On observing dynamic prioritised actions in SOC[⋆]

Rosario Pugliese[1], Francesco Tiezzi[1], and Nobuko Yoshida[2]

[1] Dipartimento di Sistemi e Informatica, Università degli Studi di Firenze
[2] Department of Computing, Imperial College London

Abstract. We study the impact on observational semantics for SOC of priority mechanisms which combine dynamic priority with local pre-emption. We define manageable notions of strong and weak labelled bisimilarities for COWS, a process calculus for SOC, and provide alternative characterisations in terms of open barbed bisimilarities. These semantics show that COWS's priority mechanisms partially recover the capability to observe receive actions (that could not be observed in a purely asynchronous setting) and that high priority primitives for termination impose specific conditions on the bisimilarities.

1 Introduction

Service-oriented computing (SOC) is an emergent paradigm for distributed computing that aims to build networks of interoperable applications through the use of platform-independent, reusable software components called *services*. Service definitions are used as templates for creating service instances that supply application functionalities to either end-user applications or other instances. Being inherently loosely coupled, SOC systems do not provide intrinsic mechanisms to identify service instances for delivering messages and to link together actions executed as part of the same client-service long-running interaction. Therefore, emerging standards like WS-BPEL and WS-CDL advocate the use of *correlation data* within exchanged messages that the interacting partners can retrieve by means of a *pattern matching* mechanism.

Recently, many process calculi have been expressly designed to model SOC scenarios, so that service definitions and service instances are represented as reactive processes running concurrently. In this setting, priority mechanisms which allow some actions to take precedence over others can be very fruitful. E.g., when a message arrives, the problem arises of rightly handling race conditions among those service instances and the corresponding service definition which are able to receive the message. This can be modelled by exploiting a parallel composition operator that gives precedence to actions with greater priority. Receive activities are then assigned priority values which depend on the messages available so that, in presence of concurrent matching receives, only a receive using a more defined pattern (i.e. having greater priority) can proceed. This way, service instances take precedence over the corresponding service definition when both can process the same message, thus preventing creation of wrong new instances.

Notably, receives would have *dynamically* assigned priority values since these values depend on the matching ability of their argument pattern. Indeed, while computation

[⋆] This work has been supported by the EU project SENSORIA, IST-2 005-016004, and EPSRC GR/T03215, GR/T03208 and EP/F003757.

proceeds, some of the variables used in the argument pattern of a receive can be assigned values, because of execution of syntactically preceding receives or of concurrent threads sharing these variables. This restricts the set of messages matching the pattern while increases the priority of the receive. Furthermore, pre-emption is *local* since receives having a more defined pattern have a higher execution priority with respect to only the other receives matching the same message.

There are other situations where local pre-emption is needed. For example, when a fault arises in a scope, (some of) the remaining activities of the enclosing scope should be terminated before starting the execution of the relative fault handler. This can be modelled by exploiting the same parallel operator as before together with actions for forcing immediate termination of concurrent activities which take the greatest priority. The same mechanism can also be used for exception and compensation handling.

However, apart from COWS [10, 13], priority mechanisms that combine dynamic priority with local pre-emption have not been studied yet in the literature [4]. COWS (*Calculus for Orchestration of Web Services*) is an extension of asynchronous π-calculus [7, 1] equipped with the priority mechanisms sketched above and with other distinctive features of SOC systems inspired by WS-BPEL, such as shared variables among parallel threads, and communication based on correlation and pattern matching.

This paper studies the impact of COWS's priority mechanisms on observational semantics for SOC. We first define strong and weak labelled bisimilarities for a fragment of COWS without primitives for termination, and prove that they are *sound* and *complete* with respect to contextual barbed ones (Section 3). Due to the locality of received endpoints, the labelled bisimilarities involves a family of relations indexed by sets of names, similar to the quasi-open bisimilarity for π-calculus [12]. The obtained semantics inhabits between asynchrony and synchrony because, with respect to a purely asynchronous setting, the priority mechanism permits partially recovering the capability to observe receive actions. We then extend our investigation and results to COWS (Section 4). We get that the primitives with greatest priority causing termination require specific conditions on the labelled bisimilarities for these to be congruences. Hence, the resulting observations are more fine-grained than the previous ones. Our semantic theories are usable to check interchangeability of services and conformance against service specifications as demonstrated through a practical example in Sections 2 and 4.

2 A 'Morra game' scenario

We start providing some insights into COWS's main features in a step-by-step fashion by means of an example service described at two different levels of abstraction.

The service allows its clients to play the well-known game Morra, where two players, named "odds" and "evens", throw out a single hand, each showing zero to five fingers. If the sum of fingers shown by both players is an even number then the "evens" player wins; otherwise the "odds" player is the winner. The service collects the two throws (i.e. two integers), calculates the winner and sends the result back to the two players. A high-level specification of the service in COWS is:

$$\begin{array}{r} *\,[x_{id}, x_p, x_{num}, y_p, y_{num}]\,(\; \mathit{odds}\bullet \mathit{throw}?\langle x_{id}, x_p, x_{num}\rangle \;|\; \mathit{evens}\bullet \mathit{throw}?\langle x_{id}, y_p, y_{num}\rangle \\ |\; x_p\bullet \mathit{res}!\langle x_{id}, \mathit{win}(x_{num}, y_{num}, 1)\rangle \;|\; y_p\bullet \mathit{res}!\langle x_{id}, \mathit{win}(x_{num}, y_{num}, 0)\rangle\;) \end{array} \qquad (1)$$

The replication operator $*\,\_$, that spawns in parallel as many copies of its argument term as necessary, supports creation of multiple instances to serve several matches simultaneously. The delimitation operator $[\_]$ declares the scope of variables x_{id}, x_p, y_p, x_{num} and y_{num}. Two distinct endpoints, i.e. pairs $odds \bullet throw$ and $evens \bullet throw$, are used by the service to receive throws from the players. When sending their throws, the players are required to provide a match identifier, stored in x_{id}, and the partner names, stored in x_p and y_p, that they will use to receive the result. To avoid interferences between matches played simultaneously, match-ids could be made unique by using delimitation.

Players throws arrive randomly, thus any interaction with the service starts with one of the two *receive* activities $odds \bullet throw?\langle x_{id}, x_p, x_{num}\rangle$ or $evens \bullet throw?\langle x_{id}, y_p, y_{num}\rangle$, that are *correlated* by means of the *shared* variable x_{id}, and terminates with the two *invoke* activities $x_p \bullet res!\langle x_{id}, win(x_{num}, y_{num}, 1)\rangle$ and $y_p \bullet res!\langle x_{id}, win(x_{num}, y_{num}, 0)\rangle$, used to reply with the result. We assume that $win(x, y, z)$ is a total function which, if x and y are integers between 0 and 5, returns w in case $(x + y)\, mod\ 2$ is equal to z, and l if $(x + y)\, mod\ 2$ is different from z; otherwise, err is returned.

A communication takes place when the arguments of a receive and of a concurrent invoke along the same endpoint match and causes replacement of the variables arguments of the receive with the corresponding values arguments of the invoke (within the scope of variables declarations). When operation *throw* is invoked, if a service instance with the same match-id already exists, then the invocation is received by the instance, otherwise a new instance is activated. This is done through the *dynamic prioritised mechanism* of COWS, i.e. assigning the receives by instances (having a more defined pattern) a greater priority than the receives by a service definition.

