# A Calculus of Bounded Capacities*

F. Barbanera[1], M. Bugliesi[2], M. Dezani-Ciancaglini[3], and V. Sassone[4]

[1] Università di Catania, Viale A.Doria 6, 95125 Catania (Italy)
barba@dmi.unict.it
[2] Università "Cà Foscari", Via Torino 155, 30170 Venezia (Italy)
michele@dsi.unive.it
[3] Università di Torino, Corso Svizzera 185, 10149 Torino (Italy)
dezani@di.unito.it
[4] University of Sussex, Falmer, Brighton BN1 9RH UK
vs@susx.ac.uk

**Abstract.** Resource control has attracted increasing interest in foundational research on distributed systems. This paper focuses on space control and develops an analysis of space usage in the context of an ambient-like calculus with bounded capacities and weighed processes, where migration and activation require space. A type system complements the dynamics of the calculus by providing static guarantees that the intended capacity bounds are preserved throughout the computation.

## Introduction

Emerging computing paradigms, such as Global Computing and Ambient Intelligence, envision scenarios where mobile devices travel across domains and networks boundaries. Current examples include smart cards, embedded devices (e.g. in cars), mobile phones, PDAs, and the list keeps growing. The notion of third-party resource usage will raise to a central role, as roaming entities will need to borrow resources from host networks and, in turn, provide guarantees of bounded resource usage. This is the context of the present paper, which focuses on *space consumption* and *capacity bound* awareness.

Resource control, in diverse incarnations, has recently been the focus of foundational research. Topics considered include the ability to read from and to write to a channel [22], the control of the location of channel names [29], the guarantee that distributed agents will access resources only when allowed to do so [12, 21, 2, 10, 11]. Specific work on the certification of bounds on resource consumption include [13], which introduces a notion of resource type representing an abstract unit of space, and uses a linear type system to guarantee linear space consumption; [7] where quantitative bounds on time usage are enforced using a typed assembly language; and [15], which puts forward a general formulation of resource usage analysis.

* F. Barbanera is partially supported by MIUR project NAPOLI, M. Bugliesi by EU-FET project 'MyThS' IST-2001-32617, and by MIUR project MEFISTO, M. Dezani-Ciancaglini by EU-FET project DART IST-2001-33477, and by MIUR Projects COMETA and McTati, V. Sassone by EU-FET project 'MIKADO' IST-2001-32222. The funding bodies are not responsible for any use that might be made of the results presented here.

We elect to formulate our analysis of space control in an ambient-like calculus, BoCa, because the notion of ambient mobility is a natural vehicle to address the intended application domain. Relevant references to related work in this context include [8], which presents a calculus in which resources may be moved across locations provided suitable space is available at the target location; [27], which uses typing systems to control resource usage and consumption; and [6], which uses static techniques to analyse the behaviour of finite control processes, i.e., those with bounded capabilities for ambient allocation and output creation.

*Overview.* BoCa relies on a physical, yet abstract, notion of "resource unit" defined in terms of a new process constructor, noted $\_$ (read "*slot*"), which evolves out of the homonym notion of [8]. A slot may be interpreted as a unit of computation space to be allocated to running processes and migrating ambients. To exemplify, the configuration

$$P \mid \underbrace{\_ \mid \ldots \mid \_}_{\mathsf{k}\text{ times}}$$

represents a system which is running process $P$ and which has $\mathsf{k}$ resource units available for $P$ to spawn – i.e. activate – new subprocesses and to accept migrating agents willing to enter. In both cases, the activation of the new components is predicated to the presence of suitable resources: only processes and agents requiring cumulatively no more than $\mathsf{k}$ units may be activated on the system. As a consequence, process activation and agent migration involve a protocol to "negotiate" the use of resources with the enclosing, resp. receiving, context (possibly competing with other processes).

For migrating agents this is accounted for by associating each agent with a tag representing the space required for activation at the target context, as in $a^{\mathsf{k}}[\,P\,]$. A notion of well-formedness will ensure that $\mathsf{k}$ provides a safe estimate of the space needed by $a[\,P\,]$; namely, the number of resource units allocated to $P$. Correspondingly, the negotiation protocol for mobility is represented formally by the following reductions (where $\_^{\mathsf{k}}$ is short for $\_ \mid \ldots \mid \_$, $\mathsf{k}$ times):

$$a^{\mathsf{k}}[\,\mathbf{in}\,b.P \mid Q\,] \mid b[\,\_^{\mathsf{k}} \mid R\,] \quad \searrow \quad \_^{\mathsf{k}} \mid b[\,a^{\mathsf{k}}[\,P \mid Q\,] \mid R\,]$$

$$\_^{\mathsf{k}} \mid b[\,P \mid a^{\mathsf{k}}[\,\mathbf{out}\,b.Q \mid R\,]\,] \quad \searrow \quad a^{\mathsf{k}}[\,Q \mid R\,] \mid b[\,P \mid \_^{\mathsf{k}}\,]$$

In both cases, the migrating agent releases the space required for its computation at the source site and gets corresponding space at the target context. Notice that the reductions construe $\_$ both as a representation of the physical space available at the locations of the system, and as a particular new kind of co-capability.

Making the weight of an ambient depend explicitly on its contents allows a clean and simple treatment of the open capability: opening does not require resources, as those needed to allocate the contents are exactly those taken by enclosing ambient.

$$\mathbf{open}\,a.P \mid a[\,\overline{\mathbf{open}}.Q \mid R\,] \quad \searrow \quad P \mid Q \mid R$$

Notice that in order for these reductions to provide the intended semantics of resource negotiation, it is crucial that the redexes are well-formed (in the sense discussed above). Accordingly, the dynamics of ambient mobility is inherently dependent on the

assumption that all migrating agents are well-formed. As we shall discuss, this assumption is central to the definition of behavioural equivalence as well.

Resource management and consumption do not concern exclusively mobility, as *all* processes need and use space. In particular, since "spawning" processes requires resources, process replication must be controlled so as to guard against processes that may consume an *infinite* amount of resources. The action of spawning a new process is made explicit in BoCa by introducing a new process construct, the prefix $\mathsf{k}\triangleright$, whose semantics is defined by the following reduction:

$$\mathsf{k}\triangleright P \mid \blacksquare^{\mathsf{k}} \quad \searrow \quad P$$

Here $\mathsf{k}\triangleright P$ is a "spawner" which launches $P$ provided that the local context is ready to allocate enough fresh resources for the activation. The tag $\mathsf{k}$ represents the "activation cost" for process $P$, viz. its weight, while $\mathsf{k}\triangleright P$, the "frozen code" of $P$, weighs 0: here, again, the hypothesis of well-formedness of terms is critical to make sense of the spawning protocol. The adoption of an explicit spawning operator allows us to delegate to the "spawner" the responsibility of resource control in the mechanism for process replication. In particular, we restrict the replication primitive "**!**" to 0-weight processes only. We can then rely on the usual congruence rule that identifies $!P$ with $!P \mid P$, and use $!(\mathsf{k}\triangleright P)$ to realise a resource-aware version of replication. This results in a system which separates process *duplication* from process *activation*, and so allows a fine analysis of resource consumption in computation.

The definition of BoCa is completed by two constructs that provide for the dynamic allocation of resources. In our approach resources are not "created" from the void, but rather acquired dynamically – in fact, transferred – from the context, again as a result of a negotiation.

$$a^{\mathsf{k}+1}[\,\mathbf{put}.P \mid \blacksquare \mid Q\,] \mid b^{\mathsf{h}}[\,\mathbf{get}\,a.R \mid S\,] \quad \searrow \quad a^{\mathsf{k}}[\,P \mid Q\,] \mid b^{\mathsf{h}+1}[\,R \mid \blacksquare \mid S\,]$$

$$\mathbf{put}^{\downarrow}.P \mid \blacksquare \mid a^{\mathsf{k}}[\,\mathbf{get}^{\uparrow}.Q \mid R\,] \quad \searrow \quad P \mid a^{\mathsf{k}+1}[\,\blacksquare \mid Q \mid R\,]$$

Resource transfer is realised as a two-way synchronisation in which a context offers some of its resource units to any enclosed or sibling ambient that makes a corresponding request. The effect of the transfer is reflected in the tags that describe the resources allocated to the receiving ambients. We formalise slot transfers only between siblings and from father to child. As we shall see, transfers across siblings make it possible to encode a notion of private resource, while transfer from child to parent can easily be encoded in terms of the existing constructs.

*Contributions and main results.* Resource allocation in BoCa is controlled by a system of capacity types that guarantees capacity bounds on computational ambients. In this system, ambient types allow resource control policies to be specified by imposing capacity bounds that control the movement and the spawning of processes inside ambients. Then, the typing system enables us to certify statically the absence of under-/over-flows, potentially arising from an uncontrolled use of the capabilities for dynamic space allocation.

Additional, and finer, control on the dynamics of space allocation/deallocation is provided by means of a naming mechanism for slots. We introduce it with a refined version of the calculus in which the notion of private resource is made primitive, rather than

encoded. As we will show, the new mechanisms provide the calculus with a rich and tractable algebraic theory. The semantics theory of the refined calculus is supported by a labelled transition system, yielding a bisimulation congruence adequate with respect to barbed congruence. Besides enabling powerful co-inductive characterizations of process equivalences, the labelled transition system yields an effective tool for contextual reasoning on process behavior. More specifically, it enables a formal representation of *open systems*, in which processes may acquire resources and space from their enclosing context.

Our approach in the definition of BoCa is typical of a way to couple language design with type analysis. This coupling is critical in frameworks like Global Computing, where it is ultimately unrealistic to assume acquaintance with all the entities which may in the future interact with us, as it is usually done for standard type systems. The openness of the network and its very dynamic nature deny us any substantial form of global knowledge. Therefore, syntactic constructs must be introduced to support the static analysis, as e.g., our "negotiation" protocols. In our system, the possibility of dynamically checking particular space constraints is a consequence of the explicit presence of the primitive $\_$.

*Structure of the paper.* In §1 we give the formal description of BoCa and illustrate it with a few examples. In §2 we introduce the system of capacity types, and prove it sound. In §3 we introduce the refined calculus and study its its operational and behavioural semantics (based on barbed congruence). We conclude with final remarks in §4. A separate appendix describes the labelled transition system for the calculus.

A preliminary version of this paper appeared in [1].

## 1 The calculus BoCa

The calculus is a conservative extension of the Ambient Calculus. We presuppose two mutually disjoint sets: $\mathcal{N}$ of names, and $\mathcal{V}$ of variables. The set $\mathcal{V}$ is ranged over by letters at the end of the alphabet, typically $x,y,z$, while $a,b,c,d,n,m$ range over $\mathcal{N}$. Finally, $\mathsf{h},\mathsf{k}$ and other letters in the same font denote integers. The syntax of the (monadic) calculus is defined below, with $W$ an exchange type as introduced in §2.

**Definition 1 (Preterms and Terms).** The set of process *preterms* is defined by the following productions (where we assume $\mathsf{k} \geq 0$):

$$
\begin{array}{lll}
\textit{Processes} & P & ::= \_ \mid \mathbf{0} \mid \pi.P \mid P \mid P \mid M^{\mathsf{k}}[\,P\,] \mid !\pi.P \mid (\boldsymbol{\nu} a : W)P \\
\textit{Capabilities} & C & ::= \mathbf{in}\, M \mid \mathbf{out}\, M \mid \mathbf{open}\, M \mid \overline{\mathbf{open}} \mid \mathbf{get}\, M \mid \mathbf{get}^{\uparrow} \mid \mathbf{put} \mid \mathbf{put}^{\downarrow} \\
\textit{Messages} & M & ::= a \in \mathcal{N} \mid x \in \mathcal{V} \mid C \mid M.M \\
\textit{Prefixes} & \pi & ::= M \mid (x : W) \mid \langle M \rangle \mid \mathsf{k} \triangleright
\end{array}
$$

A (well-formed) *term* $P$ is a preterm such that $w(P) \neq \bot$, where $w : \mathit{Processes} \rightharpoonup \omega$ is the partial *weight* function defined as follows:

$$\begin{aligned} w(\mathbf{0}) = 0 \qquad w(\_) = 1 \qquad w(P \mid Q) &= w(P) + w(Q) \\ w(M.P) = w((x:W)P) = w(\langle M\rangle P) &= w((\boldsymbol{\nu} a : W)P) = w(P) \\ w(a^{\mathsf{k}}[\,P\,]) &= \text{if } w(P) \text{ is } \mathsf{k} \text{ then } \mathsf{k} \text{ else } \bot \\ w(\mathsf{k} \triangleright P) &= \text{if } w(P) \text{ is } \mathsf{k} \text{ then } 0 \text{ else } \bot \\ w(\mathbf{!}P) &= \text{if } w(P) \text{ is } 0 \text{ then } 0 \text{ else } \bot \end{aligned}$$

We use the standard notational conventions for ambient calculi. In particular, we write $(x:W)P$ and $\langle M\rangle P$ for $(x:W).P$ and $\langle M\rangle.P$, respectively, and similarly $\mathsf{k} \triangleright P$ to denote $\mathsf{k} \triangleright .P$. We omit types when not relevant; we write $a[\,P\,]$ instead of $a^{\mathsf{k}}[\,P\,]$ when the value of $\mathsf{k}$ does not matter. In this regard, note that in $a^{\mathsf{k}}[\,P\,]$ the weight tag $\mathsf{k}$ refers to the ambient process $a[\,P\,]$ and *not* to the name $a$. We use $\_^{\mathsf{k}}$ as a shorthand for $\_ \mid \ldots \mid \_$ ($\mathsf{k}$ times) and $C^{\mathsf{k}}$ as a shorthand for $C.\ldots.C$ (again $\mathsf{k}$ times). Following a well-known approach, we restrict replication to prefixed processes, as this allows for a simplified treatment in the labelled transition system.

### 1.1 Reduction

The dynamics of the calculus is defined as usual in terms of structural congruence and reduction (cf. Figure 1). Unlike other calculi, however, in BoCa both relations are only defined for proper terms, a fact we will leave implicit in the rest of the presentation.

