposed

Table II. : Discarding broadcast

$$(\textbf{FBrd})\ (\tilde{E})@\Pi_1 \vdash_s .P \xmapsto{\Gamma':\widetilde{(\tilde{v})@\Pi_2}}_\Gamma (\tilde{E})@\Pi_1 \vdash_s .P \qquad (\textbf{FRcv})\ \frac{[\![\Pi_1[\tilde{v}/\tilde{x}]]\!]_\Gamma = \Pi_1' \qquad (\Gamma' \not\models \Pi_1')}{\Pi_1(\tilde{x}).P \xmapsto{\Gamma':\widetilde{(\tilde{v})@\Pi_2}}_\Gamma \Pi_1(\tilde{x}).P}$$

$$(\textbf{FUpd})\ [a := E].P \xmapsto{\Gamma':\widetilde{(\tilde{v})@\Pi}}_\Gamma [a := E].P \qquad (\textbf{FZero})\ \ 0 \xmapsto{\Gamma':\widetilde{(\tilde{v})@\Pi}}_\Gamma 0$$

$$(\textbf{FSum})\ \frac{P_1 \xmapsto{\Gamma':\widetilde{(\tilde{v})@\Pi}}_\Gamma P_1 \qquad P_2 \xmapsto{\Gamma':\widetilde{(\tilde{v})@\Pi}}_\Gamma P_2}{P_1 + P_2 \xmapsto{\Gamma':\widetilde{(\tilde{v})@\Pi}}_\Gamma P_1 + P_2} \qquad (\textbf{FMatch1})\ \frac{[\![\Pi]\!]_\Gamma = \mathsf{tt} \qquad P \xmapsto{\Gamma':\widetilde{(\tilde{v})@\Pi}}_\Gamma P}{\langle\Pi\rangle P \xmapsto{\Gamma':\widetilde{(\tilde{v})@\Pi}}_\Gamma \langle\Pi\rangle P}$$

$$(\textbf{FMatch2})\ \frac{[\![\Pi]\!]_\Gamma = \mathsf{ff}}{\langle\Pi\rangle P \xmapsto{\Gamma':\widetilde{(\tilde{v})@\Pi}}_\Gamma \langle\Pi\rangle P} \qquad (\textbf{FRes})\ \frac{P \xmapsto{\Gamma':\widetilde{(\tilde{v})@\Pi}}_\Gamma P}{\mathtt{new}(x)P \xmapsto{\Gamma':\widetilde{(\tilde{v})@\Pi}}_\Gamma \mathtt{new}(x)P}$$

$$(\textbf{FInt})\ \frac{P_1 \xmapsto{\Gamma':\widetilde{(\tilde{v})@\Pi}}_\Gamma P_1 \qquad P_2 \xmapsto{\Gamma':\widetilde{(\tilde{v})@\Pi}}_\Gamma P_2}{P_1|P_2 \xmapsto{\Gamma':\widetilde{(\tilde{v})@\Pi}}_\Gamma P_1|P_2}$$

Table III. : Process labeled semantics (Part 1)

$$(\textbf{Brd})\ \frac{[\![\tilde{E}]\!]_\Gamma = \tilde{v} \qquad [\![\Pi_1]\!]_\Gamma = \Pi}{(\tilde{E})@\Pi_1 \vdash_s .P \xmapsto{\overline{\Gamma|_s:(\tilde{v})@\Pi}}_\Gamma P} \qquad (\textbf{Rcv})\ \frac{[\![\Pi_1[\tilde{v}/\tilde{x}]]\!]_\Gamma = \Pi_1' \qquad (\Gamma' \models \Pi_1')}{\Pi_1(\tilde{x}).P \xmapsto{\Gamma':(\tilde{v})@\Pi_2}_\Gamma P[\tilde{v}/\tilde{x}]} \qquad (\textbf{Upd})\ \frac{[\![E]\!]_\Gamma = v}{[a := E].P \xmapsto{[a:=v]}_\Gamma P}$$

$$(\textbf{Match})\ \frac{[\![\Pi]\!]_\Gamma = \mathsf{tt} \qquad P \xmapsto{\lambda}_\Gamma P'}{\langle\Pi\rangle P \xmapsto{\lambda}_\Gamma P'} \qquad (\textbf{Sum})\ \frac{P_1 \xmapsto{\lambda}_\Gamma P_1'}{P_1 + P_2 \xmapsto{\lambda}_\Gamma P_1'} \qquad (\textbf{Rec})\ \frac{P \xmapsto{\alpha}_\Gamma P'}{K \xmapsto{\alpha}_\Gamma P'}\ K \triangleq P$$

$$(\textbf{PRes})\ \frac{P[y/x] \xmapsto{\lambda}_\Gamma P'}{\mathtt{new}(x)P \xmapsto{\lambda}_\Gamma \mathtt{new}(y)P'}\ y \notin n(\lambda) \wedge y \notin fn(P)\backslash\{x\} \qquad (\textbf{tau})\ \frac{P \xmapsto{\nu\tilde{x}\overline{\Gamma':(\tilde{v})@\mathsf{ff}}}_\Gamma P'}{P \xmapsto{\tau}_\Gamma P'}$$

$$(\textbf{Hide1})\ \frac{P \xmapsto{\nu\tilde{x}\overline{\Gamma':(\tilde{v})@\Pi}}_\Gamma P' \qquad (\Pi \blacktriangleright y) = \mathsf{ff}}{\mathtt{new}(y)P \xmapsto{\nu\tilde{x}\overline{\Gamma':(\tilde{v})@\mathsf{ff}}}_\Gamma \mathtt{new}(y)\mathtt{new}(\tilde{x})P'}\ y \in n(\Pi) \qquad (\textbf{Hide2})\ \frac{P \xmapsto{\nu\tilde{x}\overline{\Gamma':(\tilde{v})@\Pi}}_\Gamma P' \qquad (\Pi \blacktriangleright y) \neq \mathsf{ff}}{\mathtt{new}(y)P \xmapsto{\nu\tilde{x}\overline{\Gamma':(\tilde{v})@\Pi \blacktriangleright y}}_\Gamma \mathtt{new}(y)P'}\ y \in n(\Pi)$$

$$(\textbf{Open})\ \frac{P[y/x] \xmapsto{\overline{\Gamma':(\tilde{v})@\Pi}}_\Gamma P' \qquad \Pi \neq \mathsf{ff}}{\mathtt{new}(x)P \xmapsto{\nu y\overline{\Gamma':(\tilde{v})@\Pi}}_\Gamma P'}\ y \in \tilde{v}\backslash n(\Pi) \wedge y \notin fn(C)\backslash\{x\}$$

environment of the sending process Γ, the predicate that specifies the communication partners Π, and the transmitted or received message $\tilde{v}$. An output is called "bound" if its label contains a bound name (i.e., if $\tilde{x} \neq \emptyset$). The update label includes an attribute name a and the update value v. The α-labels include an additional label $\Gamma:\widetilde{(\tilde{v})@\Pi}$ to denote the case where a process is not able to receive a broadcast. As it will be shown later in this section, this kind of labels is crucial to handle appropriately dynamic constructs like Choice and Match. The notion of free names in α proceeds as follows:

— $fn(\nu\tilde{x}\overline{\Gamma:(\tilde{v})@\Pi}) = (fn(\Pi) \cup fn(\Gamma) \cup \tilde{v})\backslash\tilde{x}$

— $fn(\Gamma:(\tilde{v})@\Pi) = fn(\Pi) \cup fn(\Gamma) \cup \tilde{v}$

Table IV. : Process labeled semantics (Part 2)

$$(\textbf{PInt1})\ \frac{P_1 \overset{\Gamma':(\tilde{v})@\Pi}{\longmapsto_\Gamma} P_1' \qquad P_2 \overset{\Gamma':\widetilde{(\tilde{v})@\Pi}}{\longmapsto_\Gamma} P_2}{P_1|P_2 \overset{\Gamma':(\tilde{v})@\Pi}{\longmapsto_\Gamma} P_1'|P_2} \qquad (\textbf{PSyn})\ \frac{P_1 \overset{\Gamma':(\tilde{v})@\Pi}{\longmapsto_\Gamma} P_1' \qquad P_2 \overset{\Gamma':(\tilde{v})@\Pi}{\longmapsto_\Gamma} P_2'}{P_1|P_2 \overset{\Gamma':(\tilde{v})@\Pi}{\longmapsto_\Gamma} P_1'|P_2'}$$

$$(\textbf{PCom})\ \frac{P_1 \overset{\nu\tilde{x}\overline{\Gamma':(\tilde{v})@\Pi}}{\longmapsto_\Gamma} P_1' \qquad P_2 \overset{\Gamma':(\tilde{v})@\Pi}{\longmapsto_\Gamma} P_2' \qquad \Pi \neq \mathsf{ff}}{P_1|P_2 \overset{\nu\tilde{x}\overline{\Gamma':(\tilde{v})@\Pi}}{\longmapsto_\Gamma} P_1'|P_2'}\ \tilde{x} \cap fn(P_2) = \emptyset \qquad (\textbf{PInt2})\ \frac{P_1 \overset{\beta}{\mapsto_\Gamma} P_1'}{P_1|P_2 \overset{\beta}{\mapsto_\Gamma} P_1'|P_2}$$

$$(\textbf{FCom})\ \frac{P_1 \overset{\nu\tilde{x}\overline{\Gamma':(\tilde{v})@\Pi}}{\longmapsto_\Gamma} P_1' \qquad P_2 \overset{\Gamma':\widetilde{(\tilde{v})@\Pi}}{\longmapsto_\Gamma} P_2' \qquad \Pi \neq \mathsf{ff}}{P_1|P_2 \overset{\nu\tilde{x}\overline{\Gamma':(\tilde{v})@\Pi}}{\longmapsto_\Gamma} P_1'|P_2'}\ \tilde{x} \cap fn(P_2) = \emptyset$$

— $fn([a := v]) = \{v\}$

— $fn(\Gamma{:}\widetilde{(\tilde{v})@\Pi}) = fn(\Pi) \cup fn(\Gamma) \cup \tilde{v}$

— $fn(\tau) = \emptyset$

where $fn(\Gamma)$ is the co-domain of Γ. The free names of a predicate depend on the structure of the predicate in the sense that $fn(\mathsf{tt}) = fn(\mathsf{ff}) = \emptyset,\ fn(a = v) = \{v\},\ fn(\neg\Pi) = fn(\Pi),$ and if the predicate is a composite of subpredicates, the free names is the union of the free names of subpredicates. Only the output label has bound names (i.e., $bn(\nu\tilde{x}\overline{\Gamma:(\tilde{v})@\Pi}) = \tilde{x}$).

The transition relation $\longmapsto_\Gamma$ is formally defined in Table II, Table III, and Table IV. This relation is parametrized with respect to the local environment Γ where the process resides. We start with the set of rules that describes the meaning of the discarding label $\Gamma{:}\widetilde{(\tilde{v})@\Pi}$ in Table II, since the rest sets of rules depend on it. Rule (**FBrd**) states that any process that is ready to broadcast can discard broadcasts from other processes and stay unchanged. Rule (**FRcv**) models that fact that if the sender does not satisfy the expected requirements ($\Gamma' \not\models \Pi_1'$) then the process will discard the broadcasted message and stay unchanged. Rule (**FUpd**) models that fact that any process that is busy updating some of its attributes can discard broadcasts from other processes and stay unchanged. Rule (**FZero**) states that process $\mathsf{0}$ always discards broadcasts from other processes. Rule (**FSum**) models the fact that process $P_1 + P_2$ discards a broadcast if both its subprocesses P_1 and P_2 are able to do so. Note that the choice is not dissolved after a broadcast refusal and this is very important to capture the dynamic nature of Choice. Rule (**FMatch1**) states that process $\langle\Pi\rangle P$ can discard a broadcast even if the evaluation result of its predicate Π under the local environment Γ equals to (tt) as long as the process P is able to discard the same broadcast. Rule (**FMatch2**) models the fact that if the evaluation result of predicate Π in process $\langle\Pi\rangle P$ under the local environment Γ equals to ff, then process $\langle\Pi\rangle P$ can discard any broadcast from other processes. Note that the Match does not dissolve after a broadcast refusal. Rule (**FRes**) models the fact that process $\mathtt{new}(x)P$ discards a broadcast if process P does the same. Rule (**FInt**) has exactly the same meaning of Rule (**FSum**), the Parallel process discards a broadcast if both its subprocesses do the same.