Thus, for example, after an interaction with the following client

$$[z]\,(\,evens \bullet throw!\langle first, cbB, 1\rangle \;|\; cbB \bullet res?\langle first, z\rangle\,.\,\langle\text{rest of client B}\rangle\,)$$

service definition (1) runs in parallel with the instance identified by the match-id *first*

$$\begin{array}{l} [x_p, x_{num}]\;(\,odds \bullet throw?\langle first, x_p, x_{num}\rangle \\ \qquad\qquad |\; x_p \bullet res!\langle first, win(x_{num}, 1, 1)\rangle \;|\; cbB \bullet res!\langle first, win(x_{num}, 1, 0)\rangle\,) \end{array}$$

Now, if another client performs the invocation $odds \bullet throw!\langle first, cbA, 2\rangle$, it will be processed by the already existing instance because, w.r.t. this invocation, the receive $odds \bullet throw?\langle first, x_p, x_{num}\rangle$ has greater priority than the receive $odds \bullet throw?\langle x_{id}, x_p, x_{num}\rangle$ occurring in (1) (that has a less defined argument pattern).

For a lower level implementation, we wish to maximise the abilities of different services, while preserving the observable behaviour of the whole service w.r.t. the high-level specification. The main service is now composed of three entities as follows:

$$[req2f, req5f, resp2f, resp5f]\,(\,*\,M \;|\; *\,2F \;|\; *\,5F\,) \tag{2}$$

The delimitation operator is used here to declare that *req2f*, *req5f*, *resp2f* and *resp5f* are private operation names known to the three components M, $2F$ and $5F$, and only to them. The three subservices are defined as follows:

$$\begin{array}{l} M \triangleq [x_{id}, x_p, x_{num}, y_p, y_{num}] \\ \quad (\,odds \bullet throw?\langle x_{id}, x_p, x_{num}\rangle \;|\; evens \bullet throw?\langle x_{id}, y_p, y_{num}\rangle \\ \quad\;\; |\; [k]\,(\,m \bullet req2f!\langle x_{id}, x_{num}, y_{num}\rangle \;|\; m \bullet req5f!\langle x_{id}, x_{num}, y_{num}\rangle \\ \qquad\quad |\; [x_o, x_e]\, m \bullet resp2f?\langle x_{id}, x_o, x_e\rangle.\,(\,\mathbf{kill}(k) \;|\; \{\!|x_p \bullet res!\langle x_{id}, x_o\rangle \;|\; y_p \bullet res!\langle x_{id}, x_e\rangle|\!\}\,) \\ \qquad\quad |\; [x_o, x_e]\, m \bullet resp5f?\langle x_{id}, x_o, x_e\rangle.\,(\,\mathbf{kill}(k) \;|\; \{\!|x_p \bullet res!\langle x_{id}, x_o\rangle \;|\; y_p \bullet res!\langle x_{id}, x_e\rangle|\!\}\,)\,) \end{array}$$

Table 1. μCOWS syntax

$s ::= u \bullet u' ! \bar{\epsilon} \;\mid\; g \;\mid\; s \mid s \;\mid\; [u]\, s \;\mid\; * s$	(invoke, guard, parallel, delimitation, replication)
$g ::= \mathbf{0} \;\mid\; p \bullet o ? \bar{w}. s \;\mid\; g + g$	(nil, receive, choice)

$$\begin{array}{ll}
\mathit{2F} \triangleq [x] & \mathit{5F} \triangleq [x, y, z] \\
\quad (\, m \bullet \mathit{req2f} ? \langle x, 1, 1 \rangle .\, m \bullet \mathit{resp2f} ! \langle x, l, w \rangle & \quad (\, m \bullet \mathit{req5f} ? \langle x, y, z \rangle .\, m \bullet \mathit{resp5f} ! \langle x, \mathit{err}, \mathit{err} \rangle \\
\quad\; + m \bullet \mathit{req2f} ? \langle x, 1, 2 \rangle .\, m \bullet \mathit{resp2f} ! \langle x, w, l \rangle & \quad\; + m \bullet \mathit{req5f} ? \langle x, 0, 0 \rangle .\, m \bullet \mathit{resp5f} ! \langle x, l, w \rangle \\
\quad\; + m \bullet \mathit{req2f} ? \langle x, 2, 1 \rangle .\, m \bullet \mathit{resp2f} ! \langle x, w, l \rangle & \quad\; + m \bullet \mathit{req5f} ? \langle x, 0, 1 \rangle .\, m \bullet \mathit{resp5f} ! \langle x, w, l \rangle \\
\quad\; + m \bullet \mathit{req2f} ? \langle x, 2, 2 \rangle .\, m \bullet \mathit{resp2f} ! \langle x, l, w \rangle \,) & \quad\; + \ldots + m \bullet \mathit{req5f} ? \langle x, 5, 5 \rangle .\, m \bullet \mathit{resp5f} ! \langle x, l, w \rangle)
\end{array}$$

Service *M* is publicly invocable and can interact with players as well as with the 'internal' services *2F* and *5F*. These latter two services, instead, can only be invoked by *M* and have the task of calculating the winner of a match. In particular, *2F* performs a quick computation of simple matches where both players hold out either one or two fingers, while *5F* performs a slower computation of standard 5-fingers matches (that exactly corresponds to the computation modelled by the function $\mathit{win}(\_)$). After the two initial receives, for e.g. performance and fault tolerance purposes, *M* invokes services *2F* and *5F* concurrently. Communication between *M* and the other two subservices exploits the match identifier (stored in x) as a correlation datum. When one of *2F* and *5F* replies, *M* immediately stops the other computation. This is done by executing the *kill* activity $\mathbf{kill}(k)$, that forces termination of all unprotected parallel terms inside the enclosing $[k]$, which stops the killing effect. Kill activities take the greatest priority w.r.t. the other parallel activities included within the enclosing scope. However, critical activities can be protected from the effect of a forced termination by using the *protection* operator $\{\!|\_|\!\}$; this is indeed the case of the response $x_p \bullet \mathit{res} ! \langle x_{id}, x_o \rangle$ in our example. Finally, *M* forwards the responses to the players and terminates.

Services *2F* and *5F* use the *choice* operator $\_ + \_$ to offer alternative behaviours: one of them can be selected by executing an invoke matching the receive leading the behaviour. If the throws are not integers between 0 and 5, *2F* does not reply, while *5F* returns the string *err*. Indeed, the receive $m \bullet \mathit{req5f} ? \langle x, y, z \rangle$ is assigned less priority than the other receive activities, i.e. it is only executed when none of the other receives matches the two throws, thus avoiding to return *err* in case of admissible throws.

3 μCOWS : the protection- and kill-free fragment of COWS

μCOWS's syntax is presented in Table 1. We use two countable and disjoint sets: the set of *values* (ranged over by v, v', ...) and the set of 'write once' *variables* (ranged over by $x, y, \ldots$). The set of values includes the set of *names* (ranged over by n, m, p, o, ...) mainly used to represent partners and operations. We also use a set of *expressions* (ranged over by ϵ) which contains, at least, values and variables. Partner names and operation names can be combined to designate *endpoints*, written $p \bullet o$, and can be communicated, but dynamically received names can only be used for service invocation.