The reduction relation $\searrow$ formalizes the intuitions discussed in the introduction; we denote with $\searrow_*$ the reflexive and transitive closure of $\searrow$. Structural congruence is essentially standard. The assumption of well-formedness is central to both relations. In particular, the congruence $\mathbf{!}P \equiv P \mid \mathbf{!}P$ only holds with $P$ a proper term of weight 0. Thus, to duplicate arbitrary processes we need to first "freeze" them under $\mathsf{k}\triangleright$, i.e. we decompose arbitrary duplication into "template replication" and "process activation." Notice that replication only applies to prefixed processes, a restriction that is technically convenient and that does not involve any significant loss of expressive power.

A few remarks are in order on the form of the transfer capabilities. The **put** capability (among siblings) does not name the target ambient, as is the case for the dual capability **get**. We select this particular combination because it is the most liberal one for which our results hold. Of course, more stringent notions are possible, as e.g. when both partners in a synchronisation use each other's names. Adopting any of these would not change the nature of the calculus and preserve, mutatis mutandis, the validity of our results. In particular, the current choice makes it easy and natural to express interesting programming examples (cf. the memory management in §1.2), and protocols: e.g., it enables us to provide simple encoding of named (and private) resources allocated for spawning (cf. §3). Secondly, a new protocol is easily derived for transferring resources "upwards" from children to parents using the following pair of dual put and get.

$$\mathbf{get}^{\downarrow} a.P \triangleq (\boldsymbol{\nu} m)(\mathbf{open}\, m.P \mid m^0[\,\mathbf{get}\, a.\overline{\mathbf{open}}\,]), \quad \text{and} \quad \mathbf{put}^{\uparrow} \triangleq \mathbf{put}$$

**Fig. 1** Structural Congruence and Reduction

**Structural Congruence**: $(|, \mathbf{0})$ is a commutative monoid.

$$
\begin{array}{lcllcl}
(\boldsymbol{\nu} a)(P \mid Q) & \equiv & (\boldsymbol{\nu} a)P \mid Q \;\; (a \notin \mathrm{fn}(Q)) & \qquad (\boldsymbol{\nu} a)a^0[\,\mathbf{0}\,] & \equiv & \mathbf{0} \\
(\boldsymbol{\nu} a)\mathbf{0} & \equiv & \mathbf{0} & (\boldsymbol{\nu} a)(\boldsymbol{\nu} b)P & \equiv & (\boldsymbol{\nu} b)(\boldsymbol{\nu} a)P \\
!P & \equiv & P \mid !P & a[\,(\boldsymbol{\nu} b)P\,] & \equiv & (\boldsymbol{\nu} b)a[\,P\,] \;\; (a \neq b)
\end{array}
$$

**Reduction**: $E ::= \{\cdot\} \mid E \mid P \mid (\boldsymbol{\nu} m)E \mid m^{\mathsf{k}}[\,E\,]$ is an evaluation context

$$
\begin{array}{lrcl}
(\textsc{enter}) & a^{\mathsf{k}}[\,\mathbf{in}\, b.P \mid Q\,] \mid b[\,\_^{\mathsf{k}} \mid R\,] & \searrow & \_^{\mathsf{k}} \mid b[\,a^{\mathsf{k}}[\,P \mid Q\,] \mid R\,] \\
(\textsc{exit}) & \_^{\mathsf{k}} \mid b[\,P \mid a^{\mathsf{k}}[\,\mathbf{out}\, b.Q \mid R\,]\,] & \searrow & a^{\mathsf{k}}[\,Q \mid R\,] \mid b[\,P \mid \_^{\mathsf{k}}\,] \\
(\textsc{open}) & \mathbf{open}\, a.P \mid a[\,\overline{\mathbf{open}}.Q \mid R\,] & \searrow & P \mid Q \mid R \\
(\textsc{getS}) & a^{\mathsf{k}+1}[\,\mathbf{put}.P \mid \_ \mid Q\,] \mid b^{\mathsf{h}}[\,\mathbf{get}\, a.R \mid S\,] & \searrow & a^{\mathsf{k}}[\,P \mid Q\,] \mid b^{\mathsf{h}+1}[\,R \mid \_ \mid S\,] \\
(\textsc{getD}) & \mathbf{put}^{\downarrow}.P \mid \_ \mid a^{\mathsf{k}}[\,\mathbf{get}^{\uparrow}.Q \mid R\,] & \searrow & P \mid a^{\mathsf{k}+1}[\,\_ \mid Q \mid R\,] \\
(\textsc{spawn}) & \mathsf{k} \triangleright P \mid \_^{\mathsf{k}} & \searrow & P \\
(\textsc{exchange}) & (x:W)P \mid \langle M \rangle Q & \searrow & P\{x := M\} \mid Q \\
(\textsc{Struct}) & P \equiv P' \quad P' \searrow Q' \quad Q' \equiv Q & \Longrightarrow & P \searrow Q \\
(\textsc{Context}) & P \searrow Q & \Longrightarrow & E\{P\} \searrow E\{Q\}
\end{array}
$$

Transfers affect the amount of resources allocated at different nesting levels in a system. We delegate to the type system of §2 to control that no nesting level suffers from resource over- or under-flows. The reduction semantics itself guarantees that the global amount of resources is preserved, as it can be proved by an inspection of the reduction rules.

**Proposition 1 (Resource preservation).** *If $w(P) \neq \bot$, and $P \searrow_* Q$, then $w(Q) = w(P)$.*

Two remarks about the above proposition are worthwhile. First, resource preservation is a distinctive property of *closed* systems; in open systems, instead, a process may acquire new resources from the environment, or transfer resources to the environment, by exercising the **put** and **get** capabilities. Secondly, the fact that the global weight of a process is invariant through reduction does not imply that the amount of resources available for computation also is invariant.

Indeed, our notion of slot is an economical way to convey the three different concepts of a resource being *free*, *allocated*, or *wasted*, according to the context in which $\_$ occurs during the computation. Unguarded slots, as in $a[\,\_ \mid P\,]$, represent free resources available for spawning or movement at a given nesting level; guarded slots, like $M.\_$, represent allocated resources, which may be released and become free; and unreachable slots, like $(\boldsymbol{\nu} a)\mathbf{in}\; a.\_^{\mathsf{k}}$ or $(\boldsymbol{\nu} a)a^{\mathsf{k}}[\,\_^{\mathsf{k}}\,]$, represent wasted resources that will never be released.

Computation changes the state of resources in the expected ways: allocated resources may be freed, as in $\mathbf{open}\ a.\_ \mid a[\,P\,] \searrow \_ \mid P$; free resources may be allocated, as in $\_ \mid 1 \triangleright M.\_ \searrow M.\_$, or wasted as in $\mathbf{put}^{\downarrow} \mid \_ \mid (\boldsymbol{\nu} a)a[\,\mathbf{get}^{\uparrow}\,] \searrow (\boldsymbol{\nu} a)a[\,\_\,]$. No further transition is possible for a wasted resource: in particular, it may never become free, and re-allocated. Accordingly, while the global amount of resources is invariant through reduction, as stated in Proposition 1, the computation of a process does in general consume resources and leaves a possibly decreased, certainly non-increased amount of free and allocated resources. We will return on a more precise analysis resource usage based on the characterization we just outlined in §3.

### 1.2 Examples

We illustrate the calculus with examples and encodings of systems that require usage and control of space.

*Recovering Mobile Ambients.* The Ambient Calculus [5] is straightforwardly embedded in (an untyped version of) BoCa: it suffices to insert a process $!\overline{\mathbf{open}}$ in all ambients. The relevant clauses of the embedding are as follows:

$$[\![a[\,P\,]]\!] \triangleq a^0[\,!\overline{\mathbf{open}} \mid [\![P]\!]\,], \quad [\![(\boldsymbol{\nu} a)P]\!] \triangleq (\boldsymbol{\nu} a)[\![P]\!]$$

and the remaining ones are derived similarly; clearly all resulting processes weigh 0.

*Parent-child swap.* Given that ambients may have non null weights, in BoCa this swap is possible only in case the father and child nodes have the same weight. We present it for example in the case of weight 1. Notice the use of the primitives for child to father slot transfer we defined in §1.

$$b^1[\,\mathbf{get}^{\downarrow}a\,.\,\mathbf{put}\,.\,\mathbf{in}\ a\,.\,\mathbf{get}^{\uparrow} \mid a^1[\,\mathbf{put}^{\uparrow}\,.\,\mathbf{out}\ b\,.\,\mathbf{get}\ b\,.\,\mathbf{put}^{\downarrow} \mid \_\,]\,] \searrow_* a^1[\,b^1[\,\_\,]\,]$$

*Ambient renaming.* We can represent in BoCa a form of ambient self-renaming capability. First, define $\mathbf{spawn}^{\mathsf{k}}\ P\ \mathbf{outside}\ a \triangleq exp^0[\,\mathbf{out}\ a\,.\,\overline{\mathbf{open}}\,.\,\mathsf{k} \triangleright P\,]$ and then use it to define

$$a\ \mathbf{be}^{\mathsf{k}}b\,.\,P \triangleq \mathbf{spawn}^{\mathsf{k}}\ (b^{\mathsf{k}}[\,\_^{\mathsf{k}} \mid \mathbf{open}\ a\,])\ \mathbf{outside}\ a \mid \mathbf{in}\ b\,.\,\overline{\mathbf{open}}\,.\,P$$

Since $\mathbf{open}\ exp \mid \_^{\mathsf{k}} \mid a^{\mathsf{h}}[\,\mathbf{spawn}^{\mathsf{k}}\ (b^{\mathsf{k}}[\,P\,] \mid Q)\ \mathbf{outside}\ a\,] \searrow_* b^{\mathsf{k}}[\,P\,] \mid a^{\mathsf{h}}[\,Q\,]$ where $\mathsf{k}$, $\mathsf{h}$ are the weights of $P$ and $Q$, respectively, we get

$$a^{\mathsf{k}}[\,a\ \mathbf{be}^{\mathsf{k}}b\,.\,P \mid R\,] \mid \_^{\mathsf{k}} \mid \mathbf{open}\ exp \searrow_* b^{\mathsf{k}}[\,P \mid R\,] \mid \_^{\mathsf{k}}$$

So, an ambient needs to *borrow* space from its parent in order to rename itself. We conjecture that renaming cannot be obtained otherwise.

*A memory module.* A user can take slots from a memory module MEM_MOD using MALLOC and release them back to MEM_MOD after their use.

$$\begin{aligned}
\text{MEM_MOD} &\triangleq mem[\,_^{256MB} \mid \overbrace{\mathbf{open}\ m \mid \ldots \mid \mathbf{open}\ m}^{256MB}\,] \\
\text{MALLOC} &\triangleq m[\,\mathbf{out}\ u\,.\,\mathbf{in}\ mem\,.\,\overline{\mathbf{open}}\,.\,\mathbf{put}\,.\,\mathbf{get}\ u\,.\,\mathbf{open}\ m\,] \\
\text{USER} &\triangleq u[\,\ldots.\text{MALLOC} \mid \ldots.\mathbf{get}\ mem \ldots \mathbf{put} \mid \ldots\,]
\end{aligned}$$

*A cab trip.* As a further example, we give a new version of the the cab trip protocol from [27], formulated in our calculus. A customer sends a request for a cab, which then arrives and takes the customer to his destination. The use of slots here enables us to model very naturally the constraint that only one passenger (or actually any fixed number of them) may occupy a cab.

$$
\begin{array}{ll}
CALL(from,client) & \triangleq \\
\quad call^1[\,\mathbf{out}\ client.\mathbf{out}\ from.\mathbf{in}\ cab.\overline{\mathbf{open}}.\mathbf{in}\ from.(load^0[\,\mathbf{out}\ cab.\mathbf{in}\ client.\overline{\mathbf{open}}\,]\mid \blacksquare)\,] & \\
TRIP(from,to,client) & \triangleq trip^0[\,\mathbf{out}\ client.\overline{\mathbf{open}}.\mathbf{out}\ from.\mathbf{in}\ to.done^0[\,\mathbf{in}\ client.\overline{\mathbf{open}}\,]\,] \\
CLIENT(from,to) & \triangleq (\boldsymbol{\nu} c)c^1[CALL(from,c)\mid \mathbf{open}\ load.\mathbf{in}\ cab.TRIP(from,to,c) \\
 & \qquad\qquad \mid \mathbf{open}\ done.\mathbf{out}\ cab.bye^0[\,\mathbf{out}\ c.\mathbf{in}\ cab.\overline{\mathbf{open}}.\mathbf{out}\ to\,]] \\
CAB & \triangleq cab^1[\,\blacksquare \mid !(\mathbf{open}\ call.\mathbf{open}\ trip.\mathbf{open}\ bye)\,] \\
SITE(i) & \triangleq site_i[\,CLIENT(site_i,site_j)\mid CLIENT(site_i,site_l)\mid\cdots\mid\blacksquare\mid\blacksquare\mid\cdots\,] \\
CITY & \triangleq city[\,CAB\mid CAB\mid\cdots\mid\cdots\mid SITE(1)\mid\cdots\mid SITE(n)\mid\blacksquare\mid\blacksquare\mid\cdots\,]
\end{array}
$$

The fact that only one slot is available in *cab* together with the weight 1 of both *call* and *client* prevents the cab to carry more than one call and/or more than one client. Moreover this encoding limits also the space in each site and in the whole city.

Comparing with [27], we notice that we can deal with the cab's space satisfactorily with no need for 3-way synchronisations. Unfortunately, as already observed in [27], this encoding may lead to unwanted behaviours, since there is no way of preventing a client to enter a cab different from that called and/or the ambient *bye* to enter a cab different from that the client has left. We discuss these and related issues in further detail below.

### 1.3 Discussion

In its present definition, the calculus provides a simple, yet effective, framework for expressing resource usage and consumption. On the other hand, as observed above, it is less effective to express and enforce *policies* for resource *allocation*, and their *distribution* to distinct, possibly, competing components.