Running example (step 4/6) Assume that $Robot_1$ whose role is "*explorer*" is busy searching for a victim in some arena and that process P_3 in our example can only perform Update actions. If a process residing in $Robot_2$ whose role is "*charger*" broadcasts information about a nearby charging station then process P_R that resides in $Robot_1$ can evolve as follows:

$$P_R \overset{\Gamma_2':(info)@\widetilde{(role=explorer)}}{\longmapsto_{\Gamma_1}} P_R$$

Where Γ'_2 is the exposed portion of the local environment of $Robot_2$ and Γ_1 is the local environment of $Robot_1$. Process P_R applies rule (**FInt**) and discards the broadcast because both its subprocesses can discard the message. P_3 is not ready for receiving messages, so it applies (**FUpd**) and stay unchanged. The choice also discards the message by applying (**FSum**) because both its subprocesses can discard the message. The subprocess on the left-hand side of + is not ready for receiving messages, so it applies (**FMatch2**) and stay unchanged. The subprocess on the right-hand side of + applies (**FBrd**) and discards the broadcast, because it is not ready for receiving messages.

The set of rules in Table III describes the sequential behavior of AbC processes. To simplify the presentation, we omitted the symmetrical rule for **Sum**. Rule (**Brd**) evaluates the sequence of expressions $\tilde{E}$ and the predicate Π_1 under the local environment Γ, computes the exposed portion of the local environment $\Gamma|s$, sends these information in the broadcast, and continues as P. The exposed portion of Γ in the broadcasted message is computed as follows:

$$\Gamma|_s = \begin{cases} \Gamma(a) & \texttt{if } a \in s \\ \bot & \texttt{otherwise} \end{cases} \tag{1}$$

Rule (**Rcv**) replaces the free occurrences of the input sequence variables $\tilde{x}$ in the receiving predicate Π_1 with the corresponding message values $\tilde{v}$ and evaluates Π_1 under the local environment Γ. If the attached sender environment Γ' satisfies the evaluated predicate Π'_1, the input action is performed and the substitution $[\tilde{v}/\tilde{x}]$ is applied to the continuation process P. Rule (**Upd**) evaluates the expression E under the local environment Γ to compute v, performs the update action and continue as P. Rule (**Match**) evaluates the predicate Π under the local environment Γ. If the evaluation equals to $\texttt{tt}$, process $\langle\Pi\rangle P$ proceeds by performing an action with a λ-label and continues as P' if process P can perform the same action. Rule (**Sum**) and its symmetric represent the non-deterministic choice between the subprocesses P_1 and P_2 in the sense that if any of them say P_1 performs an action with a λ-label and becomes P'_1 then the overall process continues as P'_1. Rule (**Rec**) is standard for process definition. Rule (**PRes**) models the fact that process $\texttt{new}(x)P$ 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 process P can perform the same action. If necessary, we allow renaming with conditions that ensure a coherent behavior. Note that name restriction operator does not dissolve after this transition. Rule (**tau**) models the fact that if a process broadcasts a message with a false predicate then the whole message is not exposed and is considered as an internal action.

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 V. In process calculi where broadcasting is the basic primitive for communication like CSP [Hoare 1978] and $b\pi$-calculus [Milner et al. 1992], broadcasting on a private channel is equal to performing an internal action and no other process can observe the broadcast except the one that performed it. For example in $b\pi$-calculus, assume that $P = \nu a(P_1 \| P_2) \| P_3$ where $P_1 = \bar{a}v.Q$, $P_2 = a(x).R$, and $P_3 = b(x)$. Now if P_1 broadcasts on a then only P_2 can observe it since P_2 is included in the scope of the restriction. P_3 and other processes only observe an internal action, so $P \xrightarrow{\tau} \nu a(Q \| R[v/x]) \| b(x)$.

Table V. : Predicate restriction $\bullet\blacktriangleright x$

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

This idea is generalized in the new 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 is not

exposed according to (**tau**) rule (i.e., broadcasting with a false predicate). If the predicate restriction operator returns something different from ff then the message is exposed with a smaller predicate and the restricted name remains private. Note that any private name in the message (i.e., $\tilde{x}$) remains private if $(\Pi \blacktriangleright y) = \mathsf{ff}$ (i.e., the broadcast is not exposed) 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. This kind of behavior is very useful when modeling 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 the messages are propagated through the network. This transparency comes from the fact that networks propagate messages between their nodes through private channels while exposing the interesting messages to users through public channels.

For instance, if a network broadcasts 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. The proof is carried out by induction on the structure of Π. See Appendix A, Page 22. □

Rule (**Open**) models the fact that a process has the ability to communicate a private name to other processes and this name is included in the message itself $\tilde{v}$. This rule is different from the rule that you find 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 processes but rather the scope is dissolved. In other words, when a private name is communicated in multiparty settings then the name is not private anymore. Note that, a process that is broadcasting on a false predicate (i.e., $\Pi = \mathsf{ff}$) cannot open the scope.

The set of rules in Table IV describes the parallel behavior of AbC processes. To simplify the presentation, we omitted the symmetrical rules for **PInt1**, **PInt2**, **PCom**, and **FCom**. Rule (**PInt1**) models the interleaving when performing an input action in the sense that process P_1 inputs a broadcast and continues as P_1' while P_2 discards the broadcast and stays still. Rule (**PSyn**) states that two processes P_1 and P_2 that execute in parallel can synchronize while performing an input action. This models the fact that the same broadcast is received by the both P_1 and P_2. Rule (**PCom**) states that two processes P_1 and P_2 that execute in parallel can communicate if P_1 can broadcast a message with a predicate that is different from ff and P_2 is willing to receive that message. Rule (**PInt2**) models the interleaving between the processes P_1 and P_2 when performing an action with a β-label (i.e., $[a := v]$ and τ). Finally, Rule (**FCom**) models the non-blocking nature of the broadcast in the sense that P_1 can broadcast messages irrespective of the presence of listeners and this is modeled by allowing P_2 to discard the broadcast. This means that P_1 is not really aware of the message reception. Note that the input action is blocking in that it can only take place through synchronization with an available broadcast. Since the broadcasted message might have a bound name (i.e., $\tilde{x} \neq \emptyset$), in both rules (**PCom**) and (**FCom**), the side condition is added to avoid name clashing.

Running example (step 5/6) The process P_R running on a robot, say $Robot_1$ with an attribute environment Γ and $\Gamma(id) = 1$, apart from the behavior of P_3 (not specified here) or the possibility to discard incoming broadcasts, has the following possible transitions:

$$P_R \xmapsto{[\texttt{this}.state:=stop]}_{\Gamma_1} \quad P_1|P_3$$

$$P_R \xmapsto{\overline{\Gamma':(1,\ qry)@(role=rescuer\ \vee\ role=helping)}}_{\Gamma_1} \quad P_2|P_3$$

Where $\Gamma' = \Gamma_1|_{\{role\}}$ according to Equation 1.

Table VI. : Component labeled semantics

$$(\textbf{C-Brd})\frac{P \overset{\overline{\Gamma':(\tilde{v})@\Pi}}{\longmapsto}_{\Gamma} P'}{\Gamma : P \overset{\overline{\Gamma':(\tilde{v})@\Pi}}{\longrightarrow} \Gamma : P'} \qquad (\textbf{C-Rcv})\frac{P \overset{\Gamma':(\tilde{v})@\Pi}{\longmapsto}_{\Gamma} P'}{\Gamma : P \overset{\Gamma':(\tilde{v})@\Pi}{\longrightarrow} \Gamma : P'} (\Gamma \models \Pi)$$

$$(\textbf{C-FRcv})\ \Gamma : P \overset{\Gamma':(\tilde{v})@\Pi}{\longrightarrow} \Gamma : P \qquad (\Gamma \not\models \Pi) \qquad (\textbf{C-Fail})\frac{P \overset{\widetilde{\Gamma':(\tilde{v})@\Pi}}{\longmapsto}_{\Gamma} P}{\Gamma : P \overset{\Gamma':(\tilde{v})@\Pi}{\longrightarrow} \Gamma : P}$$

$$(\textbf{C-Upd})\frac{P \overset{[a:=v]}{\longmapsto}_{\Gamma} P'}{\Gamma : P \overset{\tau}{\rightarrow} \Gamma[a \mapsto v] : P'} \qquad (\textbf{C-tau})\frac{C \overset{\nu\tilde{x}\overline{\Gamma:(\tilde{v})@\mathsf{ff}}}{\longrightarrow} C'}{C \overset{\tau}{\rightarrow} C'}$$

$$(\textbf{Res})\frac{C[y/x] \overset{\gamma}{\rightarrow} C'}{\nu x C \overset{\gamma}{\rightarrow} \nu y C'}\ y \notin n(\gamma) \wedge y \notin fn(C)\backslash\{x\} \qquad (\textbf{LHide})\frac{P \overset{\tau}{\longmapsto}_{\Gamma} P'}{\Gamma : P \overset{\tau}{\rightarrow} \Gamma : P'}$$

$$(\textbf{EHide1})\frac{C \overset{\nu\tilde{x}\overline{\Gamma:(\tilde{v})@\Pi}}{\longrightarrow} C' \qquad (\Pi \blacktriangleright y) = \mathsf{ff}}{\nu y C \overset{\nu\tilde{x}\overline{\Gamma:(\tilde{v})@\mathsf{ff}}}{\longrightarrow} \nu y \nu \tilde{x} C'}\ y \in n(\Pi) \qquad (\textbf{EHide2})\frac{C \overset{\nu\tilde{x}\overline{\Gamma:(\tilde{v})@\Pi}}{\longrightarrow} C' \qquad (\Pi \blacktriangleright y) \neq \mathsf{ff}}{\nu y C \overset{\nu\tilde{x}\overline{\Gamma:(\tilde{v})@\Pi \blacktriangleright y}}{\longrightarrow} \nu y C'}\ y \in n(\Pi)$$

$$(\textbf{EOpen})\frac{C[y/x] \overset{\overline{\Gamma:(\tilde{v})@\Pi}}{\longrightarrow} C' \qquad \Pi \neq \mathsf{ff}}{\nu x C \overset{\nu y\overline{\Gamma:(\tilde{v})@\Pi}}{\longrightarrow} C'}\ y \in \tilde{v}\backslash n(\Pi) \wedge y \notin fn(C)\backslash\{x\} \qquad (\textbf{LOpen})\frac{P \overset{\nu x\overline{\Gamma':(\tilde{v})@\Pi}}{\longmapsto}_{\Gamma} P'}{\Gamma : P \overset{\nu x\overline{\Gamma':(\tilde{v})@\Pi}}{\longrightarrow} \Gamma : P'}$$

$$(\tau\textbf{-Int})\frac{C_1 \overset{\tau}{\rightarrow} C_1'}{C_1 \| C_2 \overset{\tau}{\rightarrow} C_1' \| C_2} \qquad (\textbf{Sync})\frac{C_1 \overset{\Gamma:(\tilde{v})@\Pi}{\longrightarrow} C_1' \quad C_2 \overset{\Gamma:(\tilde{v})@\Pi}{\longrightarrow} C_2'}{C_1 \parallel C_2 \overset{\Gamma:(\tilde{v})@\Pi}{\longrightarrow} C_1' \parallel C_2'}$$

$$(\textbf{Com})\frac{C_1 \overset{\nu\tilde{x}\overline{\Gamma:(\tilde{v})@\Pi}}{\longrightarrow} C_1' \quad C_2 \overset{\Gamma:(\tilde{v})@\Pi}{\longrightarrow} C_2' \qquad \Pi \neq \mathsf{ff}}{C_1 \parallel C_2 \overset{\nu\tilde{x}\overline{\Gamma:(\tilde{v})@\Pi}}{\longrightarrow} C_1' \parallel C_2'}\ \tilde{x} \cap fn(C_2) = \emptyset$$

3.2 Operational semantics of component

We use the transition relation $\longrightarrow \subseteq Comp \times CLAB \times Comp$ to define the behavior of a component where $Comp$ denotes a component and $CLAB$ is the set of transition labels γ which are generated by the following grammar:

$$\gamma ::= \nu\tilde{x}\overline{\Gamma:(\tilde{v})@\Pi} \quad | \quad \Gamma:(\tilde{v})@\Pi \quad | \quad \tau$$

The first two labels denote AbC output and input actions respectively while the τ-label denotes either an update action or an output action with a false predicate. The notions of free and bound names are exactly the same as defined in the previous section. The definition of the transition relation $\longrightarrow$ depends on the definition of $\longmapsto$ in the previous section in the sense that the behavior of the process-level is lifted to the component-level. The transition relation $\longrightarrow$ is formally defined in Table VI. To simplify the presentation, we omitted the symmetrical rules for τ**-Int** and **Com**. Rule (**C-Brd**) models the fact that a component $\Gamma : P$ can broadcast a message $\tilde{v}$ with a predicate Π, if its running process P is willing to do so. Γ' refers to the exposed portion of the local environment Γ that is used for communication. Rule (**C-Rcv**) states that a component $\Gamma : P$ can receive a broadcast only when its local environment Γ satisfies the predicate of the broadcast Π and its running process is willing to receive. Rule (**C-FRcv**) states that any component $\Gamma : P$ can discard a broadcast and stay unchanged if its local environment Γ does not satisfy the predicate of the broadcast Π. On the other hand, rule (**C-Fail**) states that any component $\Gamma : P$ can discard a broadcast and stay unchanged if its running

process is willing to do so. Rule (**C-Upd**) states that a component $\Gamma : P$ can update the value of attribute a to v by performing an internal action, if its running process P is willing to perform an update action for attribute a. Notation $\Gamma[a \mapsto v]$ denotes the environment update: $\Gamma[a \mapsto v](a') = \Gamma(a')$ if $a \neq a'$ and v otherwise. Rules **C-tau**, **Res**, **EHide1**, **EHide2**, **EOpen**, and **Sync** have the same meaning of the rules **tau**, **PRes**, **Hide1**, **Hide2**, **Open**, and **PSyn** respectively. The only difference is that the former rules work at the component level while the latter work at the process level. Rule (**LHide**) models local computations at the process level as internal actions at the component level. Rule (**LOpen**) exports private names from the process level to the component level. Rule (τ**-Int**) models the interleaving between components C_1 and C_2 when performing an internal action. Rule (**Com**) states that two components C_1 and C_2 that execute in parallel can communicate if C_1 can broadcast a message with a predicate that is different from ff and C_2 can possibly receive that message.