We use w to range over values and variables and u to range over names and variables. $\bar{\cdot}$ stands for tuples, e.g. $\bar{x}$ means $\langle x_1, \ldots, x_n \rangle$ (with $n \geq 0$) where the x_is are pairwise distinct. We write $a, \bar{b}$ to denote the tuple obtained by concatenating the element a to the tuple $\bar{b}$. All notations shall extend to tuples component-wise. $\mathtt{n}$ ranges over

Table 2. Matching rules

$$\mathcal{M}(x, v) = \{x \mapsto v\} \qquad \mathcal{M}(v, v) = \emptyset \qquad \mathcal{M}(\langle\rangle, \langle\rangle) = \emptyset \qquad \frac{\mathcal{M}(w_1, v_1) = \sigma_1 \qquad \mathcal{M}(\bar{w}_2, \bar{v}_2) = \sigma_2}{\mathcal{M}((w_1, \bar{w}_2), (v_1, \bar{v}_2)) = \sigma_1 \uplus \sigma_2}$$

communication endpoints that do not contain variables (e.g. $p \bullet o$), while $\mathtt{u}$ ranges over communication endpoints that may contain variables (e.g. $u \bullet u'$). When convenient, we shall regard a tuple or an endpoint simply as a set, writing e.g. $x \in \bar{y}$ to mean that x is an element of $\bar{y}$. We will omit trailing occurrences of $\mathbf{0}$ and write $[u_1, \ldots, u_n]\, s$ in place of $[u_1] \ldots [u_n]\, s$. We will write $I \triangleq s$ to assign a name I to the term s.

We assume that monadic operators bind more tightly than parallel composition, and prefixing more tightly than choice. The only *binding* construct is delimitation: $[u]\, s$ binds u in the scope s. The occurrence of a name/variable is *free* if it is not under the scope of a delimitation for it. $\mathrm{fu}(t)$ denotes the set of free names/variables in t.

The operational semantics of μCOWS is defined only for *closed* terms, i.e. terms without free variables, and is given in terms of a structural congruence and of a labelled transition relation. The *structural congruence*, written $\equiv$, is defined as the least congruence relation induced by a given set of equational laws. We explicitly show here the laws for replication and delimitation

$$*\,\mathbf{0} \equiv \mathbf{0} \quad *\,s \equiv s \,|\, *\,s \quad [u]\,\mathbf{0} \equiv \mathbf{0} \quad [u_1]\,[u_2]\,s \equiv [u_2]\,[u_1]\,s \quad s_1 \mid [u]\,s_2 \equiv [u]\,(s_1 \mid s_2) \text{ if } u \notin \mathrm{fu}(s_1)$$

while omit the (standard) laws for the other operators stating that parallel composition and guarded choice are commutative, associative and have $\mathbf{0}$ as identity element. All the presented laws are straightforward. In particular, the last law permits to extend the scope of names (as in π-calculus) and variables, thus enabling possible communication.

To define the labelled transition relation, we use three auxiliary functions. Firstly, we use the function $[\![\_]\!]$ for evaluating *closed* expressions (i.e. expressions without variables): it takes a closed expression and returns a value. It is not explicitly defined since the exact syntax of expressions is deliberately not specified. Secondly, we use the partial function $\mathcal{M}(\_, \_)$ for performing *pattern-matching* on semi-structured data and, thus, determining if a receive and an invoke over the same endpoint can synchronise. The (straightforward) rules defining $\mathcal{M}(\_, \_)$ are shown in Table 2. When tuples $\bar{w}$ and $\bar{v}$ match, $\mathcal{M}(\bar{w}, \bar{v})$ returns a substitution for the variables in $\bar{w}$; otherwise, it is undefined. *Substitutions* (ranged over by σ) are functions mapping variables to values and are written as collections of pairs of the form $x \mapsto v$. Application of σ to s, written $s \cdot \sigma$, has the effect of replacing every free occurrence of x in s with v, for each $x \mapsto v \in \sigma$, by possibly using α-conversion for avoiding v to be captured by name delimitations within s. We use $\emptyset$ to denote the empty substitution, $|\,\sigma\,|$ to denote the number of pairs in σ, and $\sigma_1 \uplus \sigma_2$ to denote the union of σ_1 and σ_2 when they have disjoint domains. Finally, in Table 3, we inductively define a predicate for checking communication conflicts: $\mathrm{noConf}(s, \mathtt{n}, \bar{v}, \ell)$ holds true if s cannot immediately perform a receive over the endpoint $\mathtt{n}$ matching $\bar{v}$ and generating a substitution σ with lesser pairs than ℓ.

The *labelled transition relation*, written $\xrightarrow{\alpha}$, is the least relation over terms induced by the rules in Table 4, where label α is generated by the following grammar:

$$\alpha \;::=\; \mathtt{n} \triangleleft [\bar{n}]\, \bar{v} \quad | \quad \mathtt{n} \triangleright [\bar{x}]\, \bar{w} \quad | \quad \sigma \quad | \quad \mathtt{n}\, \sigma\, \ell\, \bar{v}$$

Table 3. There are not conflicting receives along $\mathtt{n}$ matching $\bar{v}$

$$\mathrm{noConf}(\mathtt{u}!\bar{\epsilon}, \mathtt{n}, \bar{v}, \ell) \;=\; \mathrm{noConf}(\mathbf{0}, \mathtt{n}, \bar{v}, \ell) \;=\; \mathbf{true} \qquad \mathrm{noConf}(*\, s, \mathtt{n}, \bar{v}, \ell) \;=\; \mathrm{noConf}(s, \mathtt{n}, \bar{v}, \ell)$$

$$\mathrm{noConf}(\mathtt{n}'?\bar{w}.s, \mathtt{n}, \bar{v}, \ell) \;=\; \begin{cases} \mathbf{false} & \text{if } \mathtt{n}' = \mathtt{n} \wedge\ |\mathcal{M}(\bar{w}, \bar{v})| < \ell \\ \mathbf{true} & \text{otherwise} \end{cases}$$

$$\mathrm{noConf}(s \mid s', \mathtt{n}, \bar{v}, \ell) \;=\; \mathrm{noConf}(s + s', \mathtt{n}, \bar{v}, \ell) \;=\; \mathrm{noConf}(s, \mathtt{n}, \bar{v}, \ell) \,\wedge\, \mathrm{noConf}(s', \mathtt{n}, \bar{v}, \ell)$$

$$\mathrm{noConf}([u]\, s, \mathtt{n}, \bar{v}, \ell) \;=\; \begin{cases} \mathrm{noConf}(s, \mathtt{n}, \bar{v}, \ell) & \text{if } u \notin \mathtt{n} \\ \mathbf{true} & \text{otherwise} \end{cases}$$