Policies for resource allocation should provide safeguards against denial-of-service threats, based on the ability of misbehaved agents to attack a host by repeated space transfer requests that could overfill the target host or leave it with no space to spawn its local processes. Similarly, policies for resource distribution should be able to express protocols in which a given resource unit is selectively allocated to a specific agent, and protected against unintended use. To illustrate, consider the following term (and assume it well-formed):

$$a^1[\,\mathbf{in}\ b.P\,]\mid b[\,1\triangleright Q\mid\blacksquare\mid d[\,c^1[\,\mathbf{out}\ d.R\,]\,]\,]$$

Three agents are competing for the resource unit in ambient *b*: ambients *a* and *c*, which would use it for their move, and the local spawner inside ambient *b*. While the race between *a* and *c* may be acceptable – the resource unit may be allocated by *b* to any

migrating agent – it would also be desirable for *b* to reserve resources for internal use, i.e. for spawning new processes.

In the remainder of the paper we attack both problems. The system of capacity types in §2 provides static guarantees that the resources available at a given site remain within the intended bounds. The mechanism for slot naming, in §3, yields new and effective primitives for the selective distribution of resources, and for protecting processes against the presence of races for resource acquisition.

## 2 Bounding Resources – by Typing

As outlined above, the type system provides static guarantees for a simple behavioural property, namely the absence of space under- and over-flows arising as a result of transfers during the computation. To deal with this satisfactorily, we need to take into account that transfer capabilities can be acquired by way of exchanges. The type of a capability will hence have to express how it affects the space of the ambient in which it can be performed.

### 2.1 Capacity Types

We use $\mathcal{Z}$ to denote the set of integers, and note $\mathcal{Z}^+$ and $\mathcal{Z}^-$ the sets of non-negative and non-positive integers respectively. We define the following domains:

$$\begin{array}{rl} \textit{Intervals} & \iota \in \mathfrak{I} \triangleq \{[n,N] \mid n,N \in \mathcal{Z}^+,\ n \leq N\} \\ \textit{Effects} & \varepsilon \in \mathcal{E} \triangleq \{(d,i) \mid d \in \mathcal{Z}^-,\ i \in \mathcal{Z}^+\} \\ \textit{Thread Effects} & \phi \in \Phi \triangleq \mathcal{E} \to \mathcal{E} \end{array}$$

Intervals and effects are ordered in the expected way, namely: $[n,N] \leq [n',N']$ when $n' \leq n$ and $N \leq N'$ and $(d,i) \leq (d',i')$ when $d' \leq d$ and $i \leq i'$ . It is also convenient to define the component-wise sum operator for effects: $(d,i)+(d',i') = (d+d',i+i')$, and lift it to $\Phi$ pointwise: $\phi_1 + \phi_2 = \lambda\varepsilon.\phi_1(\varepsilon) + \phi_2(\varepsilon)$.

The syntax of types is defined by the following productions:

$$\begin{array}{rl} \textit{Message Types} & W ::= \mathit{Amb}\langle\iota,\varepsilon,\chi\rangle \mid \mathit{Cap}\langle\phi,\chi\rangle \\ \textit{Exchange Types} & \chi ::= \mathit{Shh} \mid W \\ \textit{Process Types} & \Pi ::= \mathit{Proc}\langle\varepsilon,\chi\rangle \end{array}$$

Type $\mathit{Proc}\langle\varepsilon,\chi\rangle$ is the type of processes with $\varepsilon$ effects and $\chi$ exchanges. Specifically, for a process $P$ of type $\mathit{Proc}\langle(d,i),\chi\rangle$, the effect $(d,i)$ bounds the number of slots delivered ($|d|$) and acquired (i) by $P$ as the cumulative result of exercising $P$'s transfer capabilities.

Type $\mathit{Amb}\langle\iota,\varepsilon,\chi\rangle$ is the type of ambients with weight ranging in $\iota$, and enclosing processes with $\varepsilon$ effects and $\chi$ exchanges. As in companion type systems, values that can be exchanged include ambients and (paths of) capabilities, while the type *Shh* indicates no exchange. As for capability types, $\mathit{Cap}\langle\phi,\chi\rangle$ is the type of (paths of) capabilities

which, when exercised, unleash processes with $\chi$ exchanges, and compose the effect of the unleashed process with the thread effect $\phi$. The functional domain of thread effects helps compute the composition of effects. In brief, thread effects accumulate the results from **get**'s and **put**'s, and compose these with the effects unleashed by occurrences of **open**.

We introduce the following combinators (functions in $\Phi$) to define the thread effects of the **put**, **get** and **open** capabilities.

$$\begin{aligned} \mathsf{Put} &= \lambda(\mathsf{d},\mathsf{i}).(\mathsf{d}-1,\mathsf{max}(0,\mathsf{i}-1)) \\ \mathsf{Get} &= \lambda(\mathsf{d},\mathsf{i}).(\mathsf{min}(0,\mathsf{d}+1),\mathsf{i}+1) \\ \mathsf{Open}(\varepsilon) &= \lambda(\mathsf{d},\mathsf{i}).(\varepsilon+(\mathsf{d},\mathsf{i})) \end{aligned}$$

The intuition is as follows. A **put** that prefixes a process $P$ with cumulative effect $(\mathsf{d},\mathsf{i})$, contributes to a "shift" in that effect of one unit. The effect of a **get** capability is dual. To illustrate, the thread effect $\varepsilon$ associated with $P = \textbf{put.put.get}\ a$ is computed as follows, where we use function composition in standard order (i.e. $f \circ g(x) = f(g(x))$):

$$\varepsilon \;=\; (\mathsf{Put} \circ \mathsf{Put} \circ \mathsf{Get})((0,0)) \;=\; (-2,0).$$

The intuition about an open capability is similar, but subtler, as the effect of opening an ambient is, essentially, the effect of the process unleashed by the open: in **open** $n.P$, the process unleashed by **open** $n$ runs in parallel with $P$. Consequently, open has an additive import in the computation of the effect. To motivate, assume that $n : Amb\langle\iota,\varepsilon,\chi\rangle$. Opening $n$ unleashes the enclosed process in parallel to the process $P$. To compute the resulting effect we may rely on the effect $\varepsilon$ declared by $n$ to bound the effect of the unleashed process: that effect is then added to the effect of the continuation $P$. Specifically, if $P$ has effect $\varepsilon'$, the composite effect of **open** $n.P$ is computed as $\mathsf{Open}(\varepsilon)(\varepsilon') = \varepsilon+\varepsilon'$.

### 2.2 The typing rules

The typing rules are collected in Figures 2 and 3, where we denote with $\mathsf{id}_\Phi$ the identity element in the domain $\Phi$.

The rules in Figure 2 derive judgements $\Gamma \vdash M : W$ for well-typed messages. The environment $\Gamma$ is a set of assumptions either of the shape $a : Amb\langle\iota,\varepsilon,\chi\rangle$ with $a \in \mathcal{N}$ or of the shape $x : W$ with $x \in \mathcal{V}$, where all names and variables are distinct. The rules draw on the intuitions we gave earlier. Notice, in particular, that the capabilities **in**, **out** and the cocapability $\overline{\textbf{open}}$ have no effect, as reflected by the use $\mathsf{id}_\Phi$ in their type. The same is true also of the the co-capability $\textbf{put}^{\downarrow}$. In fact, by means of the superscript $\mathsf{k}$ in $a^{\mathsf{k}}[\,P\,]$ we can record the actual weight of the ambient (cf. reduction rule (GETD)). This implies that the weight of an ambient in which $\textbf{put}^{\downarrow}$ is executed does not change: the ambient loses a slot, but the weight of one of its sub-ambients increases.

The rules in Figure 3 derive judgements $\Gamma \vdash P : Proc\langle\varepsilon,\chi\rangle$ for well-typed processes. An inspection of the typing rules shows that any well-typed process is also well-formed (in the sense of Definition 1). We let $0_{\mathcal{E}}$ denote the null effect $(0,0)$: thus, rules (**0**) and (▬) simply state that the inhert process and the slot form have no effects. Rule (*prefix*)

**Fig. 2** Good Messages

$$\frac{}{\Gamma, M : W \vdash M : W}\ (axiom) \qquad \frac{\Gamma \vdash M : Cap\langle \phi, \chi\rangle \quad \Gamma \vdash M' : Cap\langle \phi', \chi\rangle}{\Gamma \vdash M.M' : Cap\langle \phi \circ \phi', \chi\rangle}\ (path)$$

$$\frac{\Gamma \vdash M : Amb\langle -,-,-\rangle}{\Gamma \vdash \mathbf{get}\ M : Cap\langle \mathsf{Get}, \chi\rangle}\ (\mathbf{get}\ M) \qquad \frac{}{\Gamma \vdash \mathbf{put} : Cap\langle \mathsf{Put}, \chi\rangle}\ (\mathbf{put})$$

$$\frac{}{\Gamma \vdash \mathbf{get}^{\uparrow} : Cap\langle \mathsf{Get}, \chi\rangle}\ (\mathbf{get}^{\uparrow}) \qquad \frac{}{\Gamma \vdash \mathbf{put}^{\downarrow} : Cap\langle \mathsf{id}_{\Phi}, \chi\rangle}\ (\mathbf{put}^{\downarrow})$$

$$\frac{\Gamma \vdash M : Amb\langle -,-,-\rangle}{\Gamma \vdash \mathbf{in}\ M : Cap\langle \mathsf{id}_{\Phi}, \chi\rangle}\ (\mathbf{in}\ M) \qquad \frac{\Gamma \vdash M : Amb\langle -,-,-\rangle}{\Gamma \vdash \mathbf{out}\ M : Cap\langle \mathsf{id}_{\Phi}, \chi\rangle}\ (\mathbf{out}\ M)$$

$$\frac{\Gamma \vdash M : Amb\langle -, \varepsilon, \chi\rangle}{\Gamma \vdash \mathbf{open}\ M : Cap\langle \mathsf{Open}(\varepsilon), \chi\rangle}\ (\mathbf{open}\ M) \qquad \frac{}{\Gamma \vdash \overline{\mathbf{open}} : Cap\langle \mathsf{id}_{\Phi}, \chi\rangle}\ (\overline{\mathbf{open}})$$

computes the effects of prefixes, by applying the thread effect of the capability to the effect of the process. Rule (*par*) adds up the effects of two parallel threads, while the constructs for input, output and restriction do not have any effect.

Rule (*amb*) governs the formation of ambient processes. The declared weight $\mathsf{k}$ of the ambient must reflect the weight of the enclosed process. Two further conditions ensure $(i)$ that $\mathsf{k}$ modified by the effect $(\mathsf{d},\mathsf{i})$ of the enclosed process lies within the interval $[\mathsf{n},\mathsf{N}]$ declared by the ambient type, and $(ii)$ that effect $\varepsilon$ declared by the ambient type is a sound approximation for the effects released by opening the ambient itself. The condition that ensures $(i)$ is simply $[\max(\mathsf{k}+\mathsf{d},0),\mathsf{k}+\mathsf{i}] \leq [\mathsf{n},\mathsf{N}]$, where the use of $\max(\mathsf{k}+\mathsf{d},0)$ is justified by observing that the weight of an ambient may never grow negative as a result of the enclosed process exercising **put** capabilities. To motivate the condition that ensures $(ii)$, first observe that opening an ambient which encloses a process with effect $(\mathsf{d},\mathsf{i})$ may only release effects $\varepsilon \leq (\mathsf{d}-\mathsf{i},\mathsf{i}-\mathsf{d})$. The lower bound arises in a situation in which the ambient is opened right after the enclosed process has completed its $\mathsf{i}$ **get** 's and is thus left with $|\mathsf{d}-\mathsf{i}|$ **put**'s unleashed in the opening context. Dually, the upper bound arises when the ambient is opened right after the enclosed process has completed its $|\mathsf{d}|$ **put**'s, and is left with $\mathsf{i}-\mathsf{d}$ **get** 's. On the other hand, we also know that the maximum increasing effect released by opening ambients with weight ranging in $[\mathsf{n},\mathsf{N}]$ is $\mathsf{N}-\mathsf{n}$. Collectively, these two observations justify the condition $(\mathsf{d}-\mathsf{i},\min(\mathsf{N}-\mathsf{n},\mathsf{i}-\mathsf{d})) \leq \varepsilon$ in rule (*amb*).

In rule (*spawn*), the effect of $\mathsf{k} \triangleright P$ is the same as that of the reduct $P$. Finally, to prevent the effects of duplicated processes to add up beyond control, with unpredictable consequences, rule (*bang*) enforces duplicated processes to have null effects.