Running example (step 6/6) If we further specify the subprocess P_2 in the process P_R running on the robots, it becomes:

$$
\begin{array}{l}
\mathit{Query} \triangleq \\
(\texttt{this}.\mathit{id},\ \mathit{qry})@(\mathit{role} = \mathit{rescuer} \vee \mathit{role} = \mathit{helping}) \vdash_{\{\mathit{role}\}} . \\
(\quad ((\mathit{role} = \mathit{rescuer} \vee \mathit{role} = \mathit{helping}) \wedge \mathit{reply} = \mathit{ack}) \\
\qquad\qquad\qquad\qquad\qquad\qquad (\mathit{vpos},\ c,\ \mathit{reply}).P_2' \\
\quad + \\
\quad \mathit{Query} \\
)
\end{array}
$$

Basically, the explorer robot continuously sends queries to other robots whose role is either "*rescuer*" or "*helping*". Once a reply comes containing the victim position "*vpos*", the number of robots required to rescue the victim "*c*", and an acknowledgement "*reply*", the explorer binds these values in the continuation process P_2' and continues execution.

Let us assume that the role of Robot_2 is "*rescuer*", Robot_3 is "*helping*", and all other nearby robots are explorers. Robot_3 has already received the victim information from Robot_2 and now Robot_3 is the only one who is responsible for communicating these information to other robots. Robot_1 has already sent a message containing its identity "*this.id*" and "*qry*" request and Robot_3 caught it. Now by using rule (**C-Brd**), Robot_3 sends the information back to Robot_1 by generating this transition.

$$\Gamma_3 : P_{R_3} \xrightarrow{\overline{\Gamma:(\mathit{vloc},\ 2,\ \mathit{ack})@(\mathit{id}=1)}} \Gamma_3 : P'_{R_3}$$

where $\Gamma = \Gamma_3|_{\{\mathit{role}\}}$. On the other hand, Robot_1 applies rule (**C-Rcv**) to receive victim information and generates this transition.

$$\Gamma_1 : P_{R_1} \xrightarrow{\Gamma:(\mathit{vloc},\ 2,\ \mathit{ack})@(\mathit{id}=1)} \Gamma_1 : P_2'[\mathit{vloc}/\mathit{vpos},\ 2/c,\ \mathit{ack}/\mathit{reply}]$$

These robots can perform the above transitions since $\Gamma_1 \models (\mathit{id} = 1)$ and $\Gamma \models ((\mathit{role} = \mathit{rescuer} \vee \mathit{role} = \mathit{helping}) \wedge \mathit{reply} = \mathit{ack})$. Other robots which are not addressed by communication discard the broadcast by applying rule (**C-FRcv**) (i.e., every robot has a unique *id*, so only $\Gamma_1 \models (\mathit{id} = 1)$). Now the overall system evolves by applying rule (**Com**) as follows:

$$S \xrightarrow{\overline{\Gamma:(vloc,\ 2,\ ack)@(id=1)}} \begin{array}{l} \Gamma_1 : P_2'[vloc/vpos,\ 2/c,\ ack/reply] \\ \| \ \Gamma_2 : P_{R_2} \| \ \Gamma_3 : P'_{R_3} \| \ \ldots \| \ \Gamma_n : P_{R_n} \end{array}$$

4. THE EXPRESSIVENESS OF ABC CALCULUS

In this section, we show the expressive power of *AbC* by first providing a complete model for the swarm robotics scenario that is introduced in Section 1 and used as a running example in previous sections. Second, we show how some other communication models and programming frameworks can be seamlessly translated into *AbC*. The idea is to demonstrate how natural and intuitive it is to use *AbC* to model systems where collaboration, adaptation, and reconfiguration are the main concerns. And also to show that attribute-based communication is rich enough to encode the already existing approaches.

4.1 A swarm robotics model in *AbC*

The swarm robotics model exploits the fact that a process running on a robot can either only read the values of some attributes that are controlled by its sensors (i.e., their values are provided by sensors) or read and update the other attributes in its local environment. Reading the values of the attributes controlled by sensors either provides the robot with information about its environment and this models what is called (*context-awareness*) or provides the robot with information about its current status and this models what is called (*self-awareness*). For instance, reading the value of the *collision* attribute in the local environment $\Gamma(collision) = \mathsf{tt}$ makes the robot aware that an imminent collision with a wall in the arena has been detected and this triggers an adaptation mechanism to change the direction of the robot. On the other hand, reading the value of the *batteryLevel* attribute in the local environment $\Gamma(batteryLevel) = 15\%$ makes the robot aware that its battery level is critical (i.e., $< 20\%$) and this triggers an adaptation mechanism to stop movement by mean of an actuation signal for halting the actuators and the robot enters the power saving mode.

We assume that each robot has a unique identity (*id*) and since the robot acquires information about its environment or its own status by mean of reading the values provided by sensors, no additional assumptions about the initial state are needed. It is worth mentioning that sensors and actuators are not modeled by *AbC* as they represent the robot internal infrastructure while *AbC* model represents the programmable behavior of the robot (i.e., its running code).

The robotics scenario is modeled as a set of parallel *AbC* components, each component represents a robot ($Robot_1 \| \ldots \| Robot_n$) and each robot has the following form ($\Gamma_i : P_R$). The behavior of a single robot is

modeled in the following AbC process P_R:

$$
\begin{array}{l}
P_R \triangleq \\
(\quad \langle \texttt{this}.victimPerceived = \mathsf{tt} \rangle \\
\qquad [\texttt{this}.state := stop]. \\
\qquad [\texttt{this}.vPosition := vLoc]. \\
\qquad [\texttt{this}.count := 3]. \\
\qquad [\texttt{this}.role := rescuer]. \\
\qquad (role = explorer \wedge y = qry)(x,\ y). \\
\qquad (\texttt{this}.vPosition,\ \texttt{this}.count,\ ack)@(id = x) \vdash_{\{role\}} \\
\quad + \\
\qquad Query \\
)|\ RandomWalk \mid IsMoving
\end{array}
$$

The robot follows a random walk in exploring the disaster arena. The robot recognizes the presence of a victim by mean of locally reading the value of an attribute controlled by its sensors or help other robots to rescue a victim by mean of sending queries for information about the victim from other robots whose role is either "*rescuer*" or "*helping*". If the sensors recognize the presence of a victim and the value of "*victimPerceived*" becomes "tt", the robot updates its "*state*" to "*stop*" which triggers an actuation signal to the actuators to stop movement, computes the victim position and the number of the required robots to rescue the victim and store them in the attributes "*vPosition*" and "*count*" respectively, changes its role to "*rescuer*", and waits for queries from nearby explorers. Once a query is received, the robot sends back the victim information to the requesting robot addressing it by its "*id*" and the collective (i.e., the swarm) starts forming in preparation for the rescuing procedure.

On the other hand, if the victim is still not perceived, the robot continuously sends queries for information about the victim to the nearby robots whose role is either "rescuer" or "*helping*". This query contains the robot identity "*this.id*" and a special name "*qry*" to indicate the request type. If an acknowledgement arrives containing victim's information, the robot changes its role to "*helping*" and start the helping procedure.

Remark 4.1. The interaction between an explorer robot currently running "*Query*" and a rescuer robot that is waiting for a request from nearby explorers suggests a possible way of modeling point-to-point communication like in π-calculus [Milner et al. 1992].

$$
\begin{array}{l}
Query \triangleq \\
(\texttt{this}.id,\ qry)@(role = rescuer \vee role = helping) \vdash_{\{role\}} . \\
(\ ((role = rescuer \vee role = helping) \wedge reply = ack) \\
\qquad\qquad\qquad\qquad\qquad\qquad (vpos,\ c,\ reply). \\
\qquad\qquad\qquad\qquad\qquad\qquad [\texttt{this}.role := helping]. \\
\qquad\qquad\qquad\qquad\qquad\qquad HelpingRescuer \\
\ + \\
\ Query \\
)
\end{array}
$$

The "*helpingRescuer*" process defined below is triggered by receiving the victim information from the rescuer-collective as mentioned above.

$$
\begin{array}{l}
HelpingRescuer \triangleq \\
[\texttt{this}.vPosition := vpos]. \\
[\texttt{this}.target := vpos]. \\
(\ \langle \texttt{this}.position = \texttt{this}.target \rangle \\
\quad [\texttt{this}.role := rescuer] \\
\ | \\
\ \langle c > 1 \rangle \\
\quad (role = explorer \wedge y = qry)(x,\ y). \\
\quad (\texttt{this}.vPosition,\ c-1,\ ack)@(id = x) \vdash_{\{role\}} \ )
\end{array}
$$

The helping robot stores the victim position in the attribute "*vPosition*" and updates its target to be the victim position. This triggers the actuators to move to the specified location. The robot waits until it reaches the victim and at the same time is willing to respond to other robots queries, if more than one robot is needed for the rescuing procedure. Once the robot reaches the victim (i.e., its position coincides with the victim position), the robot changes its role to "*rescuer*" and join the rescuer-collective.

The "*RandomWalk*" process is defined below. This process computes a random direction to be followed by the robot. Once a collision is detected by the proximity sensor, a new random direction is calculated.

$$
\begin{array}{l}
RandomWalk \triangleq [\texttt{this}.direction := 2\pi rand()]. \\
\quad \langle \texttt{this}.collision = \mathsf{tt} \rangle \\
\quad\quad RandomWalk
\end{array}
$$

Finally, process "*IsMoving*" captures the status of the battery level in a robot at any time. Once the battery level drops into a critical level (i.e., 20%), the robot changes its status to "*stop*" which results in halting the actuators and the robot enters the power saving mode. The robot stays in this mode until it is recharged to at least 90% and then it starts moving again.

$$
\begin{array}{l}
IsMoving \triangleq \\
\quad \langle \texttt{this}.state = move \wedge \neg(\texttt{this}.batteryLevel > 20\%) \rangle \\
\quad\quad [\texttt{this}.state := stop]. \\
\quad \langle \texttt{this}.batteryLevel > 90\% \rangle \\
\quad\quad [\texttt{this}.state := move]. \\
\quad\quad IsMoving
\end{array}
$$

For simplifying the presentation, we do not model the charging task in this scenario and we assume that this task is accomplished according to some predefined procedure. It is worth mentioning that if more victims are found in the arena, different rescuer-collectives will be spontaneously formed to rescue them. To avoid forming multiple collectives for the same victim, we assume that sensors only detect isolated victims. Light-based message communication [OGrady et al. 2010] between robots can be used. Thus once a robot has reached a victim, it signals with a specific color light to other robots not to discover the victim next to it [Pinciroli et al. 2015]. Since we do not model the failure recovery in this scenario, we assume that all robots are fault-tolerant and they cannot fail. For more details, a runtime environment for the SCEL language where *AbC* is inspired from can be found in the following website "http://jresp.sourceforge.net". Different case studies from different domains have been implemented in an attribute-based fashion.