Table 4. μCOWS operational semantics

$$\frac{[\![\bar{\epsilon}]\!] = \bar{v}}{\mathtt{n}!\bar{\epsilon} \xrightarrow{\mathtt{n} \triangleleft \bar{v}} \mathbf{0}}\ (inv) \qquad \mathtt{n}?\bar{w}.s \xrightarrow{\mathtt{n} \triangleright \bar{w}} s\ \ (rec) \qquad \frac{g \xrightarrow{\alpha} s}{g + g' \xrightarrow{\alpha} s}\ (choice)$$

$$\frac{s \xrightarrow{\mathtt{n} \triangleleft [\bar{m}]\, \bar{v}} s' \qquad n \in \bar{v} \qquad n \notin (\mathtt{n} \cup \bar{m})}{[n]\, s \xrightarrow{\mathtt{n} \triangleleft [n,\bar{m}]\, \bar{v}} s'}\ (open_{inv}) \qquad \frac{s \xrightarrow{\sigma \uplus \{x \mapsto v\}} s'}{[x]\, s \xrightarrow{\sigma} s' \cdot \{x \mapsto v\}}\ (del_{com})$$

$$\frac{s \xrightarrow{\mathtt{n} \triangleright [\bar{y}]\, \bar{w}} s' \qquad x \in \bar{w} \qquad x \notin \bar{y}}{[x]\, s \xrightarrow{\mathtt{n} \triangleright [x,\bar{y}]\, \bar{w}} s'}\ (open_{rec}) \qquad \frac{s \xrightarrow{\mathtt{n}\, \sigma \uplus \{x \mapsto v\}\, \ell\, \bar{v}} s'}{[x]\, s \xrightarrow{\mathtt{n}\, \sigma\, \ell\, \bar{v}} s' \cdot \{x \mapsto v\}}\ (del_{com\,2})$$

$$\frac{s_1 \xrightarrow{\mathtt{n} \triangleright \bar{v}} s_1' \qquad s_2 \xrightarrow{\mathtt{n} \triangleleft \bar{v}} s_2'}{s_1 \mid s_2 \xrightarrow{\emptyset} s_1' \mid s_2'}\ (match) \qquad \frac{s \xrightarrow{\alpha} s' \quad u \notin (\mathrm{u}(\alpha) \cup \mathrm{ce}(\alpha))}{[u]\, s \xrightarrow{\alpha} [u]\, s'}\ (del)$$

$$\frac{s_1 \xrightarrow{\mathtt{n} \triangleright \bar{w}} s_1' \qquad s_2 \xrightarrow{\mathtt{n} \triangleleft \bar{v}} s_2' \qquad \mathcal{M}(\bar{w}, \bar{v}) = \sigma \qquad |\sigma| \geqslant 1 \qquad \mathrm{noConf}(s_1 \mid s_2, \mathtt{n}, \bar{v}, |\sigma|)}{s_1 \mid s_2 \xrightarrow{\mathtt{n}\, \sigma\, |\sigma|\, \bar{v}} s_1' \mid s_2'}\ (com)$$

$$\frac{s_1 \xrightarrow{\mathtt{n}\, \sigma\, \ell\, \bar{v}} s_1' \qquad \mathrm{noConf}(s_2, \mathtt{n}, \bar{v}, \ell)}{s_1 \mid s_2 \xrightarrow{\mathtt{n}\, \sigma\, \ell\, \bar{v}} s_1' \mid s_2}\ (par_{com}) \qquad \frac{s_1 \xrightarrow{\alpha} s_1' \qquad \alpha \neq \mathtt{n}\, \sigma\, \ell\, \bar{v}}{s_1 \mid s_2 \xrightarrow{\alpha} s_1' \mid s_2}\ (par)$$

$$\frac{s \xrightarrow{\mathtt{n}\, \sigma\, \ell\, \bar{v}} s' \qquad n \in \mathtt{n}}{[n]\, s \xrightarrow{\sigma} [n]\, s'}\ (private) \qquad \frac{s \equiv s_1 \quad s_1 \xrightarrow{\alpha} s_2 \quad s_2 \equiv s'}{s \xrightarrow{\alpha} s'}\ (str)$$

Labels $\mathtt{n} \triangleleft [\bar{n}]\, \bar{v}$ and $\mathtt{n} \triangleright [\bar{x}]\, \bar{w}$ denote execution of invoke and receive activities over the endpoint $\mathtt{n}$, resp., while labels σ and $\mathtt{n}\, \sigma\, \ell\, \bar{v}$ denote execution of a communication with generated substitution σ to be still applied. Thus, $\emptyset$ and $\mathtt{n}\, \emptyset\, \ell\, \bar{v}$ denote *computational steps* corresponding to taking place of communication without pending substitutions. In the sequel, we will write $\mathtt{n} \triangleleft \bar{v}$ (resp. $\mathtt{n} \triangleright \bar{w}$) instead of $\mathtt{n} \triangleleft [\,]\, \bar{v}$ (resp. $\mathtt{n} \triangleright [\,]\, \bar{w}$) and use $\mathrm{u}(\alpha)$ to denote the set of names and variables occurring in α, where $\mathrm{u}(\mathtt{n}\, \sigma\, \ell\, \bar{v}) = \mathrm{u}(\sigma)$, $\mathrm{u}(\{x \mapsto v\}) = \{x\} \cup \mathrm{fu}(v)$ and $\mathrm{u}(\sigma_1 \uplus \sigma_2) = \mathrm{u}(\sigma_1) \cup \mathrm{u}(\sigma_2)$. We use $\mathrm{ce}(\alpha)$ to denote the names composing the endpoint if α denotes execution of a communication, i.e. $\mathrm{ce}(\alpha)$ is $\emptyset$ except for $\alpha = \mathtt{n}\, \sigma\, \ell\, \bar{v}$ for which we let $\mathrm{ce}(\mathtt{n}\, \sigma\, \ell\, \bar{v}) = \mathtt{n}$. Finally, we use $\mathrm{bu}(\alpha)$ to denote the set of names/variables that occur bound in α; i.e. $\mathrm{bu}(\alpha)$ is $\emptyset$ except for $\alpha = \mathtt{n} \triangleleft [\bar{n}]\, \bar{v}$ and $\alpha = \mathtt{n} \triangleright [\bar{x}]\, \bar{w}$ for which we let $\mathrm{bu}(\mathtt{n} \triangleleft [\bar{n}]\, \bar{v}) = \bar{n}$ and $\mathrm{bu}(\mathtt{n} \triangleright [\bar{x}]\, \bar{w}) = \bar{x}$.

We comment on salient points of rules in Table 4. An invocation can proceed only if the expressions in the argument can be evaluated *(inv)*. This means, for example, that if it contains a variable x it is stuck until x is not replaced by a value because of execution of a receive assigning a value to x. A receive activity offers an invocable operation along a given partner name *(rec)*, and the execution of a receive permits to take a decision between alternative behaviours *(choice)*. Bound invocations, that transmit private names, can be generated by *(open$_{inv}$)*, while delimited receive activities can proceed by *(open$_{rec}$)*.

Communication can take place when two parallel terms perform matching receive and invoke activities *(com)*. Communication generates a substitution that is recorded in the transition label (for subsequent application), rather than a silent transition as in most process calculi. In particular, two different kinds of communication label can be generated: σ and $\mathtt{n}\,\sigma\,\ell\,\bar{v}$. The latter label, produced by *(com)*, carries information about the communication which has taken place (i.e. the endpoint, the transmitted values, the generated substitution and its length) used to check the presence of conflicting receives in parallel components. Indeed, if more then one matching is possible, the receive that needs fewer substitutions is selected to progress (*(com)* and *(par$_{com}$)*). This mechanism permits to correlate different service communications thus implicitly creating a long-running interaction and can be exploited to model the precedence of a service instance over the corresponding service specification when both can process the same request. However, the check for the presence of a conflict is not needed when either the performed receive has the highest priority (i.e. the substitution has length 0) or the communication takes place along a private endpoint. In the former case, label $\emptyset$ is immediately generated by *(match)*. In the latter case, when the delimitation of a name belonging to the endpoint of a communication label is encountered (i.e. the communication is identified as private), the transition label $\mathtt{n}\,\sigma\,\ell\,\bar{v}$ is turned into σ *(private)*.