**Fig. 3** Good Processes

$$\frac{}{\Gamma \vdash \_ : Proc\langle 0_{\mathcal{E}}, \chi\rangle}\ (\_) \qquad \frac{}{\Gamma \vdash \mathbf{0} : Proc\langle 0_{\mathcal{E}}, \chi\rangle}\ (\mathbf{0})$$

$$\frac{\Gamma \vdash M : Cap\langle \phi, \chi\rangle \quad \Gamma \vdash P : Proc\langle \varepsilon, \chi\rangle}{\Gamma \vdash M.P : Proc\langle \phi(\varepsilon), \chi\rangle}\ (prefix)$$

$$\frac{\Gamma \vdash P : Proc\langle \varepsilon, \chi\rangle \quad \Gamma \vdash Q : Proc\langle \varepsilon', \chi\rangle}{\Gamma \vdash P \mid Q : Proc\langle \varepsilon + \varepsilon', \chi\rangle}\ (par) \qquad \frac{\Gamma, x : W \vdash P : Proc\langle \varepsilon, W\rangle}{\Gamma \vdash (x : W)P : Proc\langle \varepsilon, W\rangle}\ (input)$$

$$\frac{\Gamma \vdash M : W \quad \Gamma \vdash P : Proc\langle \varepsilon, W\rangle}{\Gamma \vdash \langle M\rangle P : Proc\langle \varepsilon, W\rangle}\ (output) \qquad \frac{\Gamma, a : Amb\langle \iota, \varepsilon, \chi\rangle \vdash P : Proc\langle \varepsilon', \chi'\rangle}{\Gamma \vdash (\boldsymbol{\nu} a : Amb\langle \iota, \varepsilon, \chi\rangle)P : Proc\langle \varepsilon', \chi'\rangle}\ (new)$$

$$\frac{\begin{array}{ll}\Gamma \vdash M : Amb\langle [\mathsf{n}, \mathsf{N}], \varepsilon, \chi'\rangle & [\max(\mathsf{k}+\mathsf{d}, 0), \mathsf{k}+\mathsf{i}] \leq [\mathsf{n}, \mathsf{N}] \\ \Gamma \vdash P : Proc\langle (\mathsf{d}, \mathsf{i}), \chi'\rangle \qquad w(P) = \mathsf{k} & (\mathsf{d}-\mathsf{i}, \min(\mathsf{N}-\mathsf{n}, \mathsf{i}-\mathsf{d})) \leq \varepsilon\end{array}}{\Gamma \vdash M^{\mathsf{k}}[P] : Proc\langle 0_{\mathcal{E}}, \chi\rangle}\ (amb)$$

$$\frac{\Gamma \vdash P : Proc\langle 0_{\mathcal{E}}, \chi\rangle \quad w(P) = \mathsf{k}}{\Gamma \vdash \mathsf{k} \triangleright P : Proc\langle 0_{\mathcal{E}}, \chi\rangle}\ (spawn) \qquad \frac{\Gamma \vdash \pi.P : Proc\langle 0_{\mathcal{E}}, \chi\rangle \quad w(\pi.P) = 0}{\Gamma \vdash \mathbf{!}\pi.P : Proc\langle 0_{\mathcal{E}}, \chi\rangle}\ (bang)$$

A first property of the type system is that all typable preterms are proper, hence well-formed, terms. In addition, the following result complements Proposition 1 and shows that capacity bounds on ambients are preserved during computations, while the processes' ability to shrink or expand reduces.

**Theorem 1 (Subject Reduction).** *Assume* $\Gamma \vdash P : Proc\langle \varepsilon, \chi\rangle$ *and* $P \searrow_* Q$. *Then* $\Gamma \vdash Q : Proc\langle \varepsilon', \chi\rangle$ *for some* $\varepsilon' \leq \varepsilon$.

*Proof.* The proof is by induction on the length of the reduction from $P$ to $Q$. In the inductive case, the reasoning is by a cases analysis of the reduction in question. The interesting cases are those for rules (GETS) and (GETD). Below, we give the case of (GETS), the treatment of (GETD) being similar. Also, we give a proof disregarding the exchange components of our capacity types, as they are irrelevant for our argument.

Let $\mathcal{D}$ be a derivation for the redex of a (GETS) reduction:

$$\Gamma \vdash a^{\mathsf{k}+1}[\,\mathbf{put}.P \mid \_ \mid Q\,] \mid b^{\mathsf{h}}[\,\mathbf{get}\,a.R \mid S\,] : Proc\langle 0_{\mathcal{E}}\rangle$$

An inspection of the typing rules – (*amb*) and (*par*) – shows that the type of the redex must indeed be of the form given. Inside $\mathcal{D}$ there must be judgements of the shape:

$$\Gamma \vdash a : Amb\langle \iota, \varepsilon\rangle, \quad \Gamma \vdash P : Proc\langle (\mathsf{d}, \mathsf{i})\rangle, \quad \Gamma \vdash Q : Proc\langle (\mathsf{d}', \mathsf{i}')\rangle$$

for some $\iota, \varepsilon, i, i', d, d'$. By rules (**put**) and (*prefix*), $\Gamma \vdash P : Proc\langle (d, i) \rangle$ implies $\Gamma \vdash \mathbf{put}.P : Proc\langle (d-1, \max(i-1,0)) \rangle$. Then the application of rule (*amb*) for deriving

$$\Gamma \vdash a^{k+1}[\, \mathbf{put}.P \mid \_ \mid Q \,] : Proc\langle 0_{\mathcal{E}} \rangle$$

has the following conditions:

$$[\max(k+1+d-1+d', 0), k+1+\max(i-1,0)+i'] \leq \iota$$
$$(d-1+d'-\max(i-1,0)-i', \min(N-n, \max(i-1,0)+i'-d+1-d') \leq \varepsilon,$$

where $\iota = [n, N]$. The conditions required to apply rule (*amb*) for the derivation of the judgement $\Gamma \vdash a^k[\, P \mid Q \,] : Proc\langle 0_{\mathcal{E}} \rangle$ are

$$[\max(k+d+d', 0), k+i+i'] \leq \iota$$
$$(d+d'-i-i', \min(N-n, i+i'-d-d') \leq \varepsilon.$$

These conditions can be shown to derive from the previous ones by easy algebraic manipulations. With a similar reasoning one checks that the judgement $\Gamma \vdash b^{h+1}[\, \_ \mid R \mid S \,] : Proc\langle 0_{\mathcal{E}} \rangle$ is derivable. From this, we can conclude that the judgement $\Gamma \vdash a^k[\, P \mid Q \,] \mid b^{h+1}[\, \_ \mid R \mid S \,] : Proc\langle 0_{\mathcal{E}} \rangle$ is derivable, as desired.

It follows as a direct corollary that no ambient may be subject to under/over-flows during the computation of a process.

**Theorem 2 (Absence of under/over-flow).** *Assume $\Gamma \vdash P : Proc\langle \varepsilon, \chi \rangle$ and let $P \searrow_* Q$. If $a : Amb\langle [n, N], -, - \rangle \in \Gamma$, then, for any subterm of $Q$ of the form $a^k[\, R \,]$, not in the scope of a binder for $a$, we have $n \leq k \leq N$.*

The proviso "not in the scope of a binder for $a$" is needed since the environment $\Gamma$ could contain an assumption for the name $a$ which has no connection whatsoever with the type of a bound occurrence of the same name $a$. In particular these two types could have very different weight ranges.

### 2.3 Typed Examples

All the examples in §1.2 that do not use both the open and the transfer capabilities are easily seen to typecheck. For instance, in the *cab trip* protocol, one verifies that all the processes typecheck, by taking $W_1 = Amb\langle [1,1], 0_{\mathcal{E}} \rangle$, $W_0 = Amb\langle [0,0], 0_{\mathcal{E}} \rangle$, by declaring the $c$ with $W_1$ and assuming the following types for the free names $call : W_1$, $cab : W_1$, $trip : W_0$, $load : W_0$, $done : W_0$, $bye : W_0$.

Below we illustrate the typing system at work with a typed version of the memory module of Section 1.2. We start with the *malloc* ambient

$$\text{MALLOC} \triangleq m[\, \mathbf{out}\, u.\mathbf{in}\, mem.\overline{\mathbf{open}}.\mathbf{put}.\mathbf{get}\, u.\mathbf{open}\, m \,]$$

Since there are no exchanges, we give the typing annotation and derivation disregarding the exchange component from the types. Let $P_{malloc}$ denote thread enclosed within the

ambient $m$. If we let $m : Amb\langle[0,0],(-1,0)\rangle \in \Gamma$, an inspection of the typing rules for capabilities and paths shows that the following typing is derivable for any ambient type assigned to *mem*:

$$\Gamma \vdash \mathbf{out}\ u\,.\,\mathbf{in}\ mem\,.\,\overline{\mathbf{open}}\,.\,\mathbf{put}\,.\,\mathbf{get}\ u\,.\,\mathbf{open}\ m : Cap\langle \mathsf{Put} \circ \mathsf{Get} \circ \lambda\varepsilon.((-1,0)+\varepsilon)\rangle$$

From this one derives $\Gamma \vdash P_{malloc} : Proc\langle(-1,0)\rangle$, which gives $\Gamma \vdash \text{MALLOC} : Proc\langle 0_{\mathcal{E}}\rangle$. As to the memory module itself, it is a routine check to verify that the process

$$\text{MEM\_MOD} \triangleq mem[\,\_^{256MB} \mid \mathbf{open}\ m \mid \ldots \mid \mathbf{open}\ m\,]$$

typechecks with $m : Amb\langle[0,0],(-1,0)\rangle$, $mem : Amb\langle[0,256MB],(-256MB,256MB)\rangle$.

### 2.4 Inferring Effects

We have shown that the type system gives enough flexibility to typecheck interesting protocols. On the other hand, the structure of the types is somehow complex: ideally, one would like to be able to use ambient types only to specify resource-related policies, without having to worry about the effects. Thus, for instance, one would specify the memory module protocol by simply declaring $m : Amb[0,0]$ and $mem : Amb[0,256MB]$ and leave it to the type system to infer the effect-related part of the types required to ensure that such bounds are indeed complied with.

We look at effect inference below. To ease the presentation, we give an inference system for the combinatorial subset of the calculus and disregard communications (and hence we do not consider the set of variables $\mathcal{V}$).
The ambient types give only the range $\iota$ for the weights of processes inside them, i.e. they have the shape: $Amb^{-}\langle\iota\rangle$. For each name $a$ in $\mathcal{N}$ we introduce two integer variables $\mathtt{d}_a$, $\mathtt{i}_a$. So, in a sense, a pair $(\mathtt{d}_a,\mathtt{i}_a)$ can be looked at as an *effect variable* for the ambient $a$. Our inference system determines (by a set of inequalities constraints) the instances of effect variables which make a process typeable.

In the inference system an effect is no longer a pair of integers, but a pair of expressions $(e,e')$ where $e,e' \in EXP$ and $EXP$ is defined by:

$$EXP ::= \mathtt{d}_a \mid \mathtt{i}_a \mid \mathtt{n} \in \mathcal{Z} \mid EXP + EXP \mid EXP - EXP \mid \mathsf{min}(EXP,EXP) \mid \mathsf{max}(EXP,EXP)$$

with $a \in \mathcal{N}$. Of course this implies that thread effects are now functions on expressions, that is:

$$\textit{Effects} \qquad \varepsilon \in \mathcal{E} \triangleq \{(e,e') \mid e,e' \in EXP\}$$

$$\textit{Thread Effects} \qquad \phi \in \Phi \triangleq \mathcal{E} \to \mathcal{E}$$

The rules for messages in our inference system are those in Figure 2, but for rule (**open** $M$) which becomes:

$$\frac{\Gamma \vdash a : Amb^{-}\langle - \rangle}{\Gamma \vdash \mathbf{open}\ a : Cap\langle \mathsf{Open}(\mathtt{d}_a,\mathtt{i}_a)\rangle}\ (\widehat{\mathbf{open}\ a})$$

**Fig. 4** Effect Inference

$$\frac{}{\Gamma \vdash \blacksquare : Proc\langle 0_{\mathcal{E}} \rangle \Downarrow \emptyset}\ \widehat{(\blacksquare)} \qquad \frac{}{\Gamma \vdash \mathbf{0} : Proc\langle 0_{\mathcal{E}} \rangle \Downarrow \emptyset}\ \widehat{(\mathbf{0})}$$

$$\frac{\Gamma \vdash M : Cap\langle \phi \rangle \quad \Gamma \vdash P : Proc\langle \varepsilon \rangle \Downarrow \Delta}{\Gamma \vdash M.P : Proc\langle \phi(\varepsilon) \rangle \Downarrow \Delta}\ \widehat{(prefix)}$$

$$\frac{\Gamma \vdash P : Proc\langle \varepsilon \rangle \Downarrow \Delta \quad \Gamma \vdash Q : Proc\langle \varepsilon' \rangle \Downarrow \Delta'}{\Gamma \vdash P \mid Q : Proc\langle \varepsilon + \varepsilon' \rangle \Downarrow \Delta \cup \Delta'}\ \widehat{(par)}$$

$$\frac{\Gamma, a : Amb^{-}\langle \iota \rangle \vdash P : Proc\langle \varepsilon \rangle \Downarrow \Delta}{\Gamma \vdash (\boldsymbol{\nu} a : Amb^{-}\langle \iota \rangle) P : Proc\langle \varepsilon \rangle \Downarrow \Delta}\ \widehat{(new)}$$

$$\frac{\Gamma \vdash a : Amb^{-}\langle [\mathsf{n}, \mathsf{N}] \rangle \quad \Gamma \vdash P : Proc\langle (e, e') \rangle \Downarrow \Delta \quad w(P) = \mathsf{k}}{\Gamma \vdash a^{\mathsf{k}}[P] : Proc\langle 0_{\mathcal{E}} \rangle \Downarrow \Delta \cup \left\{ \begin{array}{l} \mathsf{n} \leq \max(\mathsf{k} + e, 0),\ \mathsf{k} + e' \leq \mathsf{N}, \\ \mathsf{d}_a \leq e - e',\ \min(\mathsf{N} - \mathsf{n}, e' - e) \leq \mathtt{i}_a \end{array} \right\}}\ \widehat{(amb)}$$

$$\frac{\Gamma \vdash P : Proc\langle \varepsilon \rangle \Downarrow \Delta \quad w(P) = \mathsf{k}}{\Gamma \vdash \mathsf{k} \triangleright P : Proc\langle \varepsilon \rangle \Downarrow \Delta}\ \widehat{(spawn)} \qquad \frac{\Gamma \vdash \pi.P : Proc\langle 0_{\mathcal{E}} \rangle \Downarrow \Delta \quad w(\pi.P) = 0}{\Gamma \vdash !\pi.P : Proc\langle 0_{\mathcal{E}} \rangle \Downarrow \Delta}\ \widehat{(bang)}$$

The inference rules are in Figure 4 and the judgements they derive have the shape $\Gamma \vdash P : Proc\langle \varepsilon \rangle \Downarrow \Delta$. This means that in our system, beside providing a process $P$ with a process type $Proc\langle \varepsilon \rangle$, we produce for $P$ also a set of inequality constraints $\Delta$, whose satisfiability implies the typability of the process in our former system. The constraints are of the form $e \leq e'$ where the (integer) unknowns are of the form $\mathtt{d}_a$, $\mathtt{i}_a$.

The only rules that contribute to the formation of the constraint set are $\widehat{(par)}$ and $\widehat{(amb)}$. Rule $\widehat{(par)}$ simply joins the sets of constraints of two parallel processes. It is only by means of rule $\widehat{(amb)}$ that we really produce new constraints. These ones are simply a different way of stating, in our inference setting, the conditions of the $(amb)$ rule of the type system. In our inference system, instead of checking a condition, we simply record it as a constraint whose satisfiability shall be checked at the end of the inference process.