Table VII. : Encoding bπ-calculus into AbC

(Component Level)
$(\!|\ P\ |\!)_\Gamma \triangleq \Gamma : (\!|\ P\ |\!)$

(Process Level)
$(\!|\ \texttt{nil}\ |\!) \triangleq 0 \qquad (\!|\ \tau.P\ |\!) \triangleq [Port_\tau := \tau].(\!|\ P\ |\!)$

$(\!|\ P + Q\ |\!) \triangleq (\!|\ P\ |\!) + (\!|\ Q\ |\!)$

$(\!|\ a(\tilde{x}).P\ |\!) \triangleq \Pi(\tilde{x}).(\!|\ P\ |\!) \quad \texttt{with} \quad \Pi = (\texttt{Port}_a = a)$

$(\!|\ \bar{a}\tilde{x}.P\ |\!) \triangleq (\tilde{x})@\Pi \vdash_{\{Port_a\}} .(\!|\ P\ |\!) \quad \texttt{with} \quad \Pi = (\texttt{Port}_a = a)$

$(\!|\ (rec\ A\langle\tilde{x}\rangle).P)\langle\tilde{y}\rangle\ |\!) \triangleq (A \triangleq (\!|\ P\ |\!))$

$(\!|\ P \| Q\ |\!) \triangleq (\!|\ P\ |\!) | (\!|\ Q\ |\!)$

$(\!|\ \nu x P\ |\!) \triangleq \texttt{new(}x\texttt{)}(\!|\ P\ |\!)$

$(\!|\langle x = y\rangle P, Q\ |\!) \triangleq \ \langle x = y\rangle(\!|\ P\ |\!) \ + \langle x \neq y\rangle(\!|\ Q\ |\!)$

4.2 Encoding the $b\pi$-calculus

In this section, we study the relative expressiveness of attribute-based communication, mainly in comparison to channel-based communication with value-passing. We choose $b\pi$-calculus [Ene and Muntean 2001] as a representative for channel-based process calculi and translate it into AbC. What makes $b\pi$-calculus a proper choice is that $b\pi$-calculus uses broadcast instead of binary synchronization as a basic primitive for communication which makes it a sort of variant of value-passing CBS [Prasad 1991]; channels in $b\pi$-calculus can be also communicated like in π-calculus [Milner et al. 1992] which is considered as one of the richest paradigm introduced for concurrency so far. Based on a separation result presented in [Ene and Muntean 1999], it has been proved 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 cannot. Like $b\pi$-calculus, AbC components broadcast one at a time and are instantaneously heard by all components satisfying the predicate of the broadcast. However, a possible way of modeling π-calculus point-to-point communication into AbC is already hinted in Remark 4.1, Section 4.1. 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 result between AbC and π-calculus.

The encoding of a $b\pi$-calculus process P is rendered as an AbC component $\Gamma_p : P$ where the process environment Γ_p is defined as follows.

Definition 4.2 (*Process Environment*). Let P be a $b\pi$-process and Ch be a countable set of channels such that $Ch \subseteq n(P) \cup \{\tau\}$, then there exists a corresponding process environment Γ_p in AbC such that $\Gamma_p = \{(port_x, x) |\ \text{for all } x \in Ch\}$.

The only feature of $b\pi$-calculus that is not present in AbC is the possibility of specifying the name of a channel where the exchange happens instantaneously (i.e., the communication channel instantly appears at the time of communication and disappears afterwards). AbC relies on the satisfaction of predicates over attributes for driving the communication instead. So, these attributes are always available in the local environment and cannot disappear at any time. However, this is not a problem in the sense that we can exploit the fact that AbC processes have the ability to specify the set of exposed attributes for communication.

By simply restricting this set to be a singleton (i.e., every broadcast contains only a single exposed attribute), we can have the same behavior. This approach can be illustrated in the following encoding:

$$(\!| \, \bar{a}\tilde{x}.P \, |\!) \triangleq (\tilde{x})@\Pi \vdash_{\{Port_a\}} .(\!| \, P \, |\!) \quad \texttt{with} \quad \Pi = (\texttt{Port}_a = a)$$

$$(\!| \, a(\tilde{x}).P \, |\!) \triangleq \Pi(\tilde{x}).(\!| \, P \, |\!) \quad \texttt{with} \quad \Pi = (\texttt{Port}_a = a)$$

Remark 4.3. The attributes values in AbC can be modified by means of internal actions. Changing attributes values introduces opportunistic interactions between components in the sense that an attribute update means an opportunity of interaction. The is because selecting interaction partners depends on the predicate over the attributes they expose. Changing the values of these attributes implies changing the set of possible partners and this is why modeling adaptivity in AbC is quite natural. This possibility is missing in $b\pi$-calculus and all other channel-based process calculi since internal actions and the opportunity of interaction are orthogonal in their models.

We argue that finding a compositional encoding for the following simple behavior is very difficult if not impossible in channel-based process calculi.

$$\begin{array}{l} \Gamma_1 : (msg)@(\mathsf{tt}) \vdash_{\{b\}} \| \\ \qquad \Gamma_2 : [\texttt{this}.a := 5].P \mid (b \leq \texttt{this}.a)(x).Q \end{array}$$

Initially $\Gamma_1(b) = 3$ and $\Gamma_2(a) = 2$. Changing the value of the local attribute a by the left-hand side process in the second component provides an opportunity of receiving the message "msg" from the process residing in the first component.

The full encoding of $b\pi$-calculus into AbC is reported in Table VII. The formal definition which specifies what properties are preserved by this encoding and a proof sketch for the correctness of the encoding up to a specific behavioral equivalence will be presented in Section 5.3.

4.3 Encoding interaction patterns

In this section, we provide insights on how the concept of *attribute-based communication* can be exploited to provide a general unifying framework encompassing different interaction patterns tailored for multiway interactions. We show how group-based [Agha and Callsen 1993], [Chockler et al. 2001], [Holbrook and Cheriton 1999] and publish/subscribe-based [Eugster et al. 2003], [Bass and Nguyen 2002] interaction patterns can be naturally rendered in AbC. Since these interaction patterns do not have formal descriptions, we proceeds by relying on examples.

We start with group-based interaction patterns and show that when encoding a group name as an attribute in AbC, the constructs for joining or leaving a given group can be encoded as attribute updates, see the following example:

$$\begin{array}{l} \Gamma_1 : (msg)@(group = a) \vdash_{\{group\}} \| \\ \qquad \Gamma_2 : (group = b)(x) \mid [\texttt{this}.group := c] \| \\ \qquad \vdots \\ \qquad \Gamma_7 : (group = b)(x) \mid [\texttt{this}.group := a] \end{array}$$

initially $\Gamma_1(group) = b$, $\Gamma_2(group) = a$, and $\Gamma_7(group) = c$. Component 1 wants to send the message "msg" to group "a". Only Component 2 is allowed to receive it as it is the only member of group "a". Component 2 can leave group "a" and join "c" by performing an update action. On the other hand, if Component 7 joined group "a" before "msg" is emitted then both of Component 2 and Component 7 will receive the message.

It is worth mentioning that a possible encoding of group communication into $b\pi$-calculus has been introduced in [Ene and Muntean 2001]. The encoding is relatively complicated and does not guarantee the causal order

of message reception. "Locality" is neither a first class citizen in $b\pi$-calculus nor in AbC. However, "locality" (in this case, the group name) can be naturally encoded as an attribute in AbC while in $b\pi$-calculus, it needs much more efforts.

Publish/subscribe interaction patterns can be considered as special cases of the attribute-based ones. For instance, a natural encoding of the topic-based publish/subscribe model [Eugster et al. 2003] into AbC can be accomplished by allowing publishers to broadcast messages with "tt" predicates (i.e., satisfied by all) and only subscribers can check the compatibility of the exposed publishers attributes with their subscriptions, see the following example:

$$\begin{aligned}\Gamma_1 : (msg)@(\mathsf{tt}) \vdash_{\{topic\}} \ \| \\ \Gamma_2 : (topic = \texttt{this}.subscription)(x) \ \| \\ \vdots \\ \Gamma_n : (topic = \texttt{this}.subscription)(x) \ \|\end{aligned}$$

Apparently, the publisher broadcasts the message "msg" tagged with a specific topic for all possible subscribers (the predicate is satisfied by all) and the subscriber receives the message if the topic matches its subscription. For the sake of presentation, we abstract from the existence of a possible broker/mediator between the publishers and subscribers. However, a detailed model can be easily developed.

The dynamic settings of the attributes in AbC and the possibility of controlling their visibility during interactions are the main reasons why AbC enjoys a greater flexibility and expressive power.

5. BEHAVIORAL THEORY FOR *ABC*

This section follows a standard approach in concurrency in defining behavioral equivalences. We begin with a reduction barbed congruence and then we introduce an equal definition of labeled bisimulation for AbC. At the end of this section, we show a formal definition for the encoding presented in Section 4.2 and we sketch the proof of its correctness up to the strong reduction barbed congruence.

5.1 Reduction barbed congruence

In the behavioral theory, two terms are considered as equivalent if they cannot be distinguished by any external observer (i.e., they have the same observable behavior). For instance, in π-calculus both message transmission and reception are considered to be observable. However, this is not the case in AbC because of the non-blocking nature of the broadcast. Only message transmission can be observed.

Note that the transition $C \xrightarrow{\Gamma:(\tilde{v})@\Pi} C'$ does not guarantee that C has performed an input action but rather it means that C might have performed an input action. This is because that transition might happen due to the application of one of three different rules in Table VI: (**C-Rcv**) which guarantees reception and either (**C-FRcv**) or (**C-Fail**) which guarantee non-reception. Hence, input action cannot be observed by an external observer and the output action is the only observable in AbC. Following Milner and Sangiorgi [Milner and Sangiorgi 1992], we use the term "barb" as synonymous of observable. In what follows, we shall use the following notations:

— $\Rightarrow$ denotes $\xrightarrow{\tau}{}^*$
— $\overset{\gamma}{\Rightarrow}$ denotes $\Rightarrow\xrightarrow{\gamma}\Rightarrow$ if $(\gamma \neq \tau)$
— $\overset{\hat{\gamma}}{\Rightarrow}$ denotes $\Rightarrow$ if $(\gamma = \tau)$ and $\overset{\gamma}{\Rightarrow}$ otherwise.
— $\rightarrowtriangle$ denotes $\{\xrightarrow{\gamma} \mid \gamma \text{ is an output or } \gamma = \tau\}$
— $\rightarrowtriangle^*$ denotes$(\rightarrowtriangle)^*$

A context $\mathcal{C}[\bullet]$ is a component term with a hole, denoted by $[\bullet]$ and AbC conetxts are generated by the following grammar:

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

Definition 5.1 (*Barb*). Let $C{\downarrow_\Pi}$ mean that component C can broadcast a message with a predicate Π (i.e., $C \xrightarrow{\nu\tilde{x}\overline{\Gamma:(\tilde{v})@\Pi}}$ where $\Pi \neq \mathsf{ff}$). We write $C \Downarrow_\Pi$ if $C \twoheadrightarrow^* C' \downarrow_\Pi$.[1]

Definition 5.2 (*Barb Preservation*). $\mathcal{R}$ is barb-preserving iff for every $(C_1, C_2) \in \mathcal{R}$, $C_1{\downarrow_\Pi}$ implies $C_2 \Downarrow_\Pi$

Definition 5.3 (*Reduction Closure*). $\mathcal{R}$ is reduction-closed iff for every $(C_1, C_2) \in \mathcal{R}$, $C_1 \twoheadrightarrow C_1'$ implies $C_2 \twoheadrightarrow^* C_2'$ and $(C_1', C_2') \in \mathcal{R}$

Definition 5.4 (*Context Closure*). $\mathcal{R}$ is context-closed iff for every $(C_1, C_2) \in \mathcal{R}$ and for all contexts $\mathcal{C}[\bullet]$, $(\mathcal{C}[c_1], \mathcal{C}[c_2]) \in \mathcal{R}$

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

Definition 5.5 (*Weak Reduction Barbed Congruence*). A symmetric relation $\mathcal{R}$ over the set of AbC- components which is barb-preserving, reduction-closed, and context-closed.

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

Lemma 5.6. *if* $C_1 \cong C_2$ *then*

- $C_1 \Downarrow_\Pi$ *iff* $C_2 \Downarrow_\Pi$
- $C_1 \twoheadrightarrow^* C_1'$ *implies* $C_2 \twoheadrightarrow^* \cong C_1'$

Proof. The proof follows easily by definition. □

5.2 Bisimulation Proof Methods

In this section, we define an appropriate notion of bisimulation for AbC components. We prove that our labeled bisimilarity coincides with reduction barbed congruence, and thus represents a valid tool for proving that two components are reduction barbed congruent.

Definition 5.7 (*Weak Labelled Bisimulation*). A symmetric binary relation $\mathcal{R}$ over the set of AbC- components is a weak bisimulation if for every action γ, whenever $(C_1, C_2) \in \mathcal{R}$ and

- γ is of the form τ, $\Gamma:(\tilde{v})@\Pi$, or $(\nu\tilde{x}\overline{\Gamma:(\tilde{v})@\Pi}$ with $\Pi \neq \mathsf{ff})$, it holds that
  $C_1 \xrightarrow{\gamma} C_1'$ implies $C_2 \stackrel{\hat{\gamma}}{\Rightarrow} C_2'$ and $(C_1', C_2') \in \mathcal{R}$

Two components C_1 and C_2 are weak bisimilar, written $C_1 \approx C_2$ if there exists a weak bisimulation $\mathcal{R}$ relating them. Strong bisimilarity, "$\sim$", is defined in a similar way by replacing $\Rightarrow$ with $\rightarrow$.