When the delimitation of a variable x argument of a receive involved in a communication is encountered, i.e. the whole scope of the variable is determined, the delimitation is removed and the substitution for x is applied to the term (*(del$_{com}$)* and *(del$_{com\,2}$)*); thus, x disappears from the term and cannot be reassigned a value. $[u]\,s$ behaves like s *(del)*, except when the transition label α contains u. Execution of parallel terms is interleaved *(par)*, but when a communication subject to conflict check is performed. Indeed, it must ensure that the receive activity with greater priority progresses (*(com)* and *(par$_{com}$)*).

Now, we want to define a co-inductive notion of *bisimulation* for the calculus. Since communication is asynchronous, an obvious starting point is considering as observable only the output capabilities of terms, as done by the labelled bisimulation introduced for asynchronous π-calculus in [1]. The intuition is that an asynchronous observer cannot directly observe the receipt of data that it has sent. Moreover, to enable compositional reasoning, we want our bisimulation to be a *congruence*, namely to be preserved by all μCOWS (closed) contexts $\mathbb{C}$ that are generated by the following grammar:

$$\mathbb{C} ::= [\![\cdot]\!] \;\mid\; \mathbb{G} \;\mid\; \mathbb{C} \mid s \;\mid\; s \mid \mathbb{C} \;\mid\; [u]\,\mathbb{C} \;\mid\; *\,\mathbb{C} \qquad \mathbb{G} ::= \mathtt{n}?\bar{w}.\,\mathbb{C} \;\mid\; \mathbb{G} + g \;\mid\; g + \mathbb{G}$$

such that, once the hole is filled with a closed term s, $\mathbb{C}[\![s]\!]$ is a μCOWS closed term.

In [13], we show that for μCOWS^m, a fragment of μCOWS that dispenses with priority in parallel composition, a notion of bisimulation inspired to [1] enjoys the equality

$$[x]\,(\,\emptyset + \mathtt{n}?\langle x, v\rangle.\,\mathtt{n}!\langle x, v\rangle\,) \;=\; \emptyset \qquad (3)$$

where, for the sake of presentation, we exploit the context $\emptyset + [\![\cdot]\!] \triangleq [\mathtt{m}]\,(\mathtt{m}!\langle\rangle \mid \mathtt{m}?\langle\rangle + [\![\cdot]\!])$ and the term $\emptyset \triangleq [\mathtt{m}]\,(\mathtt{m}!\langle\rangle \mid \mathtt{m}?\langle\rangle)$[1]. Intuitively, the equality means that a term that emits

[1] $\emptyset$ plays a role similar to τ in π-calculus, namely $\emptyset$ is both a label and a term such that $\emptyset \xrightarrow{\emptyset} \mathbf{0}$.

the data it has received behaves as a term that simply performs an unobservable action and is an analogous of the input absorption law, i.e. $a(b).\,\bar{a}b + \tau = \tau$, characterising strong bisimilarity for asynchronous π-calculus [1].

In μCOWS, instead, the context $\mathbb{C} \triangleq [y,z]\,\mathtt{n}?\langle y,z\rangle.\,\mathtt{m}!\langle\rangle \mid \mathtt{n}!\langle v',v\rangle \mid [\![\cdot]\!]$ can tell the two terms above apart. In fact, we have $\mathbb{C}[\![\emptyset]\!] \xrightarrow{\mathtt{n}\,\emptyset\,2\,\langle v',v\rangle} \mathtt{m}!\langle\rangle \mid \emptyset$, where the term $(\mathtt{m}!\langle\rangle \mid \emptyset)$ can perform the invoke $\mathtt{m}!\langle\rangle$. Instead, the other term cannot properly reply because the receive $\mathtt{n}?\langle x,v\rangle$ has higher priority than $\mathtt{n}?\langle y,z\rangle$ when synchronising with the invocation $\mathtt{n}!\langle v',v\rangle$. Thus, $\mathbb{C}[\![[x]\,(\,\emptyset + \mathtt{n}?\langle x,v\rangle.\,\mathtt{n}!\langle x,v\rangle\,)]\!]$ can only evolve to terms that cannot immediately perform the activity $\mathtt{m}!\langle\rangle$. This means that μCOWS^m's notion of bisimulation is not a congruence for μCOWS. This is due to the fact that receive activities that exercise a priority (i.e. receives whose arguments contain some values) can be detected by an interacting observer (as shown by the above example). Hence, for a suitable notion of labelled bisimilarity for μCOWS, equation (3) does not hold. Now, consider the term $[x,x']\,(\,\emptyset + \mathtt{n}?\langle x,x'\rangle.\,\mathtt{n}!\langle x,x'\rangle\,)$. Since $\mathtt{n}?\langle x,x'\rangle$ does not exercise any priority on parallel terms, the context $\mathbb{C}$ cannot tell the term above and $\emptyset$ apart. Similarly, we have that $\emptyset + \mathtt{n}?\langle\rangle.\,\mathtt{n}!\langle\rangle$ and $\emptyset$ cannot be distinguished by $\mathbb{D} \triangleq \mathtt{n}?\langle\rangle.\,\mathtt{m}!\langle\rangle \mid \mathtt{n}!\langle\rangle \mid [\![\cdot]\!]$. Therefore, such pairs of terms should be considered as bisimilar.

Now, consider the terms $s_1 \triangleq [\mathtt{n}]\,(\,\mathtt{m}!\langle\mathtt{n}\rangle \mid \mathtt{n}!\langle\rangle\,)$ and $s_2 \triangleq [\mathtt{n}]\,\mathtt{m}!\langle\mathtt{n}\rangle$. Although the former also contains the subterm $\mathtt{n}!\langle\rangle$, they can both perform only the invocation along the endpoint $\mathtt{m}$. In fact, $\mathtt{n}!\langle\rangle$ is blocked since initially it is in the scope of $[\mathtt{n}]$ and afterwards no interacting partner can ever be able to receive along $\mathtt{n}$ (contexts of the form $[\![\cdot]\!] \mid \mathtt{m}?\langle x\rangle.\,x?\langle\rangle.\,\mathbf{0}$ are not allowed because of the syntactic constraint on the 'localisation' of names). Therefore, s_1 and s_2 should be considered as bisimilar. Instead, the natural asynchronous labelled bisimilarity derived from [1] would tell them apart and, hence, need to be weakened. Hence, we define a labelled bisimulation as a family of relations indexed with sets of names corresponding to the names that cannot be used by contexts (to test) for reception since they are dynamically exported private names.