In order to check the satisfiability of a set of constraints, we can transform them in a standard way, by adding two fresh variables with suitable conditions[1] for each $\max()$ or $\min()$ subexpression, obtaining an integer linear programming problem (see for exam-

[1] Let $\mathsf{M}$ be the maximum weight in the types of ambients occurring in the process $P$ and $\mathsf{h}$ the total number of **put** and **get** capabilities occurring in $P$. Then $\mathsf{M} + 2\mathsf{h}$ is an upper bound for all integers which can occur in the set of constraints for $P$. We replace each $\min(e, e')$

ple [20]). Notice that we are only interested in the satisfiability (non emptyness of the solution space) of the resulting linear programming problem, that is we have no particular target function to optimize (but the auxiliary ones introduced when we get rid of the $\mathsf{max}()$ or $\mathsf{min}()$ subexpressions). In fact, given a process $P$, what matters is whether the *effect variables* can be instantiated enabling the typing of $P$. An optimization problem arises only in case we need to insert $P$ in a particular context. For instance, if $P$ is in a context where one of its ambients $a$ has to be opened, we could try to minimise the effect of the type of $a$ in order to have more chances the capacity of the opening ambient not be exceeded. On the contrary, if $P$ is in a context where one of its ambients $a$ must contain a process $Q$, we need to maximize the effect of the type of $a$ in order to allow more freedom in the choice of the process $Q$.

It is worth noticing that any set of constraints can be extended with the following two, making the set of possible solutions always finite: $-\mathsf{h} \leq \mathsf{d}_a \leq 0$ and $0 \leq \mathsf{i}_a \leq \mathsf{h}$, where $\mathsf{h}$ is the total number of **put** and **get** capabilities occurring in $P$. The soundness of these additional constraints can be checked by a simple inspection of the rules.

The type inference illustrated above can easily be adapted to infer the ranges $\iota$ of ambients (instead of using environments): it suffices to consider for each name $a$ other two integer variables $\mathsf{n}_a$, $\mathsf{N}_a$ and replace them respectively to $\mathsf{n},\mathsf{N}$ in rule $(amb)$. So in this case we obtain a set of constraints where also $\mathsf{n}_a$ and $\mathsf{N}_a$ are unknown: if the constraints can be satisfied then there is also a solution such that $0 \leq \mathsf{n}_a \leq \mathsf{k} \leq \mathsf{N}_a$, where $\mathsf{k}$ is the weight of the process $P$. Furthermore, notice that the judgements $\vdash P : Proc\langle\varepsilon\rangle \Downarrow \Delta$ are built in a compositional way, and then $Proc\langle\varepsilon\rangle \Downarrow \Delta$ can be seen as the *principal typings* of the processes $P$ in the sense of [28].

## 3 Controlling Resource Races – BoCa Revisited

In this section we extend the calculus with further, term-level, mechanisms to complement the typing system in providing for a richer and stronger control over the dynamics of space allocation and distribution.

As we observed in §1.3, a basic requirement is to provide for the ability by an ambient to reserve resources for internal use, i.e. for spawning new processes. In fact, reserving private space for spawning is possible with the current primitives, by encoding a notion of "named resource". This can be accomplished by defining:

$$\blacksquare_a^{\mathsf{k}} \triangleq a[\,\mathbf{put}^{\mathsf{k}} \mid \blacksquare^{\mathsf{k}}\,], \quad \text{and} \quad \mathsf{k} \triangleright (a,P) \triangleq (\boldsymbol{\nu} n)(n[\,(\mathbf{get}\ a)^{\mathsf{k}}.\mathsf{k} \triangleright \overline{\mathbf{open}}.P\,] \mid \mathbf{open}\ n)$$

Then, assuming $w(P) = \mathsf{k}$, one has $(\boldsymbol{\nu} a)(\blacksquare_a^{\mathsf{k}} \mid \mathsf{k} \triangleright (a,P)) \cong P$, as desired. It is also possible to encode a form of "resource renaming", by defining:

$$\{x/y\}.P \triangleq (\nu n)(n[\,\mathbf{get}\ y.\mathbf{put}.\overline{\mathbf{open}}\,] \mid x[\,\mathbf{get}\ n.\mathbf{put}\,] \mid \mathbf{open}\ n.P)$$

subexpression by a fresh variable $t$ and we add the following conditions:

$$\begin{aligned} 0 &\leq e - t \ \leq (\mathsf{M}+2\mathsf{h})y \\ 0 &\leq e' - t \leq (\mathsf{M}+2\mathsf{h})(1-y) \end{aligned}$$

with $y \in \{0,1\}$ and fresh. We deal with $\mathsf{max}()$ subexpressions similarly.

Then, a $y$-resource can be turned in to an $x$-resource: $\{x/y\}.P \mid \__y \searrow_* P \mid \__x$.

Encoding a similar form of named, and reserved, resources for mobility is subtler. On the one hand, it is not difficult to encode a construct for reserving a $x$-slot for ambients named $x$. For example, ambients $a$ and $b$ may agree on the following protocol to reserve a private slot for the move of $a$ into $b$. If we want to use the space in ambient $b$ for moving $a$ we can write the process:

$$(\boldsymbol{\nu} p,q)(p[\,\mathbf{in}\ b\,.\,\mathbf{get}\ q\,.\,1 \triangleright \overline{\mathbf{open}}\,.\,a^1[\,\_\,]\,] \mid b[\,P \mid q[\,\_ \mid \mathbf{put}\,] \mid \mathbf{open}\ p\,])$$

On the other hand, defining a mechanism to release a named resource to the context from which it has been received is more complex, as it amounts to releasing a resource with the *same* name it was allocated to. This can be simulated loosely with the current primitives, by providing a mechanism whereby a migrating ambient releases an anonymous slot, which is then renamed by the context that is in control of it. The problem is that such a mechanism of releasing and renaming lacks the *atomicity* required to guard against unexpected races for the released resource. Indeed, we believe that such atomic mechanisms for named resources can not be defined in the current calculus.

### 3.1 Named resources and their semantics

To counter the lack of atomicity discussed above, we enrich the calculus with named resources as primitive notions, and tailor the constructs for mobility, transfer and spawning accordingly. Resource units come now always with a tag, as in $\__\eta$, where $\eta \in \mathcal{N} + \mathcal{V} + \{*\}$ is the unit name. To make the new calculus a conservative extension of the one presented in §1, we make provision for a special tag '*', to be associated with *anonymous* units: any process can be spawned on an anonymous slot, as well any ambient can be moved on it. In addition, we extend the structure of the transfer capabilities, as well as the construct for spawning and ambient as shown in the productions below, which replace the corresponding ones in §1.

| | | |
|---|---|---|
| *Processes* | $P ::= \__\eta \mid M[\,P\,]_\eta \mid \ldots$ | as in Section 1 |
| *Capabilities* | $C ::= \mathbf{get} M_\eta \mid \mathbf{get}^{\uparrow}_\eta \mid \ldots$ | as in Section 1 |
| *Messages* | $M ::= \ldots$ | as in Section 1 |
| *Prefixes* | $\pi ::= \mathsf{k} \triangleright_\eta \mid \ldots$ | as in Section 1 |

Again, a (well-formed) *term* is a preterm such that in any subterm of the form $a^{\mathsf{k}}[P]$ or $\mathsf{k} \triangleright_\eta P$, $P$ has weight $\mathsf{k}$. The weight of a process can be computed by rules similar to those of Section 1. The anonymous slots $\__*$ will be often denoted simply as $\_$.

The dynamics of the refined calculus is again defined by means of structural congruence and reduction. Structural congruence is exactly as in Figure 1, the top-level reductions are defined in Figure 5.
Ambients acquire tags, as in $a[\,P\,]_\eta$, as they move. Initially, we may interpret each occurrence of an ambient $a[\,P\,]$ as denoting the tagged occurrence $a[\,P\,]_a$. To complete a move an ambient $a$ must be granted an anonymous resource or an $a$-resource. The migrating ambient releases a resource under the name that it was assigned upon the

**Fig. 5** Top-level reductions with named units

The reductions for ambient opening and exchanges are as in Figure 1, and the rules (ENTER) and (EXIT) have $\rho, \eta \in \{a, \star\}$ as side condition. The omitted subscripts on ambients are meant to remain unchanged by the reductions.

$$
\begin{array}{lrcl}
(\text{ENTER}) & a^k[\,\mathbf{in}\,b.P \mid Q\,]_\rho \mid b[\,\blacksquare^k_\eta \mid R\,] & \searrow & \blacksquare^k_\rho \mid b[\,a^k[\,P \mid Q\,]_\eta \mid R\,] \\
(\text{EXIT}) & \blacksquare^k_\eta \mid b[\,P \mid a^k[\,\mathbf{out}\,b.Q \mid R\,]_\rho\,] & \searrow & a^k[\,Q \mid R\,]_\eta \mid b[\,P \mid \blacksquare^k_\rho\,] \\
(\text{GETS}) & b^{h+1}[\,\mathbf{put}.P \mid \blacksquare_\eta \mid Q\,] \mid a^k[\,\mathbf{get}\,b_\eta.R \mid S\,] & \searrow & b^h[\,P \mid Q\,] \mid a^{k+1}[\,R \mid \blacksquare_\eta \mid S\,] \\
(\text{GETU}) & \mathbf{put}^\downarrow.P \mid \blacksquare_\eta \mid a^{k+1}[\,\mathbf{get}^\uparrow_\eta.Q \mid R\,] & \searrow & P \mid a^k[\,\blacksquare_\eta \mid Q \mid R\,] \\
(\text{SPAWN}) & \mathsf{k} \triangleright_\eta P \mid \blacksquare^k_\eta & \searrow & P
\end{array}
$$

move (as recorded in the tag associated with the ambient construct): this solves the problem we discussed above. Also note that the dynamics of mobility guarantees the invariant that in $a[\,P\,]_\eta$ one has $\eta \in \{a, *\}$.

The reductions for the transfer capabilities are the natural extensions of the original reductions of §1. Here, in addition to naming the target ambient, the **get** capabilities also indicate the name of the unit they request. The choice of the primitives enables natural forms of scope extrusion for the names of resources. Consider the following system:

$$S \triangleq n[\,(\boldsymbol{\nu} a)(\mathbf{put}.P \mid \blacksquare_a \mid p[\,\mathbf{out}\,n.\mathbf{in}\,m.\overline{\mathbf{open}}.\mathbf{get}\,n_a\,])\,] \mid m[\,\mathbf{open}\,p.Q\,]$$

Here, the private resource enclosed within ambient $n$ is communicated to ambient $m$, as $S \searrow_* (\boldsymbol{\nu} a)(n[\,P\,] \mid m[\,Q \mid \blacksquare_a\,])$.

Finally, the new semantics of spawning acts as expected, by associating the process to be spawned with a specific set of resources.

These definitions suggest a natural form of resource renaming (or rebinding), noted $\{\eta/\rho\}_\mathsf{k}$ with the following operational semantics.

$$\{\eta/\rho\}_\mathsf{k}.P \mid \blacksquare^k_\rho \searrow P \mid \blacksquare^k_\eta$$

Notice that this is a dangerous capability, since it allows processes to give particular names to anonymous slots, and for instance put in place possible malicious behaviours to make all public resources their own: $!\{y/*\}$. This suggests that in many situations one ought to restrict $\mathsf{k}\triangleright_\eta$ to $\eta \in \mathcal{N}$. The inverse behaviour, that is a "communist for $y$ spaces," is also well-formed and it is often useful (even though not commendable by everyone). Notice however that it can be harmful too: $!\{*/y\}$. We have not defined the name rebinding capability as a primitive of our calculus since it can be encoded using the new form of spawning as follows, for $a$ fresh.

$$\{\eta/\rho\}_\mathsf{k}.P \triangleq (\boldsymbol{\nu} a)(\mathsf{k}\triangleright_\rho (\blacksquare^k_\eta \mid a^0[\,\overline{\mathbf{open}}\,]) \mid \mathbf{open}\,a.P)$$

Observe that the simpler encoding $\mathsf{k} \rhd_\rho (\blacksquare^{\mathsf{k}}_\eta \mid P)$ is allowed only for processes $P$ of weight 0.

The possibility of encoding the renaming capability justifies also our choice of tags in the (SPAWN) reduction. As a matter of fact it would seem more reasonable to define this reduction as $\mathsf{k} \rhd_\eta P \mid \blacksquare^{\mathsf{k}}_\rho \searrow P$ with the side condition $\eta = \rho$ *or* $\rho = *$. This behaviour, however, can be approximated closely enough by putting a renaming process in parallel with the spawning one, namely $\{^{\eta}/_{*}\} \mid \mathsf{k} \rhd_\eta P$.

It is easy to check that the type system of Section 2 can be used with no substantial modifications also for the calculus with named slots. The obvious changes required are those which guarantee syntax consistency, as e.g. that the $x$ in $\blacksquare_x$ can only be instantiated by a name. For this calculus the same properties proved in Section 2 hold.

**Theorem 3 (Subject Reduction and Under/Over-flow absence).** *For the processes and reduction relation of this section, we have:*

*(i)* $\Gamma \vdash P : Proc\langle \varepsilon, \chi \rangle$ *and* $P \searrow_* Q$ *imply* $\Gamma \vdash Q : Proc\langle \varepsilon', \chi \rangle$ *with* $\varepsilon' \leq \varepsilon$.
*(ii)* *If* $\Gamma, a : Amb\langle [\mathsf{n}, \mathsf{N}], \chi \rangle \vdash P : Proc\langle \varepsilon, \chi \rangle$, $P \searrow_* C[a^{\mathsf{k}}[\, R \,]]$, *and the showed occurrence of* $a$ *is not in the scope of a binder for* $a$, *then* $\mathsf{n} \leq \mathsf{k} \leq \mathsf{N}$.