It is easy to prove that $\sim$ and $\approx$ are equivalence relations by relying on the classical arguments of [Milner 1989]. However, our labeled bisimilarity enjoys a much more interesting property: the closure under any context. So, in the next two lemmas, we prove that our labeled bisimilarity is preserved by name restriction and parallel composition.

[1] From now on, we use the predicate Π to denote only its meaning, not its syntax.

Lemma 5.8 ($\sim$ and $\approx$ are preserved by parallel composition). *Let C_1 and C_2 be two components such that:*

— *$C_1 \sim C_2$ implies $C_1 \| C \ \sim \ C_2 \| C$ for all components C.*
— *$C_1 \approx C_2$ implies $C_1 \| C \ \approx \ C_2 \| C$ for all components C.*

Proof. It is sufficient to prove that the relation $\mathcal{R} = \{(C_1 \| C, C_2 \| C) |$ for all C such that $(C_1 \sim (\approx)\ C_2)\}$ is a bisimulation. See Appendix B, Page 23. □

Lemma 5.9 ($\sim$ and $\approx$ are preserved by name restriction). *Let C_1 and C_2 be two components such that:*

— *$C_1 \sim C_2$ implies $\nu x C_1 \ \sim \ \nu x C_2$ for all names x.*
— *$C_1 \approx C_2$ implies $\nu x C_1 \ \approx \ \nu x C_2$ for all names x.*

Proof. it is sufficient to prove that the relation $\mathcal{R} = \{(C, B) |\ C = \nu x C_1,\ B = \nu x C_2$ with $(C_1 \sim (\approx)\ C_2)\}$ is a bisimulation. See Appendix B, Page 24. □

As an immediate results of Lemma 5.8 and Lemma 5.9, $\sim$ and $\approx$ are congruence relations (i.e., closed under any context).

Now, we can show that our label bisimilarity represents a proof technique for reduction barbed congruence.

Theorem 5.10 (Soundness). *Let C_1 and C_2 be two components such that:*

— *$C_1 \sim C_2$ implies $C_1 \ \simeq \ C_2$.*
— *$C_1 \approx C_2$ implies $C_1 \ \cong \ C_2$.*

Proof. (we only prove the weak case)
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 $C_1 \xrightarrow{\nu \tilde{x} \overline{\Gamma : (\tilde{v}) @ \Pi}}$ for an output label $\nu \tilde{x} \overline{\Gamma : (\tilde{v}) @ \Pi}$ with $\Pi \neq \mathsf{ff}$. As $(C_1 \approx C_2)$, We have that also $C_2 \xRightarrow{\nu \tilde{x} \overline{\Gamma : (\tilde{v}) @ \Pi}}$ and $C_2 \Downarrow_\Pi$.
— (Reduction-closure): $C_1 \twoheadrightarrow C_1'$ means that either $C_1 \xrightarrow{\tau} C_1'$ or $C_1 \xrightarrow{\nu \tilde{x} \overline{\Gamma : (\tilde{v}) @ \Pi}} C_1'$. As $(C_1 \approx C_2)$, then there exists C_2' such that either $C_2 \Rightarrow C_2'$ or $C_2 \xRightarrow{\nu \tilde{x} \overline{\Gamma : (\tilde{v}) @ \Pi}} C_2'$ with $(C_1' \approx C_2')$. So $C_2 \twoheadrightarrow^* C_2'$.
— (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 5.8 and Lemma 5.9, We have that $\mathcal{C}[c_1] \approx \mathcal{C}[c_2]$.

In conclusion, We have that $(C_1 \cong C_2)$ as required. □

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

Lemma 5.11 (Completeness). *Let C_1 and C_2 be two components such that:*

— *$C_1 \simeq C_2$ implies $C_1 \ \sim \ C_2$.*
— *$C_1 \cong C_2$ implies $C_1 \ \approx \ C_2$.*

Proof. It is sufficient to prove that the relation $\mathcal{R} = \{(C_1, C_2) \mid$ such that $(C_1 \simeq (\cong)\ C_2)\}$ is a bisimulation. See Appendix B, Page 24. □

As a direct consequence of Theorem 5.10 and Lemma 5.11, we have that bisimilarity and reduction barbed congruence coincide.

Theorem 5.12 (Characterization). *Bisimilarity and reduction barbed congruence coincide.*

5.3 The correctness of encoding

In this section, we provide a formal definition and a proof sketch for the encoding (translation) presented in Section 4.2. We begin with the properties that our encoding preserves. Basically, when translating a term from $b\pi$-calculus into AbC, we would like the translation: to be compositional in the sense that it is independent from contexts; to be independent from the names of the source term (i.e., name invariance); to preserve the parallel composition (i.e., Homomorphic w.r.t. '|'); to be faithful in the sense it preserves the observable behavior (i.e., barbs) and reflects divergence; to translate output (input) action in $b\pi$-calculus into a corresponding output (input) in AbC, and finally the translation should 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 mimked 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 5.13 (*Divergence*). P diverges, written $P \Uparrow$, iff $P \rightarrowtriangle^{\omega}$ where ω denotes an infinite number of reductions.

Definition 5.14 (*Uniform Encoding*). An encoding $(\!|\ \cdot\ |\!) : \mathcal{L}_1 \rightarrow \mathcal{L}_2$ is uniform if it enjoys the following properties:

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

Lemma 5.15 (Operational Completeness). *if* $P \rightarrowtriangle_{b\pi} P'$ *then* $(\!|\ P\ |\!) \rightarrowtriangle^* \simeq (\!|\ P'\ |\!)$ *where* $\simeq$ *is the strong barbed equivalence of* AbC.

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: $P_1 \| P_2 \xrightarrow{\nu\tilde{y}\tilde{a}\tilde{x}} P_1' \| P_2'$.

By applying induction hypotheses on the premises $P_1 \xrightarrow{\nu\tilde{y}\tilde{a}\tilde{x}} P_1'$ and $P_2 \xrightarrow{a(\tilde{x})} P_2'$, we have that $(\!|\ P_1\ |\!) \rightarrowtriangle^* \simeq (\!|\ P_1'\ |\!)$ and $(\!|\ P_2\ |\!) \rightarrowtriangle^* \simeq (\!|\ P_2'\ |\!)$. We can apply (C-Brd) if $\tilde{y} = \emptyset$ or (LOpen) otherwise. We consider (C-Brd) only and (LOpen) follows in the same way.

$$\dfrac{\dfrac{(\!|\ P_1\ |\!) \xmapsto{\overline{\Gamma':(\tilde{v})@\Pi}}_{\Gamma} (\!|\ P_1'\ |\!) \qquad (\!|\ P_2\ |\!) \xmapsto{\Gamma':(\tilde{v})@\Pi}_{\Gamma} (\!|\ P_2'\ |\!)}{(\!|\ P_1\ |\!)|(\!|\ P_2\ |\!) \xmapsto{\overline{\Gamma':(\tilde{v})@\Pi}}_{\Gamma} (\!|\ P_1'\ |\!)|(\!|\ P_2'\ |\!)}}{\Gamma : (\!|\ P_1\ |\!)|(\!|\ P_2\ |\!) \xrightarrow{\overline{\Gamma':(\tilde{v})@\Pi}} \Gamma : (\!|\ P_1'\ |\!)|(\!|\ P_2'\ |\!)}$$

where $\Pi = (port_a = a)$ and $\Gamma' = \{(port_a, a)\}$ and we have that:
$(\!|\ P_1' \| P_2'\ |\!)_{\Gamma} \simeq \Gamma : (\!|\ P_1'\ |\!)|(\!|\ P_2'\ |\!)$ as required. See full proof at Appendix C, Page 25. □

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 5.16 (STRONG COMPLETENESS). *if* $P \twoheadrightarrow_{b\pi} P'$ *then* $\exists Q$ *such that* $Q \equiv (\!|P'|\!)$ *and* $(\!|P|\!) \twoheadrightarrow_{\text{AbC}} Q$.

COROLLARY 5.17 (STRONG SOUNDNESS). *if* $(\!|\ P\ |\!) \twoheadrightarrow_{\text{AbC}} Q$ *then* $Q \equiv (\!|\ P'\ |\!)$ *and* $P \twoheadrightarrow_{b\pi} P'$

THEOREM 5.18. *The encoding* $(\!|\ \centerdot\ |\!) : b\pi \to \text{AbC}$ *is uniform.*

PROOF. Definition 5.14(1) and 5.14(2) hold by construction. Definition 5.14(4) holds by Lemma 5.15, Corollary 5.16, and Corollary 5.17 respectively. Definition 5.14(3) holds easily and as a result of the strong formulation of operational correspondence in Corollary 5.16, and Corollary 5.17, this encoding preserves the observable behavior and cannot introduce divergence. □

6. DISCUSSION AND RELATED WORK

In this section, we discuss the main differences between the version of AbC presented in this paper and the one of [Abd Alrahman et al. 2015], then we touch on related works concerning calculi with primitives that permit selecting/restricting communication partners.

The main aim of the new version of AbC is to overcome some of the limitations of the original one and enrich the calculus with new primitives to effectively control interactions in an attribute-based fashion. Indeed, the new calculus is much more richer and we can say that only the basic idea of attribute-based communication survived from the old one. Even the operational semantic model is different, the new AbC is equipped with a labeled operational semantics rather than just with a reduction based one. The main advantage of the labeled semantics is that it can be used to provide a compositional account of systems and thus to study the interaction of different components in different contexts. The reduction semantics, instead, can only be used to describe the overall system evolution, and not to study the impact of a given component on the rest.

The most notable novelty of the current version of AbC calculus is that components can specify the visibility of attributes during interaction. As opposed to the previous version where components attach the whole attribute environment alongside the broadcasted message, in the current version, a component can decide the attributes to be exposed. This new feature enhances the expressive power of AbC, as already shown in Section 4, and also helps in avoiding improper behaviors. Consider the following example:

(**Old *AbC***)

$$\Gamma_1{:}(v_1)@(port_1 = a) + (v_2)@(port_2 = b) \parallel \Gamma_2{:}(port_1 = a)(x) + (port_2 = b)(x)$$

(**New *AbC***)

$$\Gamma_1{:}(v_1)@(port_1 = a) \vdash_{\{port_1\}} + (v_2)@(port_2 = b) \vdash_{\{port_2\}} \parallel \Gamma_2{:}(port_1 = a)(x) + (port_2 = b)(x)$$

where $\Gamma_1 = \Gamma_2 = \{(port_1, a),\ (port_2, b)\}$. Apparently, the receiving component on right-hand side of "$\parallel$" is waiting to receive two different messages, one from a component satisfying $(port_1 = a)$ and the other from a component satisfying $(port_2 = b)$. However, in the previous version of AbC since the whole Γ_1 is exposed in the interaction, the message v_1 can be received by any of the two alternative subprocesses. On the other hand, this behavior can be avoided in the new version of AbC since components can decide what attribute to expose. So in this example, they either expose "$port_1$" or "$port_2$".

Other differences are that the new version has a polyadic (i.e., multi-valued) instead of a monadic communication, name restriction operators both at the component and process levels, and it supports multi-threaded components through a parallel composition operator at the process level. Moreover the new AbC has specific operators ($this.-$ and $match$) for guaranteeing *self-awareness* and *context-awareness*, and its predicate language is richer and permits considering the sent values when selecting partners.

As already mentioned in the introduction, the main source of inspiration of AbC has been the SCEL language [De Nicola et al. 2013], [De Nicola et al. 2014] that had been introduced to handle the challenges posed by the design of ensembles [Sanders and Smith 2008] of autonomic components. In SCEL, autonomic components are equipped with dedicated knowledge repositories, and different components cooperate by storing and retrieving information about themselves and their environment. Each component is equipped with an interface where its attributes are published and predicates over attributes are used to dynamically specify the communication partners.

Clearly there are many other calculi that aim at providing tools for specifying and reasoning about communicating systems, here we would like to touch only on those tailored for group communications while identifying the ones enjoying specific properties.

CBS [Prasad 1991], [Prasad 1995], [Ostrovsky et al. 2002] is probably the first process calculus to rely on broadcast rather than on channel-based communication. It 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. The CPC calculus [Given-Wilson et al. 2010] 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 [John et al. 2008], [John et al. 2010] 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 [John et al. 2009] 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 [Vigo et al. 2013] 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.

All the above mentioned calculi, just like $b\pi$-calculus [Ene and Muntean 1999], [Ene and Muntean 2001] considered in Section 4.2, can be modelled in AbC; the advantage of the latter is that the dynamic setting of the attributes and the possibility of hiding some of them during specific interactions offers greater flexibility and expressiveness.