Definition 1. *A* names-indexed family $\mathcal{F}$ of relations *is a set of symmetric binary relations $\mathcal{R}_N$ on μCOWS closed terms, one for each set of names N, i.e. $\mathcal{F} = \{\mathcal{R}_N\}_N$.*

Definition 2 (Labelled bisimilarity). *A names-indexed family of relations $\{\mathcal{R}_N\}_N$ is a* labelled bisimulation *if, whenever $s_1 \mathcal{R}_N s_2$ and $s_1 \xrightarrow{\alpha} s_1'$, where* bu$(\alpha)$ *are fresh, then:*

1. *if $\alpha = \mathtt{n} \triangleright [\bar{x}]\,\bar{w}$ then one of the following holds:*
   (a) $\exists\, s_2' \,:\, s_2 \xrightarrow{\mathtt{n}\triangleright[\bar{x}]\,\bar{w}} s_2'$ *and*
   $\forall\,\bar{v}$ *s.t.* $\mathcal{M}(\bar{x},\bar{v}) = \sigma$ *and* noConf$(s_2, \mathtt{n}, \bar{w}\cdot\sigma, |\bar{x}|) \,:\, s_1'\cdot\sigma\,\mathcal{R}_N\, s_2'\cdot\sigma$
   (b) $|\bar{x}| = |\bar{w}|$ *and* $\exists\, s_2' \,:\, s_2 \xrightarrow{\emptyset} s_2'$ *and*
   $\forall\,\bar{v}$ *s.t.* $\mathcal{M}(\bar{x},\bar{v}) = \sigma$ *and* noConf$(s_2, \mathtt{n}, \bar{w}\cdot\sigma, |\bar{x}|) \,:\, s_1'\cdot\sigma\,\mathcal{R}_N\,(s_2' \mid \mathtt{n}!\bar{v})$
2. *if $\alpha = \mathtt{n}\,\emptyset\,\ell\,\bar{v}$ where $\ell = |\bar{v}|$ then one of the following holds:*
   (a) $\exists\, s_2' \,:\, s_2 \xrightarrow{\mathtt{n}\,\emptyset\,\ell\,\bar{v}} s_2'$ *and* $s_1'\,\mathcal{R}_N\,s_2'$ *(b)* $\exists\, s_2' \,:\, s_2 \xrightarrow{\emptyset} s_2'$ *and* $s_1'\,\mathcal{R}_N\,s_2'$
3. *if $\alpha = \mathtt{n} \triangleleft [\bar{n}]\,\bar{v}$ where $\mathtt{n} \notin N$ then $\exists\, s_2' \,:\, s_2 \xrightarrow{\mathtt{n}\triangleleft[\bar{n}]\,\bar{v}} s_2'$ and $s_1'\,\mathcal{R}_{N\cup\bar{n}}\,s_2'$*
4. *if $\alpha = \emptyset$ or $\alpha = \mathtt{n}\,\emptyset\,\ell\,\bar{v}$, where $\ell \neq |\bar{v}|$, then $\exists\, s_2' \,:\, s_2 \xrightarrow{\alpha} s_2'$ and $s_1'\,\mathcal{R}_N\,s_2'$*

*Two closed terms s_1 and s_2 are $\mathcal{N}$-*bisimilar*, written $s_1 \sim_\mu^{\mathcal{N}} s_2$, if $s_1 \mathcal{R}_{\mathcal{N}} s_2$ for some $\mathcal{R}_{\mathcal{N}}$ in a labelled bisimulation. They are* labelled bisimilar*, written $s_1 \sim_\mu s_2$, if they are $\emptyset$-bisimilar. $\sim_\mu^{\mathcal{N}}$ is called $\mathcal{N}$-bisimilarity, while $\sim_\mu$ is called labelled bisimilarity.*

The resulting definition somewhat recalls that of quasi-open bisimilarity for π-calculus [12]. Clause 1 deals with both observable and unobservable receives. In fact, all receives can be simulated in a normal way (clause 1.(a)); additionally, receives such that $|\bar{x}|=|\bar{w}|$, i.e. $\bar{w}$ contains only variables or is the empty tuple (since $\bar{x} \subseteq \bar{w}$ and $\bar{w} \backslash \bar{x}$ does not contain variables), can be simulated by a computational step leading to a term that, when composed with the invoke activity consumed by the receive, stands in the appropriate relation (clause 1.(b)). Execution of receives whose argument contains variables leads to open terms, which the operational semantics is not defined for. Since the freed variables are placeholders for values to be received, clause 1 requires the two continuations to be related for any matching tuple of values that can be effectively received (i.e. that do not give rise to communication conflicts). Clause 2 permits replying also with an $\emptyset$-transition to communications involving an unobservable receive ($\ell=|\bar{v}|$ implies that the argument of the receive is a, possible empty, tuple of variables). Clause 3, and the use of names-indexed families of relations, handles the fact that dynamically exported private names cannot be used by a receiver within the endpoint of a receive (whose syntax does not allow to use variables). With abuse of notation, $\mathtt{n} \notin \mathcal{N}$ in clause 3, with $\mathtt{n} = p \bullet o$, stands for $p \notin \mathcal{N} \wedge o \notin \mathcal{N}$. Thus, invocations along endpoints using either of the names in $\mathcal{N}$ are unobservable, hence these endpoints cannot be used to tell the executing terms apart. Finally, clause 4 deals with computational steps. Notably, actions σ and $\mathtt{n}\,\sigma\,\ell\,\bar{v}$, with $\sigma \neq \emptyset$, are not taken into account, since they cannot be performed by closed terms (see rules *(com)*, *(del$_{com}$)* and *(del$_{com2}$)*).

Theorem 1. *$\sim_\mu$ is a congruence for μ*COWS *closed terms.*

As a further evidence of the reasonableness of our notion of bisimilarity, we provide now an alternative characterization in terms of *(open) barbed bisimilarity* along the line of [8, 12]. To this aim, first we identify an appropriate *basic observable*, namely a predicate that points out the interaction capabilities of a term. Since communication is asynchronous, again we consider as observable only the output capabilities of terms.

Definition 3 (Observable for μCOWS). *Let s be a μ*COWS *closed term. Predicate $s \downarrow_{\mathtt{n}}$ holds true if there exist s', $\bar{n}$ and $\bar{v}$ such that $s \xrightarrow{\mathtt{n} \triangleleft [\bar{n}]\, \bar{v}} s'$.*

Definition 4 (Open barbed bisimilarity). *A symmetric binary relation $\mathcal{R}$ on μ*COWS *closed terms is an* open barbed bisimulation *if whenever $s_1 \mathcal{R} s_2$ the following holds:*

(Barb preservation) *if $s_1 \downarrow_{\mathtt{n}}$ then $s_2 \downarrow_{\mathtt{n}}$;*
(Computation closure) *if $s_1 \xrightarrow{\emptyset} s_1'$ (resp. $s_1 \xrightarrow{\mathtt{n}\,\emptyset\,\ell\,\bar{v}} s_1'$) then there exists s_2' such that $s_2 \xrightarrow{\emptyset} s_2'$ (resp. $s_2 \xrightarrow{\mathtt{n}\,\emptyset\,\ell\,\bar{v}} s_2'$ or $\ell=|\bar{v}| \wedge s_2 \xrightarrow{\emptyset} s_2'$) and $s_1' \mathcal{R} s_2'$;*
(Context closure) *$\mathbb{C}[\![s_1]\!] \,\mathcal{R}\, \mathbb{C}[\![s_2]\!]$, for every closed context $\mathbb{C}$.*

Two closed terms s_1 and s_2 are open barbed bisimilar*, written $s_1 \simeq_\mu s_2$, if $s_1 \mathcal{R} s_2$ for some open barbed bisimulation $\mathcal{R}$. $\simeq_\mu$ is called open barbed bisimilarity.*

Barbed bisimilarity has the advantage of an intuitive meaning, since it is induced by a simple notion of observable and is defined by means of computation and context closure. Differently from $\sim_\mu$, the definition of $\simeq_\mu$ suffers from universal quantification over all possible language contexts, which makes the reasoning on terms very hard.