### 3.2 Behavioural Semantics

The semantic theory of BoCa is based on *barbed congruence* [19], a standard equality relation based on reduction and a notion of observability. As usual in ambient calculi, our observation predicate, $P \downarrow_a$, indicates the possibility for process $P$ to interact with the environment via an ambient named $a$. In Mobile Ambients (MA) this is defined as follows:

$$P \downarrow_a \;\triangleq\; P \equiv (\nu \tilde{m})(a[\, P' \,] \mid Q) \qquad a \notin \tilde{m} \tag{1}$$

Since no authorisation is required to cross a boundary, the presence of an ambient $a$ at top level denotes a potential interaction between the process and the environment via $a$. In the presence of co-capabilities [16], however, the process $(\boldsymbol{\nu} \tilde{m})(a[\, P \,] \mid Q)$ only represents a potential interaction if $P$ can exercise an appropriate co-capability. The same observation applies to BoCa, as many aspects of its dynamics rely on co-capabilities: notably, mobility, opening, and transfer across ambients. Correspondingly, we have several reasonable choices of observation, among which (for $a, \eta \notin \{\tilde{m}\}$):

$$P \downarrow^{opn}_a \;\triangleq\; P \equiv (\boldsymbol{\nu} \tilde{m})(a[\, \overline{\mathbf{open}}.P' \mid Q \,] \mid R) \tag{2}$$

$$P \downarrow^{slt}_a \;\triangleq\; P \equiv (\boldsymbol{\nu} \tilde{m})(a[\, \blacksquare_\eta \mid Q \,] \mid R) \tag{3}$$

$$P \downarrow^{put}_a \;\triangleq\; P \equiv (\boldsymbol{\nu} \tilde{m})(a[\, \mathbf{put}.P' \mid \blacksquare_\eta \mid Q \,] \mid R) \tag{4}$$

As it turns out, definitions (1)–(4) yield the same barbed congruence relation. Indeed, the presence of 0-weighted ambients makes it possible to rely on the same notion of observation as in MA, that is (1), without consequences on barbed congruences. We discuss this in further detail below.

Our notion of barbed congruence is standard in typed calculi, in that we require closure (only) by well-formed contexts. Say that a relation $\mathcal{R}$ is *reduction closed* if $P\mathcal{R}Q$ and $P \searrow P'$ imply the existence of some $Q'$ such that $Q \searrow_* Q'$ and $P'\mathcal{R}Q'$; it is *barb preserving* if $P\mathcal{R}Q$ and $P\downarrow_a$ imply $Q\Downarrow_a$, i.e. $Q \searrow_*\downarrow_a$.

**Definition 2 (Barbed Congruence).** Barbed bisimulation, noted $\simeq$, is the largest symmetric relation on closed processes that is reduction closed and barb preserving. Two processes $P$ and $Q$ are *barbed congruent*, written $P \cong Q$, if for all contexts $\mathcal{C}[\cdot]$, preterm $\mathcal{C}[P]$ is a term iff so is $\mathcal{C}[Q]$, and then $\mathcal{C}[P] \simeq \mathcal{C}[Q]$.

Let then $\cong_i$ be the barbed congruence relation resulting from Definition 2 and from choosing the notion of observation as in ($i$) above (with $i \in [1..4]$).

**Proposition 2 (Independence from barbs).** $\cong_i = \cong_j$ *for all* $i, j \in [1..4]$.

Since the relations differ only on the choice of barb, Proposition 2 is proved by just showing that all barbs imply each other. This can be accomplished, as usual, by exhibiting a corresponding context. For instance, to see that $\cong_3$ implies $\cong_2$ use the context $\mathcal{C}[\cdot] = [\cdot] \mid \mathbf{open}\ a.b^1[\,\blacksquare\,]$, and note that for all $P$ such that $b$ is fresh in $P$ one has $P \Downarrow_a^{opn}$ if and only if $\mathcal{C}[P] \Downarrow_b^{slt}$. Thanks to this Proposition we can denote barbed congruence for BoCa simply by $\cong$.

The import of the processes' weight in the relation of behavioural equivalence is captured directly by the well-formedness requirement in Definition 2. In particular, processes of different weight are distinguished, irrespective of the their "purely" behavioral properties. To see that, note that any two processes $P$ and $Q$ of weight, say, $\mathsf{k}$ and $\mathsf{h}$ with $\mathsf{h} \neq \mathsf{k}$, are immediately distinguished by the context $\mathcal{C}[\cdot] = a^{\mathsf{k}}[\,[\cdot]\,]$, as $\mathcal{C}[P]$ is well-formed while $\mathcal{C}[Q]$ is not. Ultimately, weight is a 'behavioural' property, in that it requires system's space allocation.

### 3.3 Algebraic laws

In Appendix A we give a labelled transition system for the calculus of this section, and sketch a proof of adequacy of the resulting notion of labelled bisimilarity with respect to the relation of barbed congruence given in Definition 2. A number of algebraic laws can be proved based on the resulting co-inductive characterization of barbed congruence. We highlight some of these below.

*Garbage.* There are many different characterization of wasted resources, and all these are congruent, provided they have the same weights.

$$(A_1) \qquad (\mathbf{v}a)a^{\mathsf{k}}[\,\blacksquare^{\mathsf{k}}_{\eta}\,] \;\cong\; (\mathbf{v}a)\mathbf{cap}\ a.\blacksquare^{\mathsf{k}}_{\eta} \qquad (\mathbf{cap} \in \{\mathbf{in}\,, \mathbf{out}\,, \ldots\})$$

Indeed, all these processes are inhert, hence behaviorally equivalent to the null process. Given that they have non-null weight, however, they are not congruent to the $\mathbf{0}$ process, but rather to what may be construed as a new process construct, noted $\mathbf{0}^{\mathsf{k}}$, that provides an explicit representation of a notion of garbage in the calculus.

*Spawning.* The spawning of a process cannot be observed as long as the space required is protected from other, unintended uses. This is true of the form of private spawning based on the primitive naming mechanism for slots, as well as of the encoding given earlier in this section. Specifically, we have:

$$(A_2) \qquad (\boldsymbol{\nu} a)(a^{\mathsf{k}}[\,\_^{\mathsf{k}} \mid \mathsf{k} \triangleright \overline{\mathbf{open}}.\,P\,] \mid \mathbf{open}\; a) \;\cong\; P$$

$$(A_3) \qquad (\boldsymbol{\nu} a)(\__a \mid \mathsf{k} \triangleright_a P) \;\cong\; P$$

*Transfers.* Similar laws relate the exchange of slots between ambients. For example:

$$(A_4) \qquad (\boldsymbol{\nu} a)(a^{\mathsf{k}}[\,\_^{\mathsf{k}} \mid \mathbf{put}^{\mathsf{k}}\,] \mid b^{\mathsf{h}}[\,\mathbf{get}\; a^{\mathsf{k}} \mid P\,]) \;\cong\; b^{\mathsf{k}+\mathsf{h}}[\,\_^{\mathsf{k}} \mid P\,]$$

*Ambient opening and movement.* Given the presence of a co-capability for **open** , ambient opening satisfies the same laws as the calculus of Safe Ambients of [16]. As for movement, corresponding laws hold only for ambients with non-null weight: such ambients may be characterized by means of a typing system in which one requires that all ambient types have a strictly positive lower bound.

For the calculus of §1, the mobility laws are weaker than the corresponding laws in [16], as our slots act uniformly as co-capabilities for **in** and **out** moves. For the calculus of this section, instead, if we assume the typing restriction discussed above, we have:

$$(A_5) \qquad (\boldsymbol{\nu} b)(b^{\mathsf{k}}[\,\mathbf{in}\; a.\,P\,] \mid a^{\mathsf{k}}[\,\__b^{\mathsf{k}}\,]) \;\cong\; (\boldsymbol{\nu} b)(\__b^{\mathsf{k}} \mid a^{\mathsf{k}}[\,b^{\mathsf{k}}[\,P\,]\,])$$

As a further remark, we note that there are congruences, like $(A_4)$, between typeable and untypeable terms for a fixed environment. In fact assuming $w(P) = \mathsf{h}$, $b \notin \Gamma$, and $\Gamma \vdash P : \mathit{Proc}\langle 0_{\mathcal{E}}, \chi\rangle$ the term $b^{\mathsf{k}+\mathsf{h}}[\,\_^{\mathsf{k}} \mid P\,]$ can be typed in the environment $\Gamma, b : \mathit{Amb}\langle[\mathsf{k}+\mathsf{h}, \mathsf{k}+\mathsf{h}], 0_{\mathcal{E}}, \chi\rangle$, while $(\boldsymbol{\nu} a)(a^{\mathsf{k}}[\,\_^{\mathsf{k}} \mid \mathbf{put}^{\mathsf{k}}\,] \mid b^{\mathsf{h}}[\,\mathbf{get}\; a^{\mathsf{k}} \mid P\,])$ cannot.

### 3.4 More Examples

**The cab trip revisited.** Named slots allow us to avoid unwanted behaviours when encoding the cab trip example. The main steps of the protocol may now be described as follows:

- ▷ The *cab* initially contains one slot named *call* to signal that it is vacant.
- ▷ Once the ambient *call* reaches a *cab*, it is opened there to drive *cab* to the client's site. In addition, opening *call* inside *cab* leaves in *cab* a slot with the (private) name *client* of the client. Consequently, only the client whose *call* reached *cab* may eventually enter *cab*.
- ▷ When *client* enters *cab*, it leaves a slot named *client* which is then rebound to an anonymous slot in order for the enclosing site to be able to accept new incoming cabs on that slot.
- ▷ Upon completing the trip to destination, *cab* sets out to complete the protocol: it opens the synchronization ambient *arrived*, and rebinds the slot named *client* to the (again private) name of the acknowledgement ambient *bye*. Opening *arrived* also unleashes the inner occurrence of *done* which in turn enters *client* to signal that it is time for *client* to leave *cab*.

▷ At this stage *client* exits *cab* and the protocol completes with ambient *bye* entering *cab* and being opened there to drive *cab* out of the destination site with a slot named *call*.

Let $W_1 = Amb[1,1]$ $W_0 = Amb[0,0]$. As in Subsection 2.3 the typing environment contains $call : W_1$, $cab : W_1$, $trip : W_0$, $done : W_0$: instead, in the new encoding the ambient *bye* has type $W_1$ and is private to the client. Furthermore, we add the ambient *arrived* with type $W_0$, and we do not need the ambient *load*.

$$
\begin{array}{l}
CALL(from, client, bye) \triangleq \\
\quad call^1[\,\mathbf{out}\ client\,.\,\mathbf{out}\ from\,.\,\mathbf{in}\ cab\,.\,\overline{\mathbf{open}}\,. \\
\qquad\qquad\qquad\qquad \mathbf{in}\ from\,.(\blacksquare_{client} \mid \mathbf{open}\ arrived\,.\{^{bye}/_{client}\}\,.\,\mathbf{open}\ bye)] \\
TRIP(from, to, client) \triangleq \\
\quad trip^0[\,\mathbf{out}\ client\,.\,\overline{\mathbf{open}}\,.\,\mathbf{out}\ from\,.\,\mathbf{in}\ to\,.\,arrived^0[\,\overline{\mathbf{open}}\,.\,done^0[\,\mathbf{in}\ client\,.\,\overline{\mathbf{open}}\,]\,]\,] \\
CLIENT(from, to) \triangleq \\
\quad (\mathbf{v} client : W_1, bye : W_1)\,(client^1[CALL(from, client, bye) \mid \mathbf{in}\ cab\,.\,TRIP(from, to, client) \\
\qquad \mid \mathbf{open}\ done\,.\,\mathbf{out}\ cab\,.\,1 \triangleright_{call} bye^1[\,\mathbf{out}\ client\,.\,\mathbf{in}\ cab\,.\,\overline{\mathbf{open}}\,.\,\mathbf{out}\ to\,.\blacksquare_{call}\,]] \mid \{^*/_{client}\}) \\
CAB \triangleq cab^1[\,\blacksquare_{call} \mid !(\mathbf{open}\ call\,.\,\mathbf{open}\ trip)\,] \mid \{^*/_{cab}\} \\
SITE(i) \triangleq site_i[\,CLIENT(site_i, site_j) \mid CLIENT(site_i, site_l) \mid \cdots \mid \blacksquare \mid \blacksquare \mid \cdots\,] \\
CITY \triangleq city[\,CAB \mid CAB \mid \cdots \mid SITE(1) \mid \cdots \mid SITE(n) \mid \blacksquare \mid \blacksquare \mid \cdots\,]
\end{array}
$$

Using the labelled transition system given in the Appendix we can prove properties of the CAB system, including the expected ones that clients will not be able to board a taxi different from that called and that the ambient *bye* always enter the cab the client just left.

**A Travel Agency.** We conclude the presentation with an example that shows the expressiveness of the naming mechanisms for resources in the refined calculus. We wish to model clients buying tickets from a travel agency, paying them one slot (the $\blacksquare_{fortkt}$ inside the client), and then use them to travel by plane. At most two clients may enter the travel agency, and they are served one by one. The three components of the systems are defined below.

▷ THE AGENCY: $ag^5[\,\blacksquare^2_{cl} \mid \blacksquare_{req} \mid \blacksquare_{tkt}$
$desk^1[\,\blacksquare_{req} \mid !(\mathbf{open}\ req\,.\,1 \triangleright_{fortkt} .\, tkt^1[\,\mathbf{out}\ desk\,.\,\mathbf{in}\ cl\,.\,CONT\,] \mid \{^{req}/_{tkt}\})]\,]$
where $CONT = (\overline{\mathbf{open}}\,.\,\mathbf{out}\ ag\,.\,\mathbf{in}\ plane\,.\,rdy^0[\,\mathbf{out}\ cl\,] \mid \blacksquare_{getoff} \mid \mathbf{open}\ getoff)$

▷ THE CLIENT: $cl^1[\,\mathbf{in}\ ag\,.\,req^1[\,\mathbf{out}\ cl\,.\,\mathbf{in}\ desk\,.\,\overline{\mathbf{open}}\,.\blacksquare_{fortkt}\,] \mid \{^{tkt}/_{req}\} \mid \mathbf{open}\ tkt\,]$

▷ THE AIRCRAFT: $plane^4[\,\blacksquare^2_{cl} \mid \mathbf{open}\ rdy\,.\,\mathbf{open}\ rdy\,.\,TRIP\,.(GETOFF \mid GETOFF)\,]$
where $GETOFF = getoff^1[\,\mathbf{in}\ \mathrm{cl}\,.\,\overline{\mathbf{open}}\,.\,\mathbf{out}\ plane \mid \blacksquare\,]$ and *TRIP* is the unspecified path modelling the route of the aircraft.