7. CONCLUDING REMARKS

We have introduced an extended and polyadic version of AbC calculus for attribute-based communication initially proposed in [Abd Alrahman et al. 2015]. Apart from the primitives of attribute-based input and output, the extended version includes: name restriction, multithreading, a match operator, a rich language for predicates, and the ability to specify which attributes to be exposed during interactions. We have investigated the expressive power of AbC both in terms of its ability to model scenarios featuring collaboration, reconfiguration, and adaptation and of its ability to encode a powerful process calculus for channel-based communication and other interaction paradigms. We have defined behavioral equivalences for AbC and finally we proved the correctness of the proposed encoding up to some reasonable equivalence.

We plan to investigate the impact of bisimulation in terms of axioms, proof techniques, etc. and to consider also alternative behavioral relations like testing preorders. Further attention will be also dedicated to the actual implementation of the different linguistic abstractions that we have introduced. We also plan to define a full-fledged language based on AbC operators and to test its effectiveness not only as a tool for encoding calculi but also for dealing with case studies from different application domains.

ACKNOWLEDGMENT

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

APPENDIX

A. SECTION 3.1 PROOFS

PROOF OF LEMMA 3.1. The "only if" implication is straightforward. For the "if" implication, the proof is carried out by induction on the structure of Π.

—if $(\Pi = \mathsf{tt})$: according to Table V, $(\mathsf{tt}\blacktriangleright x = \mathsf{tt})$ which means that the satisfaction of tt does not depend on x (i.e., $\Gamma \models \mathsf{tt}\blacktriangleright x$ iff $\Gamma \models \mathsf{tt}$). From Table I (b), 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 V, $(\mathsf{ff}\blacktriangleright x = \mathsf{ff})$ which again means that the satisfaction of ff does not depend on x. From Table I (b), We have that ff is not satisfied by any Γ, so this case holds vacuously.

—if $(\Pi = (a = m)\blacktriangleright x)$: according to Table V, We have two cases:
  —if $(x = m)$ then $\Pi = \mathsf{ff}$ and by induction hypotheses, the case holds vacuously.
  —if $(x \neq m)$ then $\Pi = (x = m)$, according to Table I (b), 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 V, $(\Pi_1 \wedge \Pi_2)\blacktriangleright x = (\Pi_1\blacktriangleright x \wedge \Pi_2\blacktriangleright x)$. From Table I (b), 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 V, $(\neg\Pi)\blacktriangleright x = \neg(\Pi\blacktriangleright x)$. From Table I (b), 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.

and this concludes the proof. □

B. SECTION 5.2 PROOFS

The following Lemma is useful to prove that a component with a restricted name does not need any renaming when performing a τ action. We will use it in the proof of Lemma 4

LEMMA B.1. *$C[y/x] \Rightarrow C'$ implies $\nu x C \Rightarrow \nu y C'$*

PROOF. The proof proceeds by induction on the length of the derivation $\Rightarrow_n$

—Base Case: $n = 0$
$C[y/x] \equiv_\alpha C'$ which implies $\nu x C \equiv_\alpha \nu y C[y/x]$

—For all $k \leq n$: $C[y/x] \Rightarrow_k C'$ implies $\nu x C \Rightarrow_k \nu y C'$
if $C[y/x] \Rightarrow_{n+1} C'$, then we have that $C[y/x] \Rightarrow_n C'' \xrightarrow{\tau} C'$
This implies that $\nu x C \Rightarrow_n \nu y C''$ and $C'' \xrightarrow{\tau} C'$ which means that $\nu y C'' \xrightarrow{\tau} \nu y C'$.
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 $\nu y C'' \xrightarrow{\tau} \nu y C'$ and we have that $\nu x C \Rightarrow \nu y C'$ as required.

and this concludes the proof. □

Proof of Lemma 5.8. (we only prove the weak case)
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 applied rule to derive the transition $C_1\|C \xrightarrow{\gamma} \hat{C}$, we have several cases.

—$C_1\|C \xrightarrow{\tau} \hat{C}$, then the last applied rule is (τ-Int) or its symmetry.
  —if (τ-Int) 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 (τ-Int) several times, we have that $C_2\|C \Rightarrow C_2'\|C$ and $(C_1'\|C, C_2'\|C) \in \mathcal{R}$
  —if the symmetry of (τ-Int) 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 $(C_1\|C', C_2\|C') \in \mathcal{R}$

—$C_1\|C \xrightarrow{\nu\tilde{x}\overline{\Gamma:(\tilde{v})@\Pi}} \hat{C}$ with $\hat{x} \cap fn(C) = \emptyset$ and $\Pi \neq \mathsf{ff}$, then the last applied rule is (Com) or its symmetry.
  —if (Com) is applied then $\hat{C} = C_1'\|C'$, $C_1 \xrightarrow{\nu\tilde{x}\overline{\Gamma:(\tilde{v})@\Pi}} C_1'$ and $C \xrightarrow{\Gamma:(\tilde{v})@\Pi} C'$. Since $C_1 \approx C_2$ then there exists C_2' such that $C_2 \xRightarrow{\nu\tilde{x}\overline{\Gamma:(\tilde{v})@\Pi}} C_2'$ and $(C_1' \approx C_2')$. By an application of (Com) and several application of (τ-Int), we have that $C_2\|C \xRightarrow{\nu\tilde{x}\overline{\Gamma:(\tilde{v})@\Pi}} C_2'\|C'$ and $(C_1'\|C', C_2'\|C') \in \mathcal{R}$
  —if the symmetry of (Com) is applied then $\hat{C} = C_1'\|C'$, $C_1 \xrightarrow{\Gamma:(\tilde{v})@\Pi} C_1'$ and $C \xrightarrow{\nu\tilde{x}\overline{\Gamma:(\tilde{v})@\Pi}} C'$. So it is immediate to have that $C_2\|C \xRightarrow{\nu\tilde{x}\overline{\Gamma:(\tilde{v})@\Pi}} C_2'\|C'$ and $(C_1'\|C', C_2'\|C') \in \mathcal{R}$

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

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

Proof of Lemma 5.9. (we only prove the weak case)
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 $C \xrightarrow{\gamma} \hat{C}$.

—if $(\gamma = \tau)$ then only rule (Res) is applied. if (Res) is applied, then $C_1[x/x] \xrightarrow{\tau} C_1'$ and $\hat{C} = \nu x C_1'$. As $(C_1 \approx C_2)$, We have that $C_2[x/x] \Rightarrow C_2'$ with $(C_1' \approx C_2')$. By Lemma B.1 and several applications of (Res), we have that $B \Rightarrow \nu x C_2'$ and $(\nu x C_1', \nu x C_2') \in \mathcal{R}$.

—if $(\gamma = \nu\tilde{y}\overline{\Gamma:(\tilde{v})@\Pi})$ then either rule (EOpen), (Res), (EHide1) or (EHide2) is applied.
  —if (EOpen) is applied, then $x \in (\tilde{v} \cup \tilde{y})\backslash n(\Pi)$ and $C_1[z/x] \xrightarrow{\overline{\Gamma:(\tilde{v})@\Pi}} C_1'$ with $\hat{C} = C_1'$. As $(C_1 \approx C_2)$, We have that $C_2[z/x] \xRightarrow{\overline{\Gamma:(\tilde{v})@\Pi}} C_2'$ with $(C_1' \approx C_2')$. By Lemma B.1, an application of (EOpen), and several applications of (Res), we have that $B \xRightarrow{\nu\tilde{y}\overline{\Gamma:(\tilde{v})@\Pi}} C_2'$ and $(C_1', C_2') \in \mathcal{R}$.
  —if (Res) is applied, then $C_1[z/x] \xrightarrow{\nu\tilde{y}\overline{\Gamma:(\tilde{v})@\Pi}} C_1'$ and $\hat{C} = \nu z C_1'$. As $(C_1 \approx C_2)$, We have that $C_2[z/x] \xRightarrow{\nu\tilde{y}\overline{\Gamma:(\tilde{v})@\Pi}} C_2'$ with $(C_1' \approx C_2')$. By Lemma B.1 and several applications of (Res), we have that $B \xRightarrow{\nu\tilde{y}\overline{\Gamma:(\tilde{v})@\Pi}} \nu z C_2'$ and $(\nu z C_1', \nu z C_2') \in \mathcal{R}$
  —if (EHide1) is applied, then $C_1 \xrightarrow{\nu\tilde{y}\overline{\Gamma:(\tilde{v})@\Pi}} C_1'$ and $\hat{C} = \nu x \nu\tilde{y} C_1'$. As $(C_1 \approx C_2)$, We have that $C_2 \xRightarrow{\nu\tilde{y}\overline{\Gamma:(\tilde{v})@\Pi}} C_2'$ with $(C_1' \approx C_2')$. By Lemma B.1, an application of (EHide1), and several applications of (Res), we have that $B \xRightarrow{\nu\tilde{y}\overline{\Gamma:(\tilde{v})@\mathsf{ff}}} \nu x \nu\tilde{y} C_2'$ and $(\nu x \nu\tilde{y} C_1', \nu x \nu\tilde{y} C_2') \in \mathcal{R}$

—if (EHide2) is applied, then $C_1 \xrightarrow{\nu\tilde{y}\overline{\Gamma:(\tilde{v})@\Pi}} C_1'$ and $\hat{C} = \nu x C_1'$. As $(C_1 \approx C_2)$, We have that $C_2 \xRightarrow{\nu\tilde{y}\overline{\Gamma:(\tilde{v})@\Pi}} C_2'$ with $(C_1' \approx C_2')$. By Lemma B.1, an application of (EHide2), and several applications of (Res), we have that $B \xRightarrow{\nu\tilde{y}\overline{\Gamma:(\tilde{v})@\Pi \blacktriangleright x}} \nu x C_2'$ and $(\nu x C_1', \nu x C_2') \in \mathcal{R}$

—if $(\gamma = \Gamma:(\tilde{v})@\Pi)$ then $x \notin n(\gamma)$ and only rule (Res) is applied. So we have that $C_1[y/x] \xrightarrow{\Gamma:(\tilde{v})@\Pi} C_1'$ and $\hat{C} = \nu y C_1'$. As $(C_1 \approx C_2)$, We have that $C_2[y/x] \xRightarrow{\Gamma:(\tilde{v})@\Pi} C_2'$ with $(C_1' \approx C_2')$. By Lemma B.1 and several applications of (Res), we have that $B \xRightarrow{\Gamma:(\tilde{v})@\Pi} \nu y C_2'$ 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 5.11. (we only prove the weak case)
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 $C_1 \xrightarrow{\nu\tilde{x}\overline{\Gamma:(\tilde{v})@\Pi}} C_1'$ for any Γ, Π and a sequence of values $\tilde{v}$ where $\Pi \neq \mathsf{ff}$. We build up a context to mimic the effect of this action.
   Our context has the following form:

$$\mathcal{C}[\bullet] \triangleq [\bullet] \parallel \prod_{i\in I} \Gamma_i : \Pi_i(\tilde{x}_i).\langle \tilde{x}_i = \tilde{v}\rangle(\tilde{x}_i)@(in = a) \vdash_{\{in\}} \parallel \prod_{j\in J} \Gamma_j : (in = a)(\tilde{x}_j).(\tilde{x}_j)@(out = b) \vdash_{\{out\}}$$

   where $I \cap J = \emptyset$, $\Gamma_i|_{\{in\}} \models (in = a)$, $\Gamma_j \models (in = a)$, and the names a and b are fresh. Π_i is an arbitrary predicate. 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 $\prod_{i\in I} \Gamma_i : P_i$ to denote the parallel composition of all components $\Gamma_i : P_i$, for $i \in I$. Now assume that $(\Gamma_i \models \Pi)$ and $(\Gamma \models \Pi_i)$. We let Π_i to range over any predicate that is satisfied by Γ (i.e., the exposed environment in the broadcast of C_1). Intuitively, the existence of a barb on $(in = a)$ indicates that the action has not yet happened, whereas the presence of a barb on $(out = b)$ together with the absence of the barb on $(in = a)$ 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 $C_1 \xrightarrow{\nu\tilde{x}\overline{\Gamma:(\tilde{v})@\Pi}} C_1'$, it follows by Lemma 2 that:

$$\mathcal{C}[C_1] \;\Rightarrow\; C_1' \parallel \prod_{i\in I} \Gamma_i : 0 \parallel \prod_{j\in J} \Gamma_j : (\tilde{v})@(out = b) \vdash_{\{out\}} \;=\; \hat{C}_1$$

   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 $\mathcal{C}[C_2] \Rightarrow \hat{C}_2 \cong \hat{C}_1$ with $\hat{C}_2 \not\Downarrow_{(in=a)}$ and $\hat{C}_2 \Downarrow_{(out=b)}$. The conditions on the barbs allow us to get the structure of the above reduction sequence as follows:

$$\mathcal{C}[C_2] \;\Rightarrow\; C_2' \parallel \prod_{i\in I} \Gamma_i : 0 \parallel \prod_{j\in J} \Gamma_j : (\tilde{v})@(out = b) \vdash_{\{out\}} \;\cong\; \hat{C}_1$$

   This implies that $C_2 \xRightarrow{\nu\tilde{x}\overline{\Gamma:(\tilde{v})@\Pi}} C_2'$. 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 $C_1' \cong C_2'$ as required.