Our main results state that labelled bisimilarity is *sound* and *complete* with respect to open barbed bisimilarity. Due to the intuitiveness of $\simeq_\mu$, this result makes us confident that the notion of labelled bisimularity is sufficiently reasonable.

Theorem 2. *$\sim_\mu$ and $\simeq_\mu$ coincide.*

Our semantic theories extend in a standard way to the weak case so that results of congruence and coincidence still hold. Due to space limitations, the exact definitions are relegated to [13]. Here, we conclude with an example inspired to the law $!(a(b).\,\bar{a}b) = \mathbf{0}$ that holds for weak bisimilarity in asynchronous π-calculus [1]. In fact, the analogous of equality (3) for the weak case is $*\,[x,x']\,\mathtt{n}?\langle x,x'\rangle.\,\mathtt{n}!\langle x,x'\rangle \approx_\mu \mathbf{0}$. To prove validity, the most significant case is simulating the transition

$$*\,[x,x']\,\mathtt{n}?\langle x,x'\rangle.\,\mathtt{n}!\langle x,x'\rangle \xrightarrow{\mathtt{n}\triangleright[x,x']\,\langle x,x'\rangle} *\,[x,x']\,(\mathtt{n}?\langle x,x'\rangle.\,\mathtt{n}!\langle x,x'\rangle) \mid \mathtt{n}!\langle x,x'\rangle$$

The term on the right replies with an empty transition and it is easy to show that, for all v and v', $(*\,[x,x']\,(\mathtt{n}?\langle x,x'\rangle.\,\mathtt{n}!\langle x,x'\rangle) \mid \mathtt{n}!\langle v,v'\rangle)$ and $(\mathbf{0} \mid \mathtt{n}!\langle v,v'\rangle)$ are weak bisimilar.

4 COWS

COWS is obtained by enriching μCOWS as follows:

$$s ::= \ \ldots \quad | \quad \mathbf{kill}(k) \quad | \quad \{\!|s|\!\} \quad | \quad [k]\,s$$

Besides the sets of values and variables, we also use the set of *(killer) labels* (ranged over by $k, k', \ldots$). Notably, expressions do not include killer labels (that, hence, are *not* communicable). Delimitation now is a binder also for killer labels and, differently from the scope of names and variables, that of killer labels *cannot* be extended. A COWS term is *closed* if it does not contain free variables and killer labels.

Informally, execution of a *kill* activity $\mathbf{kill}(k)$ causes termination of all parallel terms inside the enclosing $[k]$, which stops the killing effect. Critical activities can be protected from the effect of a forced termination by using the *protection* operator $\{\!|s|\!\}$. E.g., $K \triangleq \mathtt{n}!\langle v\rangle \mid [k]\,(\,[x]\,\mathtt{n}?\langle x\rangle.s \mid \{\!|\,\mathtt{m}!\langle v'\rangle\,|\!\} \mid \mathbf{kill}(k)\,)$ can perform a *computational step* $\dagger$ by executing the kill activity and evolving to $(\mathtt{n}!\langle v\rangle \mid [k]\,(\,\mathit{halt}([x]\,\mathtt{n}?\langle x\rangle.s) \mid \mathit{halt}(\{\!|\,\mathtt{m}!\langle v'\rangle|\!\})\,)) \equiv (\mathtt{n}!\langle v\rangle \mid [k]\,\{\!|\,\mathtt{m}!\langle v'\rangle\,|\!\})$, where function $\mathit{halt}(\_)$, given a term s, returns the term obtained by only retaining the protected activities inside s. COWS's priority mechanism assigns greatest priority to kill activities so that they pre-empt all other activities inside the enclosing killer label's delimitation. For example, in the term K above communication along $\mathtt{n}$ and activity $\mathtt{m}!\langle v'\rangle$ are blocked until the kill activity has been performed. We refer to [13] for a complete account of COWS semantics.

When considering observational semantics for COWS we soon discover that $\sim_\mu$ is not preserved by those contexts forcing termination of the activities in the hole. For example, $\emptyset \ \sim_\mu\ \{\!|\emptyset|\!\}$ trivially holds. However, the COWS context $[k]\,(\mathbf{kill}(k) \mid [\![\cdot]\!])$ can tell the two terms apart. Indeed, $[k]\,(\mathbf{kill}(k) \mid \emptyset) \sim_\mu [k]\,(\mathbf{kill}(k) \mid \{\!|\emptyset|\!\})$ does not hold since, after execution of the kill activity (that has highest priority), we would get $\mathbf{0} \sim_\mu \{\!|\emptyset|\!\}$ which is trivially false. Therefore, $\sim_\mu$ is not a congruence for COWS.

Open barbed bisimilarity, namely $\simeq$, is by definition closed under all contexts and its definition only needs to be tuned for considering also †-transitions in the 'Computation closure' (the exact definition is in [13]). Instead, labelled bisimilarity must explicitly take care of the effects of execution of kill activities and of occurrences of the protection operator. These differences w.r.t. to Definition 2 are highlighted with a gray background.

Definition 5 (Labelled bisimilarity). *A names-indexed family of relations $\{\mathcal{R}_N\}_N$ is a* labelled bisimulation *if $s_1 \mathcal{R}_N s_2$ then $halt(s_1)\,\mathcal{R}_N\,halt(s_2)$ and if $s_1 \xrightarrow{\alpha} s_1'$, where* bu$(\alpha)$ *are fresh, then: we replace clauses 1.(b) and 4 in Definition 2 as follows (other clauses are identical).*

1. (b) $|\bar{x}|=|\bar{w}|$ and $\exists\, s_2' : s_2 \xrightarrow{\emptyset} s_2'$ and $\forall\, \bar{v}$ s.t. $\mathcal{M}(\bar{x}, \bar{v}) = \sigma$ and

$$\text{noConf}(s_2, \mathtt{n}, \bar{w}\cdot\sigma, |\bar{x}|) \;:\; s_1'\cdot\sigma\,\mathcal{R}_N\,(s_2' \mid \mathtt{n}!\bar{v}) \;\; or \;\; s_1'\cdot\sigma\,\mathcal{R}_N\,(s_2' \mid \{\!|\mathtt{n}!\bar{v}|\!\})$$

4. if $\alpha = \emptyset$, $\alpha = \dagger$ or $\alpha = \mathtt{n}\,\emptyset\,\ell\,\bar{v}$, where $\ell \neq |\bar{v}|$, then $\exists\, s_2' : s_2 \xrightarrow{\alpha} s_2'$ and $s_1'\,\mathcal{R}_N\,s_2'$

*Two closed terms s_1 and s_2 are $\mathcal{N}$-*bisimilar*, written $s_1 \sim^{\mathcal{N}} s_2$, if $s_1 \mathcal{R}_N s_2$ for some $\mathcal{R}_N$ in a labelled bisimulation. They are* labelled bisimilar*, written $s_1 \sim s_2$, if they are $\emptyset$-bisimilar. $\sim^{\mathcal{N}}$ is called $\mathcal{N}$-bisimilarity, while $\sim$ is called labelled bisimilarity.*

halt-closure takes into account kill activities performed by contexts ($halt(s)$ gets the same effect as of plunging s within the context $[k]\,(\mathbf{kill}(k) \mid [\![\cdot]\!])$), while clause 4 takes into account kill activities that are active within the considered terms. Clause 1.(b) considers that if a closed term s performs a transition labelled by $\mathtt{n} \lhd [\bar{m}]\,\bar{v}$, then s contains an invoke of the form $\mathtt{n}!\bar{\epsilon}$, with $[\![\bar{\epsilon}]\!] = \bar{v}$, which can be either protected or not.