We assume that there exists only one sort of ticket, but it is easy to extend the example with as many kinds of ticket as possible plane routes. What makes the example

interesting is the possibility of letting two clients into the agency, but serving them non-deterministically in sequence. Notice that the use of the named slots is essential for a correct implementation of the protocol. When the request goes to the desk, a slot named *tkt* is left in the client. This slot allows the ticket to enter the client. In this way we guarantee that no ticket can enter a client before its request has reached the desk.

We assume the aircraft to leave only when full. This constraint is implemented by means of the *rdy* ambient. The ambient *getoff* enables the passengers to get off once at destination; assigning weight 1 to the *getoff* ambients prevents them to get both into the same client.

## 4 Conclusion and Future Work

We have presented an ambient-like calculus centred around an explicit primitive representing a resource unit: the space "slot" $\blacksquare$. The calculus, dubbed BoCa, features capabilities for resource control, namely pairs **get**/**put** to transfer spaces between sibling ambients and from parent to child, as well as the capabilities **in** *a* and **out** *a* for ambient migration, which represent an abstract mechanism of resource negotiation between travelling agent and its source and destination environments. A fundamental ingredient of the calculus is $\triangleright(\_)$, a primitive which consumes space to activate processes. The combination of such elements makes of BoCa a suitable formalism, if initial, to study the role of resource consumption, and the corresponding safety guarantees, in the dynamics of mobile systems. We have experimented with the all important notion of private resource, which has guided our formulation of a refined version of the calculus featuring named resources.

The presence of the space construct $\blacksquare$ induces a notion of weight on processes, and by exercising their transfer capabilities, processes may exchange resources with their surrounding context, so making it possible to have under- and over-filled ambients. We have introduced a type system which prevents such unwanted effects and guarantees that the contents of each ambient remain within its declared capacity.

As we mentioned in the Introduction, our approach is related to the work on *Controlled Mobile Ambients* (CMA) [27] and on *Finite Control Mobile Ambients* [6]. There are, however, important difference with respect to both approaches.

In CMA the notions of process weight and capacity are entirely characterized at the typing level, and so are the mechanisms for resource control (additional control on ambient behavior is achieved by means of a three-way synchronization for mobility, but that is essentially orthogonal to the mechanisms targeted at resource control). In BoCa, instead, we characterize the notions of space and resources directly in the calculus, by means of an explicit process constructor, and associated capabilities. In particular, the primitives for transferring space, and more generally for the explicit manipulation of space and resources by means of spawning and replication appear to be original to BoCa, and suitable for the development of formal analyses of the fundamental mechanism of the usage and and consumption of resources which do not seem to be possible for CMA.

As to [6], their main goal is to isolate an expressive fragment of Mobile Ambients for which the model checking problem against the ambient logic can be made decidable.

Decidability requires guarantees of finiteness which in turn raise boundedness concerns that are related to those we have investigated here. However, a more thorough comparison between the two approaches deserves to be made and we leave it to our future work.

Plans for future include further work in several directions. A finer typing discipline could be put in place to regulate the behavior of processes in the presence of primitive notions of named slots. Also, the calculus certainly needs behavioral theories and proof techniques adequate for reasoning about resource usage and consumption. Such theories and techniques could be assisted by enhanced typing systems providing static guarantees of a controlled, and bounded, use of resources, along the lines of the work by Hofmann and Jost in [14].

A further direction for future development is to consider a version of weighed ambients whose "external" weight is independent of their "internal" weight, that is the weight of their contents. This approach sees an ambient as a packaging abstraction whose weight may have a different interpretation from that of contents'. For instance, modelling a wallet the weight of its contents could represent the value of the money inside, whereas its external weight could measure the physical space it occupies. A directory's internal weight could be the cumulative size of its files, while the external weight their number.

Last, but not least, we would like to identify logics for BoCa to formulate (quantitative) resource properties and analyses; and to model general resource bounds negotiation and enforcement in the Global Computing scenario.

**Acknowledgements** We gratefully acknowledge the anonymous referees of ASIAN 2003 for careful reading and useful suggestions.

## References

1. F. Barbanera, M. Bugliesi, M. Dezani-Ciancaglini, and V. Sassone. A Calculus of Bounded Capacities. In *ASIAN'03, Eighth Asian Computing Science Conference*, number xxxx in Lecture Notes in Computer Science, pages xx–xx, 2003. To Appear.
2. M. Bugliesi and G. Castagna. Secure safe ambients. In *POPL'01*, pages 222–235, New York, 2001. ACM Press.
3. M. Bugliesi, S. Crafa, M. Merro, and V. Sassone. Communication Interference in Mobile Boxed Ambients. In *FSTTCS'02, Int. Conf. on Foundations of Software Technology and Theoretical Computer Science*, number 2556 in Lecture Notes in Computer Science, pages 71–84. Springer–Verlag, 2002.
4. M. Bugliesi, S. Crafa, M. Merro, and V. Sassone. Communication and Mobility Control in Boxed Ambients. Submitted for publication. Extended and revised version of [3], April 2003.
5. L. Cardelli and A. D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, 2000. Special Issue on Coordination, D. Le Métayer Editor.
6. W. Charatonik, A. D. Gordon, and J.-M. Talbot. Finite-control mobile ambients. In D. Le Métayer, editor, *ESOP'02*, volume 2305 of *LNCS*, pages 295–313, Berlin, 2002. Springer-Verlag.
7. K. Crary and S. Weirich. Resource bound certification. In *POPL'00*, pages 184–198, New York, 2000. ACM Press.

8. J. C. Godskesen, T. Hildebrandt, and V. Sassone. A calculus of mobile resources. In L. Brim, P. Jančar, M. Křetínskỳ, and A. Kučera, editors, *CONCUR'02*, volume 2421 of *LNCS*, pages 272–287, Berlin, 2002. Springer-Verlag.
9. A. D. Gordon and L. Cardelli. Equational properties of mobile ambients. *Mathematical Structures in Computer Science*, 12:1–38, 2002.
10. M. Hennessy, M. Merro, and J. Rathke. Towards a behavioural theory of access and mobility control in distributed system (extended abstract). In A. D. Gordon, editor, *FOSSACS'03*, volume 2620 of *LNCS*, pages 282–299, Berlin, 2003. Springer-Verlag.
11. M. Hennessy and J. Riely. Information flow vs. resource access in the asynchronous pi-calculus. *ACM Transactions on Programming Languages and Systems*, 24(5):566–591, 2002.
12. M. Hennessy and J. Riely. Resource access control in systems of mobile agents. *Information and Computation*, 173:82–120, 2002.
13. M. Hofmann. The strength of non size-increasing computation. In *POPL'02*, pages 260–269, New York, 2002. ACM Press.
14. M. Hofmann and S. Jost. Static prediction of heap space usage for first-order functional programs. In *POPL'03*, pages 185–197, New York, 2003. ACM Press.
15. A. Igarashi and N. Kobayashi. Resource usage analysis. In *POPL'02*, pages 331–342, New York, 2002. ACM Press.
16. F. Levi and D. Sangiorgi. Controlling interference in Ambients. In *POPL'00*, pages 352–364. ACM Press, New York, 2000.
17. F. Levi and D. Sangiorgi. Controlling interference in ambients. In *POPL'00*, pages 352–364, New York, 2000. ACM Press.
18. M. Merro and M. Hennessy. Bisimulation Congruences in Safe Ambients. In *POPL'02*, pages 71–80, New York, 2002. ACM Press.
19. R. Milner and D. Sangiorgi. Barbed bisimulation. In W.Kuich, editor, *ICALP'92*, volume 623 of *LNCS*, pages 685–695, Berlin, 1992. Springer-Verlag.
20. G. Nemhauser and L. Wolsey. *Integer and combinatorial optimization*. Wiley, 1988.
21. R. D. Nicola, G. Ferrari, R. Pugliese, and B. Venneri. Types for access control. *Theoretical Computer Science*, 240(1):215–254, 2000.
22. B. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–454, 1996.
23. D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis CST–99–93, Department of Computer Science, University of Edinburgh, 1992.
24. D. Sangiorgi. Bisimulation for Higher-Order Process Calculi. *Information and Computation*, 131(2):141–178, 1996.
25. D. Sangiorgi. Extensionality and intensionality of the ambient logic. In *POPL'01*, pages 4–13, New York, 2001. ACM Press.
26. D. Sangiorgi and R. Milner. The problem of "Weak Bisimulation up to". In *Proc. of CONCUR'92*, volume 630 of *Lecture Notes in Computer Science*, pages 32–46. Springer-Verlag, 1992.
27. D. Teller, P. Zimmer, and D. Hirschkoff. Using ambients to control resources. In L. Brim, P. Jančar, M. Křetínskỳ, and A. Kučera, editors, *CONCUR'02*, volume 2421 of *LNCS*, pages 288–303, Berlin, 2002. Springer-Verlag.
28. J. Wells. The essence of principal typings. In P. Widmayer, F. Triguero, R. Morales, M. Hennessy, S. Eidenbez, and R. Conejo, editors, *ICALP'02*, volume 2380 of *LNCS*, pages 913–925, Berlin, 2002. Springer-Verlag.
29. N. Yoshida and M. Hennessy. Subtyping and locality in distributed higher order mobile processes (extended abstract). In J. C. M. Baeten and S. Mauw, editors, *CONCUR'99*, volume 1664 of *LNCS*, pages 557–573, Berlin, 1999. Springer-Verlag.

# A A Labelled Transition system for BoCa

**Table 1** Labels, concretions and outcomes

| | |
|---|---|
| *Prefixes* | $\gamma ::= \mathbf{in}\ a \mid \mathbf{out}\ a \mid \mathbf{open}\ a \mid \overline{\mathbf{open}} \mid \mathbf{get}\, a_\eta \mid \mathbf{get}^{\uparrow}_\eta \mid \mathbf{put} \mid \mathbf{put}^{\downarrow} \mid \mathsf{k} \triangleright \eta$ |
| *Labels* | $\alpha ::= \gamma \mid \mathsf{k} \mid \blacksquare^{\mathsf{k}}_\eta \mid \mathbf{put}\,\eta \mid \mathbf{put}^{\downarrow}\,\eta \mid \langle M \rangle \mid \langle - \rangle$ $\mid\ *[\,\mathbf{get}\, a_\eta\,] \mid *[\,\mathbf{get}^{\uparrow}_\eta\,] \mid \eta^{\mathsf{k}}[\,\mathbf{out}\ a\,] \mid \eta^{\mathsf{k}}[\,\mathbf{in}\ a\,] \mid \eta^{\mathsf{k}}[\,\mathbf{exit}\ a\,]$ $\mid\ a[\,\overline{\mathbf{open}}\,] \mid a[\,\mathbf{put}\,\eta\,] \mid a[\,\blacksquare^{\mathsf{k}}_\eta\,]$ |
| *Concretions* | $K ::= (\boldsymbol{\nu}\tilde{m})\langle P \rangle Q \mid (\boldsymbol{\nu}\tilde{m})\langle M \rangle P$ |
| *Outcomes* | $O ::= P \mid K$ |

We give a labelled transition system for the calculus of §3. A corresponding LTS can readily be obtained for the calculus of §1 by simply erasing all the occurrences of $\eta$ and $\rho$ from the labels and the corresponding transitions. Based on the labelled transitions, we then introduce a labelled bisimilarity which, because of its co-inductive nature, will provide powerful proof techniques for establishing equivalences [23, 26, 24]. As usual for ambient calculi [9, 17, 18, 4], the labelled transitions have the form $P \xrightarrow{\alpha} O$, where $P$ is a well-formed term , and

- ▷ the *label* $\alpha$ encodes the minimal contribution by the environment needed for the process to complete the transition;
- ▷ the *outcome* $O$ can be either a *concretion*, i.e. a partial derivative which needs a contribution from the environment to be completed, or a process.

Table 1 defines labels and concretions. In $(\boldsymbol{\nu}\tilde{p})\langle P \rangle Q$ the process $P$ represents the moving ambient and the process $Q$ represents the remaining system not affected by the movement. In $(\boldsymbol{\nu}\tilde{p})\langle M \rangle P$ the message $M$ represents the information transmitted and the process $P$ represents the remaining system not affected by the output. In both cases $\tilde{p}$ is the set of shared private names.

Tables 2 and 3 give the labelled transition system. In writing the rules we will use the following standard conventions:

- ▷ if $O$ is the concretion $(\boldsymbol{\nu}\tilde{p})\langle P \rangle Q$, then:
  - $(\boldsymbol{\nu} r)O = (\boldsymbol{\nu}\tilde{p})\langle P \rangle (\boldsymbol{\nu} r)Q$, if $r \notin \mathrm{fn}(P)$, and $(\boldsymbol{\nu} r)O = (\boldsymbol{\nu} r, \tilde{p})\langle P \rangle Q$ otherwise.
  - $O \mid R = (\boldsymbol{\nu}\tilde{p})\langle P \rangle (Q \mid R)$, where $\tilde{p}$ are chosen so that $\mathrm{fn}(R) \cap \{\tilde{p}\} = \emptyset$.
- ▷ if $O$ is the concretion $(\boldsymbol{\nu}\tilde{p})\langle M \rangle P$, then:
  - $(\boldsymbol{\nu} r)O$ is $(\boldsymbol{\nu}\tilde{p})\langle M \rangle ((\boldsymbol{\nu} r)P)$, if $r \notin \mathrm{fn}(M)$, and $(\boldsymbol{\nu} r, \tilde{p})\langle M \rangle Q$ otherwise.
  - $O \mid R = (\boldsymbol{\nu}\tilde{p})\langle M \rangle (P \mid R)$, where $\tilde{p}$ are chosen so that $\mathrm{fn}(R) \cap \{\tilde{p}\} = \emptyset$.