2. Suppose that $C_1 \xrightarrow{\Gamma:(\tilde{v})@\Pi} C_1'$ for any Γ, Π and a sequence of values $\tilde{v}$. We build up the following context to mimic the effect of this action.

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

where $\Gamma = \Gamma'|_{\{in\}}$, $\Pi = (in = a)$, 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 $C_1 \xrightarrow{\Gamma:(\tilde{v})@\Pi} C_1'$, it follows by Lemma 2 that:

$$\mathcal{C}[C_1] \;\Rightarrow\; C_1' \parallel (\tilde{v})@(out = b) \vdash_{\{out\}} = \hat{C_1}$$

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 $\mathcal{C}[C_2] \;\Rightarrow\; \hat{C_2} \cong \hat{C_1}$ with $\hat{C_2} \not\Downarrow_{(in=a)}$ and $\hat{C_2} \Downarrow_{(out=b)}$ as follows:

$$\mathcal{C}[C_2] \;\Rightarrow\; C_2' \parallel (\tilde{v})@(out = b) \vdash_{\{out\}} \cong \hat{C_1}$$

This implies that $C_2 \xRightarrow{\Gamma:(\tilde{v})@\Pi} C_2'$. 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 $C_1' \cong C_2'$ as required.

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

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

C. SECTION 5.3 PROOFS

Proof of Lemma 5.15. 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 $P \triangleq \texttt{nil}$: This case is immediate $(\!|\ \texttt{nil}\ |\!)_\Gamma \triangleq \Gamma : 0$

—if $P \triangleq \tau.Q$: We have that $\tau.Q \xrightarrow{\tau} Q$ and it is translated to $(\!|\ \tau.P\ |\!)_\Gamma \triangleq \Gamma : [Port_\tau := \tau].(\!|\ Q\ |\!)$. We only can apply rule (C-Upd) to mimic this transition.

$$\frac{[Port_\tau := \tau].(\!|\ Q\ |\!) \xmapsto{[Port_\tau := \tau]}_\Gamma (\!|\ Q\ |\!)}{\Gamma : [Port_\tau := \tau].(\!|\ Q\ |\!) \xrightarrow{\tau} \Gamma[Port_\tau \mapsto \tau] : (\!|\ Q\ |\!)}$$

Now it is not hard to see that $(\!|\ Q\ |\!)_\Gamma \;\simeq\; \Gamma[Port_\tau \mapsto \tau] : (\!|\ Q\ |\!)$. They are even structural congruent.

—if $P \triangleq a(\tilde{x}).Q$: We have that $a(\tilde{x}).Q \xrightarrow{a(\tilde{z})} Q[\tilde{z}/\tilde{x}]$ and it is translated to $(\!|\ a(\tilde{x}).Q\ |\!)_\Gamma \triangleq \Gamma : (\texttt{Port}_a = a)(\tilde{x}).(\!|\ Q\ |\!)$. We only can apply rule (C-Rcv) to mimic this transition.

$$\frac{(\texttt{Port}_a = a)(\tilde{x}).(\!|\ Q\ |\!) \xmapsto{\Gamma':(\tilde{z})@\Pi}_\Gamma (\!|\ Q\ |\!)[\tilde{z}/\tilde{x}] \quad (\Gamma \models \Pi\ )}{\Gamma : (\texttt{Port}_a = a)(\tilde{x}).(\!|\ Q\ |\!) \xrightarrow{\Gamma':(\tilde{z})@\Pi} \Gamma : (\!|\ Q\ |\!)[\tilde{z}/\tilde{x}]}$$

where $\Pi = (\texttt{Port}_a = a)$ and $\Gamma' = \{(port_a, a)\}$ and again, it's not hard to see that:
$(\!|\ Q[\tilde{z}/\tilde{x}]\ |\!)_\Gamma \;\simeq\; \Gamma : (\!|\ Q\ |\!)[\tilde{z}/\tilde{x}]$.

—if $P \triangleq \bar{a}\tilde{x}.Q$: This is analogous to the previous case but with applying (C-Brd) instead.

—The fail rules for **nil**, τ, input and output are proved in a similar way but with applying (C-Fail) instead.

—if $P \triangleq \nu x Q$: We have that either $\nu x Q \xrightarrow{\gamma} \nu x Q'$, $\nu x Q \xrightarrow{\tau} \nu x \nu \tilde{y} Q'$ or $\nu x Q \xrightarrow{\nu x \nu \tilde{y} \bar{a} \tilde{z}} Q'$ and it is translated to $(\!|\ \nu x Q\ |\!)_\Gamma \triangleq \Gamma : \texttt{new}(x).(\!|\ Q\ |\!)$. We prove each case independently.

—Case $\nu x Q \xrightarrow{\gamma} \nu x Q'$: By applying induction hypotheses on the premise $Q \xrightarrow{\gamma} Q'$, we have that $(\!|\ Q\ |\!) \rightarrowtriangle^* \simeq (\!|\ Q'\ |\!)$. We can use either (C-Brd), (C-Rcv), or (C-Upd) to mimic transition depending on the performed action. We only consider (C-Brd) and the other cases follow in the same way.

$$\dfrac{\dfrac{(\!|\ Q\ |\!)[y/x] \overset{\overline{\Gamma':(\tilde{v})@\Pi}}{\longmapsto}_\Gamma (\!|\ Q'\ |\!)[y/x]}{\texttt{new}(x)(\!|\ Q\ |\!) \overset{\overline{\Gamma':(\tilde{v})@\Pi}}{\longmapsto}_\Gamma \texttt{new}(y)(\!|\ Q'\ |\!)[y/x]}}{\Gamma:\texttt{new}(x)(\!|\ Q\ |\!) \xrightarrow{\overline{\Gamma':(\tilde{v})@\Pi}} \Gamma:\texttt{new}(y)(\!|\ Q'\ |\!)[y/x]}$$

And We have that $(\!|\ \nu x Q'\ |\!)_\Gamma \simeq \Gamma:\texttt{new}(y)(\!|\ Q'\ |\!)[y/x]$ as required.

—Case $\nu x Q \xrightarrow{\tau} \nu x \nu \tilde{y} Q'$: By applying induction hypotheses on the premise $Q \xrightarrow{\nu \tilde{y} \nu \tilde{x} z} Q'$, we have that $(\!|\ Q\ |\!) \rightarrowtriangle^* \simeq (\!|\ Q'\ |\!)$. We only can use (C-tau) to mimic transition depending on the performed action.

$$\dfrac{\dfrac{\dfrac{(\!|\ Q\ |\!) \overset{\nu\tilde{y}\overline{\Gamma':(\tilde{v})@\Pi}}{\longmapsto}_\Gamma (\!|\ Q'\ |\!)}{\texttt{new}(x)(\!|\ Q\ |\!) \overset{\nu\tilde{y}\overline{\Gamma':(\tilde{v})@\texttt{ff}}}{\longmapsto}_\Gamma \texttt{new}(x)\texttt{new}(\tilde{y})(\!|\ Q'\ |\!)}}{\Gamma:\texttt{new}(x)(\!|\ Q\ |\!) \xrightarrow{\nu\tilde{y}\overline{\Gamma':(\tilde{v})@\texttt{ff}}} \Gamma:\texttt{new}(x)\texttt{new}(\tilde{y})(\!|\ Q'\ |\!)}}{\Gamma:\texttt{new}(x)(\!|\ Q\ |\!) \xrightarrow{\tau} \Gamma:\texttt{new}(x)\texttt{new}(\tilde{y})(\!|\ Q'\ |\!)}$$

where $\Pi = (port_x = x)$ and $\Gamma' = \{(port_x = x)\}$. We have that $(\!|\ \nu x \nu \tilde{y} Q'\ |\!)_\Gamma \simeq \Gamma:\texttt{new}(x)\texttt{new}(\tilde{y})(\!|\ Q'\ |\!)$ as required.

—Case $\nu x Q \xrightarrow{\nu x \nu \tilde{y} \bar{a} \tilde{z}} Q'$: follows in a similar way.

—Case $\nu x Q \xrightarrow{\alpha:}$: follows in a similar way.

—if $P \triangleq ((rec\ A\langle\tilde{x}\rangle).P)\langle\tilde{y}\rangle)$: This case is trivial.

—if $P \triangleq P_1 + P_2$: We have that either $P_1 + P_2 \xrightarrow{\alpha} P_1'$ or $P_1 + P_2 \xrightarrow{\alpha} P_2'$. We only consider the first case with $P_1 \xrightarrow{\alpha} P_1'$ and the other case follows in a similar way. This process is translated to $(\!|\ P_1 + P_2\ |\!)_\Gamma \triangleq \Gamma : (\!|\ P_1\ |\!) + (\!|\ P_2\ |\!)$. By applying induction hypotheses on the premise $P_1 \xrightarrow{\alpha} P_1'$, we have that $(\!|\ P_1\ |\!) \rightarrowtriangle^* \simeq (\!|\ P_1'\ |\!)$. We cn apply either (C-Brd), (C-Rcv), (C-Upd), or (LOpen) to mimic this transition depending on the performed action. We consider the case of (C-Brd) only and the other cases follow in a similar way.

$$\dfrac{\dfrac{(\!|\ P_1\ |\!) \overset{\overline{\Gamma':(\tilde{v})@\Pi}}{\longmapsto}_\Gamma (\!|\ P_1'\ |\!)}{(\!|\ P_1\ |\!) + (\!|\ P_2\ |\!) \overset{\overline{\Gamma':(\tilde{v})@\Pi}}{\longmapsto}_\Gamma (\!|\ P_1'\ |\!)}}{\Gamma:(\!|\ P_1\ |\!) + (\!|\ P_2\ |\!) \xrightarrow{\overline{\Gamma':(\tilde{v})@\Pi}} \Gamma:(\!|\ P_1'\ |\!)}$$

Again $(\!|\ P_1'\ |\!)_\Gamma \simeq \Gamma : (\!|\ P_1'\ |\!)$

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

—$P_1 \| P_2 \xrightarrow{\nu \tilde{y} \bar{a} \tilde{x}} P_1' \| P_2'$: 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 $(\!|\ P_1\ |\!) \rightarrowtriangle^* \simeq (\!|\ P_1'\ |\!)$ and $(\!|\ P_2\ |\!) \rightarrowtriangle^* \simeq (\!|\ P_2'\ |\!)$. We can apply (C-Brd) if $\tilde{y} = \emptyset$ or (LOpen) otherwise. We consider (C-Brd) only and (LOpen) follows in the same way.

$$\dfrac{\dfrac{(\!|\ P_1\ |\!) \overset{\overline{\Gamma':(\tilde{v})@\Pi}}{\longmapsto}_{\Gamma} (\!|\ P_1'\ |\!) \qquad (\!|\ P_2\ |\!) \overset{\Gamma':(\tilde{v})@\Pi}{\longmapsto}_{\Gamma} (\!|\ P_2'\ |\!)}{(\!|\ P_1\ |\!)|(\!|\ P_2\ |\!) \overset{\overline{\Gamma':(\tilde{v})@\Pi}}{\longmapsto}_{\Gamma} (\!|\ P_1'\ |\!)|(\!|\ P_2'\ |\!)}}{\Gamma : (\!|\ P_1\ |\!)|(\!|\ P_2\ |\!) \xrightarrow{\overline{\Gamma':(\tilde{v})@\Pi}} \Gamma : (\!|\ P_1'\ |\!)|(\!|\ P_2'\ |\!)}$$

where $\Pi = (port_a = a)$ and $\Gamma' = \{(port_a, a)\}$ and again we have that:
$(\!|\ P_1' \| P_2'\ |\!)_{\Gamma} \simeq \Gamma : (\!|\ P_1'\ |\!)|(\!|\ P_2'\ |\!)$

—$P_1 \| P_2 \xrightarrow{a(\tilde{x})} P_1' \| P_2'$: 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 $(\!|\ P_1\ |\!) \twoheadrightarrow^* \simeq (\!|\ P_1'\ |\!)$ and $(\!|\ P_2\ |\!) \twoheadrightarrow^* \simeq (\!|\ P_2'\ |\!)$. We can apply (C-Rcv) to mimic this transition.

$$\dfrac{\dfrac{(\!|\ P_1\ |\!) \overset{\Gamma':(\tilde{v})@\Pi}{\longmapsto}_{\Gamma} (\!|\ P_1'\ |\!) \qquad (\!|\ P_2\ |\!) \overset{\Gamma':(\tilde{v})@\Pi}{\longmapsto}_{\Gamma} (\!|\ P_2'\ |\!)}{(\!|\ P_1\ |\!)|(\!|\ P_2\ |\!) \overset{\Gamma':(\tilde{v})@\Pi}{\longmapsto}_{\Gamma} (\!|\ P_1'\ |\!)|(\!|\ P_2'\ |\!)}}{\Gamma : (\!|\ P_1\ |\!)|(\!|\ P_2\ |\!) \xrightarrow{\Gamma':(\tilde{v})@\Pi} \Gamma : (\!|\ P_1'\ |\!)|(\!|\ P_2'\ |\!)}$$

where $\Pi = (port_a = a)$ and $\Gamma' = \{(port_a, a)\}$ and again we have that: $(\!|\ P_1' \| P_2'\ |\!)_{\Gamma} \simeq \Gamma : (\!|\ P_1'\ |\!)|(\!|\ P_2'\ |\!)$.