Theorem 3. (1) $\sim$ *is a congruence for* COWS *closed terms; and* (2) $\sim$ *and* $\simeq$ *coincide.*

Extension to the weak case is standard (definitions of the bisimilarities are in [13]). Again, results of congruence and coincidence hold.

We finish studying the relationship between the specifications of the Morra service introduced in Section 2. We can prove that (1) $\not\sim$ (2). Indeed, (1) can perform two transitions labelled by $evens\bullet throw \rhd [x_{id}, y_p, y_{num}]\,\langle x_{id}, y_p, y_{num}\rangle$ and by $odds\bullet throw \rhd [x_p, x_{num}]\,\langle first, x_p, x_{num}\rangle$ and, because of application of substitutions $\{x_{id} \mapsto first, y_p \mapsto cbB, y_{num} \mapsto 1\}$ and $\{x_p \mapsto cbA, x_{num} \mapsto 2\}$, evolve to $(\,cbA \bullet res!\langle first, win(2,1,1)\rangle \mid cbB \bullet res!\langle first, win(2,1,0)\rangle\,)$. (2) can properly simulate the above transitions but it can only evolve to $\{\!|\,cbA \bullet res!\langle first, w\rangle \mid cbB \bullet res!\langle first, l\rangle\,|\!\}$. Of course, the latter term behaves differently from the former one in presence of kill activities. In fact, given the context $\mathbb{C} \triangleq [k']\,(\,[\mathtt{n}]\,(\mathtt{n}!\langle\rangle \mid \mathtt{n}?\langle\rangle.\,\mathbf{kill}(k')) \mid [\![\cdot]\!]\,)$, we have that $\mathbb{C}[\![(1)]\!] \not\sim \mathbb{C}[\![(2)]\!]$.

If we modify the last two invokes in the high-level specification (1) as follows:

$$\begin{aligned} * \,[x_{id}, x_p, x_{num}, y_p, y_{num}]\,(\,& odds \bullet throw?\langle x_{id}, x_p, x_{num}\rangle \mid evens \bullet throw?\langle x_{id}, y_p, y_{num}\rangle \\ & \mid \{\!|\, x_p \bullet res!\langle x_{id}, win(x_{num}, y_{num}, 1)\rangle \mid y_p \bullet res!\langle x_{id}, win(x_{num}, y_{num}, 0)\rangle\,|\!\}\,) \end{aligned} \quad (4)$$

the problem persists, because (4) after the first two transitions always provides a response, while (2) could fail to provide a response in presence of kill activities. Instead, a term bisimilar to (4) can be obtained by replacing M in (2) by the following term:

$$[x_{id}, x_p, x_{num}, y_p, y_{num}]\,(\,odds \bullet throw?\langle x_{id}, x_p, x_{num}\rangle \mid evens \bullet throw?\langle x_{id}, y_p, y_{num}\rangle \mid \{\!|\,[k]\,(\ldots)\,|\!\}\,)$$

5 Related work

Many process calculi with priority have been proposed in the literature [4]. In previous proposals, dynamic priorities are basically used to model scheduling approaches and real-time aspects (see e.g. [2, 5]) while in COWS they are used for coordination, as well as for orchestration, purposes. For example, in the service *5F* of Section 2 they enable implementing a sort of 'default' behaviour, that returns *err* when a throw is not admissible. To the best of our knowledge (see also [4]), the interplay between dynamic priorities and local pre-emption, and their impact on semantic theories of processes have never been explored before. A termination construct similar to COWS's **kill** activity has been introduced in [6] in the setting of a distributed pi-calculus, where it is used to model the failure of a node. Instead, COWS's **kill** activity (in conjunction with protection and delimitation) is more flexible since it permits terminating parallel activities in a more selective way. [9, 3] use labelled bisimilarities to prove compliance between service implementations and specifications for process calculi based on an explicit notion of *session* rather than on correlation. COWS's barbed bisimilarities follow the approach of open barbed bisimilarities [8, 12] rather than that of barbed congruence [11], i.e. quantification over contexts occurs recursively inside the definition of bisimilarity. COWS's priority mechanisms make some receive actions observable (which leads to a novel notion of observation that refines the purely asynchronous one [1, 7]), and require specific conditions on the labelled bisimilarities for these to be congruences.

References

1. R.M. Amadio, I. Castellani, D. Sangiorgi. On Bisimulations for the Asynchronous pi-Calculus. *Theoretical Computer Science*, 195(2):291–324, 1998.
2. G. Bhat, R. Cleaveland, G. Lüttgen. A Practical Approach to Implementing Real-Time Semantics. *Annals of Software Engineering*, 7:127–155, 1999.
3. R. Bruni, et al. Multiparty sessions in soc. *COORDINATION*, LNCS 5052, pp. 67–82, 2008.
4. R. Cleaveland, G. Lüttgen, V. Natarajan. Priorities in process algebra. *Handbook of Process Algebra*, pp. 391–424, 2001.
5. H. Fecher. A Real-Time Process Algebra with Open Intervals and Maximal Progress. *Nordic Journal of Computing*, 8(3):346–365, 2001.
6. A. Francalanza, M. Hennessy. A theory for observational fault tolerance. *Journal of Logic and Algebraic Programming*, 73(1-2):22–50, 2007.
7. K. Honda, M. Tokoro. An Object Calculus for Asynchronous Communication. *ECOOP*, LNCS 512, pp. 133–147, 1991.
8. K. Honda, N. Yoshida. On Reduction-Based Process Semantics. *Theoretical Computer Science*, 151(2):437–486, 1995.
9. I. Lanese, et al. Disciplining Orchestration and Conversation in Service-Oriented Computing. *SEFM*, pp. 305–314, IEEE, 2007.
10. A. Lapadula, R. Pugliese, F. Tiezzi. A calculus for orchestration of web services. *ESOP*, LNCS 4421, pp. 33-47, 2007.
11. R. Milner, D. Sangiorgi. Barbed Bisimulation. *ICALP*, LNCS 623, pp. 685–695, 1992.
12. D. Sangiorgi, D. Walker. On Barbed Equivalences in pi-Calculus. *CONCUR*, LNCS 2154, pp. 292–304, 2001.
13. F. Tiezzi. Specification and Analysis of Service-Oriented Applications. PhD Thesis, Univ. Florence, 2009. `http://rap.dsi.unifi.it/cows/theses/tiezzi_phdthesis.pdf`.