—$P_1 \| P_2 \xrightarrow{\alpha} P_1' \| P_2$ 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 $(\!|\ P_1\ |\!) \twoheadrightarrow^* \simeq (\!|\ P_1'\ |\!)$ and $(\!|\ P_2\ |\!) \twoheadrightarrow^* \simeq (\!|\ P_2\ |\!)$. We have many cases depending on the performed action:

—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 (C-Upd) to mimic this transition.

$$\dfrac{\dfrac{(\!|\ P_1\ |\!) \overset{[port_\tau = \tau]}{\longmapsto}_{\Gamma} (\!|\ P_1'\ |\!)}{(\!|\ P_1\ |\!)|(\!|\ P_2\ |\!) \overset{[port_\tau = \tau]}{\longmapsto}_{\Gamma} (\!|\ P_1'\ |\!)|(\!|\ P_2\ |\!)}}{\Gamma : (\!|\ P_1\ |\!)|(\!|\ P_2\ |\!) \xrightarrow{\tau} \Gamma : (\!|\ P_1'\ |\!)|(\!|\ P_2\ |\!)}$$

and again we have that: $(\!|\ P_1' \| P_2\ |\!)_{\Gamma} \simeq \Gamma : (\!|\ P_1'\ |\!)|(\!|\ P_2\ |\!)$.

—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 (C-Rcv) to mimic this transition.

$$\dfrac{\dfrac{(\!|\ P_1\ |\!) \overset{\Gamma':(\tilde{v})@\Pi}{\longmapsto}_{\Gamma} (\!|\ P_1'\ |\!) \qquad (\!|\ P_2\ |\!) \overset{\widetilde{\Gamma':(\tilde{v})@\Pi}}{\longmapsto}_{\Gamma} (\!|\ P_2\ |\!)}{(\!|\ P_1\ |\!)|(\!|\ P_2\ |\!) \overset{\Gamma':(\tilde{v})@\Pi}{\longmapsto}_{\Gamma} (\!|\ P_1'\ |\!)|(\!|\ P_2\ |\!)}}{\Gamma : (\!|\ P_1\ |\!)|(\!|\ P_2\ |\!) \xrightarrow{\Gamma':(\tilde{v})@\Pi} \Gamma : (\!|\ P_1'\ |\!)|(\!|\ P_2\ |\!)}$$

where $\Pi = (port_a = a)$ and $\Gamma' = \{(port_a, a)\}$ and again we have that: $(\!|\ P_1' \| P_2\ |\!)_{\Gamma} \simeq \Gamma : (\!|\ P_1'\ |\!)|(\!|\ P_2\ |\!)$.

—if $\alpha = \nu\tilde{y}\tilde{a}\tilde{x}$ then $P_1 \| P_2 \xrightarrow{\nu\tilde{y}\tilde{a}\tilde{x}} P_1' \| P_2$ with $P_1 \xrightarrow{\nu\tilde{y}\tilde{a}\tilde{x}} P_1'$ and $P_2 \xrightarrow{a:}$. We can apply (C-Brd) if $\tilde{y} = \emptyset$ or (LOpen) otherwise. We consider (C-Brd) only and (LOpen) follows in the same way.

$$\dfrac{\dfrac{(\!|\ P_1\ |\!) \overset{\overline{\Gamma':(\tilde{v})@\Pi}}{\longmapsto}_\Gamma (\!|\ P_1'\ |\!) \qquad (\!|\ P_2\ |\!) \overset{\widetilde{\Gamma':(\tilde{v})@\Pi}}{\longmapsto}_\Gamma (\!|\ P_2\ |\!)}{(\!|\ P_1\ |\!)|(\!|\ P_2\ |\!) \overset{\overline{\Gamma':(\tilde{v})@\Pi}}{\longmapsto}_\Gamma (\!|\ P_1'\ |\!)|(\!|\ P_2\ |\!)}}{\Gamma:(\!|\ P_1\ |\!)|(\!|\ P_2\ |\!) \xrightarrow{\overline{\Gamma':(\tilde{v})@\Pi}} \Gamma:(\!|\ P_1'\ |\!)|(\!|\ P_2\ |\!)}$$

where $\Pi = (port_a = a)$ and $\Gamma' = \{(port_a, a)\}$ and again we have that:
$(\!|\ P_1' \| P_2\ |\!)_\Gamma \simeq \Gamma:(\!|\ P_1'\ |\!)|(\!|\ P_2\ |\!)$

—$P_1 \| P_2 \xrightarrow{a:}$ with $P_1 \xrightarrow{a:} P_1'$ and $P_2 \xrightarrow{a:}$. We can apply (C-Fail) to mimic this transition.

$$\dfrac{\dfrac{(\!|\ P_1\ |\!) \overset{\widetilde{\Gamma':(\tilde{v})@\Pi}}{\longmapsto}_\Gamma (\!|\ P_1\ |\!) \qquad (\!|\ P_2\ |\!) \overset{\widetilde{\Gamma':(\tilde{v})@\Pi}}{\longmapsto}_\Gamma (\!|\ P_2\ |\!)}{(\!|\ P_1\ |\!)|(\!|\ P_2\ |\!) \overset{\widetilde{\Gamma':(\tilde{v})@\Pi}}{\longmapsto}_\Gamma (\!|\ P_1\ |\!)|(\!|\ P_2\ |\!)}}{\Gamma:(\!|\ P_1\ |\!)|(\!|\ P_2\ |\!) \xrightarrow{\Gamma':(\tilde{v})@\Pi} \Gamma:(\!|\ P_1\ |\!)|(\!|\ P_2\ |\!)}$$

where $\Pi = (port_a = a)$ and $\Gamma' = \{(port_a, a)\}$ and again we have that: $(\!|\ P_1 \| P_2\ |\!)_\Gamma \simeq \Gamma:(\!|\ P_1\ |\!)|(\!|\ P_2\ |\!)$.

—if $P \triangleq \langle x = y\rangle P_1, P_2$: We have that either $\langle x = x\rangle P_1, P_2 \xrightarrow{\alpha} P_1'$ or $\langle x \neq x\rangle P_1, P_2 \xrightarrow{\alpha} P_2'$. By applying (Sum) rule or its symmetric, it is not hard to see that this is exactly the same meaning of the following translation $\langle x = y\rangle(\!|\ P\ |\!) + \langle x \neq y\rangle(\!|\ Q\ |\!)$ in the sense that $(\!|\ P\ |\!)$ can only evolve when $x = y$, otherwise $(\!|\ Q\ |\!)$ evolves.

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

REFERENCES

Yehia Abd Alrahman, Rocco De Nicola, Michele Loreti, Francesco Tiezzi, and Roberto Vigo. 2015. A Calculus for Attribute-based Communication. In *Proceedings of SAC 2015, The 30th ACM/SIGAPP Symposium On Applied Computing*.

Gul Agha and Christian J Callsen. 1993. *ActorSpace: an open distributed programming paradigm*. Vol. 28. ACM.

Michael A Bass and Frank T Nguyen. 2002. Unified publish and subscribe paradigm for local and remote publishing destinations. (June 11 2002). US Patent 6,405,266.

Gregory V Chockler, Idit Keidar, and Roman Vitenberg. 2001. Group communication specifications: a comprehensive study. *ACM Computing Surveys (CSUR)* 33, 4 (2001), 427–469.

Antonio Coronato, Vincenzo De Florio, Mohamed Bakhouya, and Giovanna Di Marzo Serugendo. 2012. Formal Modeling of Socio-technical Collective Adaptive Systems. In *Sixth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, SASOW 2012, Lyon, France, September 10-14, 2012*. 187–192. DOI:"http://dx.doi.org/10.1109/SASOW.2012.40

Rocco De Nicola, Gianluigi Ferrari, Michele Loreti, and Rosario Pugliese. 2013. A language-based approach to autonomic computing. In *Formal Methods for Components and Objects*. Springer, 25–48.

Rocco De Nicola, Michele Loreti, Rosario Pugliese, and Francesco Tiezzi. 2014. A formal approach to autonomic systems programming: the SCEL Language. *ACM Transactions on Autonomous and Adaptive Systems* (2014), 1–29.

Cristian Ene and Traian Muntean. 1999. Expressiveness of point-to-point versus broadcast communications. In *Fundamentals of Computation Theory*. Springer, 258–268.

Christian Ene and Traian Muntean. 2001. A broadcast-based calculus for communicating systems. In *Parallel and Distributed Processing Symposium, International*, Vol. 3. IEEE Computer Society, 30149b–30149b.

Patrick Th Eugster, Pascal A Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. 2003. The many faces of publish/subscribe. *ACM Computing Surveys (CSUR)* 35, 2 (2003), 114–131.

Thomas Given-Wilson, Daniele Gorla, and Barry Jay. 2010. Concurrent pattern calculus. In *Theoretical Computer Science*. Springer, 244–258.

Charles Antony Richard Hoare. 1978. Communicating sequential processes. *Commun. ACM* 21, 8 (1978), 666–677.

Hugh W Holbrook and David R Cheriton. 1999. IP multicast channels: EXPRESS support for large-scale single-source applications. In *ACM SIGCOMM Computer Communication Review*, Vol. 29. ACM, 65–78.

Mathias John, Cédric Lhoussaine, and Joachim Niehren. 2009. Dynamic compartments in the imperative π-Calculus. In *Computational Methods in Systems Biology*. Springer, 235–250.

Mathias John, Cédric Lhoussaine, Joachim Niehren, and Adelinde M Uhrmacher. 2008. The attributed pi calculus. In *Computational Methods in Systems Biology*. Springer, 83–102.

Mathias John, Cédric Lhoussaine, Joachim Niehren, and Adelinde M Uhrmacher. 2010. The attributed pi-calculus with priorities. In *Transactions on Computational Systems Biology XII*. Springer, 13–76.

Robin Milner. 1980. A calculus of communicating systems. (1980).

Robin Milner. 1989. *Communication and concurrency*. Prentice-Hall, Inc.

Robin Milner, Joachim Parrow, and David Walker. 1992. A calculus of mobile processes, II. *Information and computation* 100, 1 (1992), 41–77.

Robin Milner and Davide Sangiorgi. 1992. Barbed bisimulation. In *Automata, Languages and Programming*. Springer, 685–695.

Karol Ostrovsky, KVS Prasad, and Walid Taha. 2002. Towards a primitive higher order calculus of broadcasting systems. In *Proceedings of the 4th ACM SIGPLAN international conference on Principles and practice of declarative programming*. ACM, 2–13.

Rehan OGrady, Roderich Groß, Anders Lyhne Christensen, and Marco Dorigo. 2010. Self-assembly strategies in a group of autonomous mobile robots. *Autonomous Robots* 28, 4 (2010), 439–455.

Carlo Pinciroli, Michael Bonani, Francesco Mondada, and Marco Dorigo. 2015. Adaptation and awareness in robot ensembles: Scenarios and algorithms. In *Software Engineering for Collective Autonomic Systems*. Springer, 471–494.

KVS Prasad. 1991. A calculus of broadcasting systems. In *TAPSOFT'91*. Springer, 338–358.

Kuchi VS Prasad. 1995. A calculus of broadcasting systems. *Science of Computer Programming* 25, 2 (1995), 285–327.

Jeffrey W Sanders and Graeme Smith. 2008. Formal ensemble engineering. In *Software-Intensive Systems and New Computing Paradigms*. Springer, 132–138.

Ian Sommerville, Dave Cliff, Radu Calinescu, Justin Keen, Tim Kelly, Marta Kwiatkowska, John Mcdermid, and Richard Paige. 2012. Large-scale complex IT systems. *Commun. ACM* 55, 7 (2012), 71–77.

Roberto Vigo, Flemming Nielson, and Hanne Riis Nielson. 2013. Broadcast, Denial-of-Service, and Secure Communication. In *10th International Conference on integrated Formal Methods (iFM'13) (LNCS)*, Vol. 7940. 410–427.

INSTITUTE FOR ADVANCED STUDIES LUCCA

Piazza San ponziano 6, 5100 Lucca, Italy. www.imtlucca.it