**Table 2** Commitments: Visible transitions

(CAP)

$$\frac{M \in \{\mathbf{in}\ a, \mathbf{out}\ a, \mathbf{open}\ a, \overline{\mathbf{open}}, \mathbf{put}, \mathbf{put}^{\downarrow}, \mathsf{k} \triangleright \eta\}}{M.P \xrightarrow{M} P}$$

(PATH)

$$\frac{M_1.(M_2.P) \xrightarrow{\alpha} P'}{(M_1.M_2).P \xrightarrow{\alpha} P'}$$

(WEIGHT)

$$\frac{w(P) = \mathsf{k}}{P \xrightarrow{\mathsf{k}} P}$$

(SLOT-0)

$$\frac{}{P \xrightarrow{\blacksquare_*^0} P}$$

(SLOT-1)

$$\frac{}{\blacksquare \xrightarrow{\blacksquare_\eta^1} \mathbf{0}}$$

(SLOT-PAR)

$$\frac{P \xrightarrow{\blacksquare_\eta^{\mathsf{k}}} P_1 \quad Q \xrightarrow{\blacksquare_\eta^1} Q_1 \quad (\mathsf{k} \geq 1)}{P \mid Q \xrightarrow{\blacksquare_\eta^{\mathsf{k}+1}} P_1 \mid Q_1}$$

(GET)

$$\frac{M \in \{\mathbf{get}\, a_\eta, \mathbf{get}^{\uparrow}{}_\eta\}}{M.P \xrightarrow{M} \blacksquare_\eta \mid P}$$

(PUT)

$$\frac{M \in \{\mathbf{put}, \mathbf{put}^{\downarrow}\}\ P \xrightarrow{M} P'\ Q \xrightarrow{\blacksquare_\eta^1} Q'}{P \mid Q \xrightarrow{M\,\eta} P' \mid Q'}$$

(SLOT-INC)

$$\frac{M \in \{\mathbf{get}\, a_\eta, \mathbf{get}^{\uparrow}{}_\eta\} \quad P \xrightarrow{M} P'}{a^{\mathsf{k}}[\,P\,] \xrightarrow{*[M]} a^{\mathsf{k}+1}[\,P'\,]}$$

(SLOT-DEC)

$$\frac{P \xrightarrow{\mathbf{put}\,\eta} P'}{a^{\mathsf{k}+1}[\,P\,] \xrightarrow{a[\,\mathbf{put}\,\eta\,]} a^{\mathsf{k}}[\,P'\,]}$$

(INPUT)

$$\frac{}{(x:\chi).P \xrightarrow{\overline{\langle M \rangle}} P\{x := M\}}$$

(OUTPUT)

$$\frac{}{\langle M \rangle.P \xrightarrow{\langle - \rangle} (\boldsymbol{\nu})\langle M \rangle P}$$

(IN-OUT)

$$\frac{P \xrightarrow{M} P' \quad M \in \{\mathbf{in}\ a, \mathbf{out}\ a\},\ \eta \in \{b, *\}}{b^{\mathsf{k}}[\,P\,]_\rho \xrightarrow{\eta^{\mathsf{k}}[M]} (\boldsymbol{\nu})\langle b^{\mathsf{k}}[\,P'\,]_\eta \rangle \blacksquare_\rho^{\mathsf{k}}}$$

(CO-IN)

$$\frac{P \xrightarrow{\blacksquare_\eta^{\mathsf{k}}} P'}{a^{\mathsf{h}}[\,P\,] \xrightarrow{a[\,\blacksquare_\eta^{\mathsf{k}}\,]} (\boldsymbol{\nu})\langle P' \rangle \mathbf{0}}$$

(EXIT)

$$\frac{P \xrightarrow{\eta^{\mathsf{k}}[\,\mathbf{out}\ a\,]} (\boldsymbol{\nu}\tilde{p})\langle P_1 \rangle P_2}{a^{\mathsf{h}}[P] \xrightarrow{\eta^{\mathsf{k}}[\,\mathbf{exit}\ a\,]} (\boldsymbol{\nu}\tilde{p})(P_1 \mid a^{\mathsf{h}}[P_2])}$$

(CO-OPEN)

$$\frac{P \xrightarrow{\overline{\mathbf{open}}} P'}{a[\,P\,] \xrightarrow{a[\,\overline{\mathbf{open}}\,]} P'}$$

**Table 3** Commitments: τ transitions and structural transitions

(τ-ENTER)

$$\frac{P \xrightarrow{\eta^k[\,\mathbf{in}\ a\,]} (\nu\tilde{p})\langle P_1\rangle P_2 \quad Q \xrightarrow{a[\,\blacksquare^k_\eta\,]} (\nu\tilde{q})\langle Q_1\rangle Q_2 \quad \begin{array}{r}\mathsf{h} = w(P_1 \mid Q_1)\\ (\mathrm{fn}(P_1)\cup\mathrm{fn}(P_2))\cap\{\tilde{q}\} = \emptyset\\ (\mathrm{fn}(Q_1)\cup\mathrm{fn}(Q_2))\cap\{\tilde{p}\} = \emptyset\end{array}}{P \mid Q \xrightarrow{\tau} (\nu\tilde{p},\tilde{q})(a^{\mathsf{h}}[Q_1 \mid P_1] \mid P_2 \mid Q_2)}$$

(τ-EXIT)

$$\frac{P \xrightarrow{\eta^k[\,\mathbf{exit}\ a\,]} P_1 \quad Q \xrightarrow{\blacksquare^k_\eta} Q_1}{P \mid Q \xrightarrow{\tau} P_1 \mid Q_1}$$

(τ-OPEN)

$$\frac{P \xrightarrow{\mathbf{open}\ a} P_1 \quad Q \xrightarrow{a[\,\overline{\mathbf{open}}\,]} Q_1}{P \mid Q \xrightarrow{\tau} P_1 \mid Q_1}$$

(τ-TRANS)

$$\frac{P \xrightarrow{*[\,\mathbf{get}\,a_\eta\,]} P_1 \quad Q \xrightarrow{a[\mathbf{put}\,\eta]} Q_1}{P \mid Q \xrightarrow{\tau} P_1 \mid Q_1}$$

(τ-TRAND)

$$\frac{P \xrightarrow{*[\,\mathbf{get}^{\uparrow}_\eta\,]} P_1 \quad Q \xrightarrow{\mathbf{put}^{\downarrow}\,\eta} Q_1}{P \mid Q \xrightarrow{\tau} P_1 \mid Q_1}$$

(τ-EXCHANGE)

$$\frac{P \xrightarrow{(M)} P_1 \quad Q \xrightarrow{\langle - \rangle} (\nu\tilde{q})\langle M\rangle Q_1 \quad \mathrm{fn}(P)\cap\tilde{q} = \emptyset}{P \mid Q \xrightarrow{\tau} (\nu\tilde{q})(P_1 \mid Q_1)}$$

(τ-SPAWN)

$$\frac{P \xrightarrow{\mathsf{k}\triangleright\eta} P_1 \quad Q \xrightarrow{\blacksquare^k_\eta} Q_1}{P \mid Q \xrightarrow{\tau} P_1 \mid Q_1}$$

(PAR)

$$\frac{P \xrightarrow{\alpha} O}{P \mid Q \xrightarrow{\alpha} O \mid Q}$$

(RES)

$$\frac{P \xrightarrow{\alpha} O \quad n \notin \mathrm{fn}(\alpha)}{(\nu n)P \xrightarrow{\alpha} (\nu n)O}$$

(τ-AMB)

$$\frac{P \xrightarrow{\tau} P'}{n[P] \xrightarrow{\tau} n[P']}$$

(REPL)

$$\frac{\pi.P \xrightarrow{\alpha} O}{!\pi.P \xrightarrow{\alpha} !\pi.P \mid O}$$

The transitions are similar to the transitions defined for [17, 4] when we interpret an occurrence of a slot as a co-capability for movement and spawning. The newest rule is (WEIGHT) which allows to distinguish processes only on the basis of their weights. Peculiar to our calculus are the rules dealing with slots: in particular rule (SLOT-0) says that each process becomes itself using no slot, instead rule (SLOT-1) says that one slot can be consumed becoming the null process. Rule (SLOT-0) is useful for allowing the movement and the spawning of processes with weight 0. Peculiar are also rules (SLOT-INC) and (SLOT-DEC) in which the weights of the ambient change.

The labelled transition semantics agrees with the reduction semantics: this can be easily checked from the definitions.

**Theorem 4.** *If $P \xrightarrow{\tau} Q$ then $P \searrow Q$. Conversely, if $P \searrow Q$ then $P \xrightarrow{\tau} Q'$ for some $Q' \equiv Q$.*

**Table 4** Commitments: Higher-Order transitions

(HO OUTPUT)

$$\frac{P \xrightarrow{\langle - \rangle} (\boldsymbol{\nu}\tilde{p})\langle M \rangle P' \quad \mathrm{fv}(Q) \subseteq \{x\},\ \tilde{p} \cap \mathrm{fn}(Q) = \emptyset}{P \xrightarrow{\langle - \rangle Q} (\boldsymbol{\nu}\tilde{p})(P' \mid Q\{x := M\})}$$

(HO IN/CO-IN)

$$\frac{P \xrightarrow{M} (\boldsymbol{\nu}\tilde{p})\langle P_1 \rangle P_2 \quad M \in \{\eta^k[\,\mathbf{in}\ a\,], a[\,\blacksquare^k_\eta\,]\} \quad \tilde{p} \cap \mathrm{fn}(Q) = \emptyset, \quad \mathsf{h} = w(P_1 \mid Q)}{P \xrightarrow{MQ} (\boldsymbol{\nu}\tilde{p})(a^{\mathsf{h}}[\,P_1 \mid Q\,] \mid P_2)}$$

(HO OUT)

$$\frac{P \xrightarrow{\eta^k[\,\mathbf{out}\ a\,]} (\boldsymbol{\nu}\tilde{p})\langle P_1 \rangle P_2 \quad \tilde{p} \cap \mathrm{fn}(Q) = \emptyset, \quad \mathsf{h} = w(P_2 \mid Q)}{P \xrightarrow{\eta^k[\,\mathbf{out}\ a\,]Q} (\boldsymbol{\nu}\tilde{p})(P_1 \mid a^{\mathsf{h}}[\,P_2 \mid Q\,])}$$

We can also shows that our definition of barb coincides with one particular action of the labelled transition system: the action $a[\,\blacksquare^0_*\,]$. This follows from the fact that for all $Q$ we get $a[\,Q\,] \xrightarrow{a[\,\blacksquare^0_*\,]} (\boldsymbol{\nu})\langle Q \rangle \mathbf{0}$. Below, we write $\overline{\mathbf{in}}$ to denote $\blacksquare^0_*$.

**Proposition 3.** *$P \downarrow_a$ if and only if $P \xrightarrow{a[\,\overline{\mathbf{in}}\,]} (\boldsymbol{\nu})\langle Q \rangle R$ for some $Q, R$.*

Following [18, 4], in order to provide a characterization of barbed congruence in terms of (weak) labelled bisimilarity, we introduce a new, higher-order transition for each of the first-order transitions whose outcome is a concretion, rather than a process.

The new transitions are collected in Table 4. The higher-order labels occurring in these transitions encode the minimal contribution by the environment needed for the process to complete a transition. Thus, in (HO OUTPUT) the process $Q$ represents the context receiving the value $M$ output by $P$, and the variable $x$ is a placeholder for that value. In rule (HO IN) the environment provides an ambient $a[Q]$ in which $P_1$ moves, while in the rule (HO CO-IN) the environment provides an ambient $Q$ moving into $a$. Finally in rule (HO OUT) we can imagine the environment wrapping the process $P$ with an ambient $a[Q]$.

We are now ready to give the relation of labelled bisimilarity. Let $\Lambda$ be the set of all labels including the first-order labels of Table 1 as well as the higher-order labels determined by the transitions in Table 4. We denote with $\lambda$ any label in the set $\Lambda$. As usual, we focus on weak bisimilarities based on weak transitions, and use the following notation:

i) $\stackrel{\lambda}{\Longrightarrow}$ denotes $\xrightarrow{\tau}{}^* \xrightarrow{\lambda} \xrightarrow{\tau}{}^*$;
ii) $\stackrel{\hat{\lambda}}{\Longrightarrow}$ denotes $\xrightarrow{\tau}{}^*$ (also noted $\Longrightarrow$) if $\lambda = \tau$ and $\stackrel{\lambda}{\Longrightarrow}$ otherwise.

**Definition 3 (Bisimilarity).** *A symmetric relation $\mathcal{R}$ over closed processes is a* bisimulation *if $P\mathcal{R}Q$ and $P \xrightarrow{\lambda} P'$ imply that there exists $Q'$ such that $Q \stackrel{\hat{\lambda}}{\Longrightarrow} Q'$ and $P'\mathcal{R}Q'$. Two processes $P$ and $Q$ are bisimilar, written $P \approx Q$, if $P\mathcal{R}Q$ for some bisimulation $\mathcal{R}$.*

This definition of bisimilarity is only given for closed processes. We generalize it to arbitrary processes as follows:

**Definition 4 (Full bisimilarity).** *Two processes $P$ and $Q$ are full bisimilar, $P \approx_c Q$, if $P\sigma \approx Q\sigma$ for every closing substitution $\sigma$.*

Note that the definition of bisimilarity only tests transitions from processes to processes. As expected the full bisimilarity is a congruence: this can be proved using the technique of [18, 4]. Moreover the full bisimilarity is sound but not complete w.r.t. the reduction barbed congruence.

**Theorem 5 (Soundness of full bisimilarity).** *If $P \approx_c Q$ then $P \cong Q$.*

*Proof.* Notice that rule (WEIGHT) distinguishes processes of different weights. Then it is enough to show that $\approx_c$ is reduction closed and barb preserving, up to $\equiv$. Assume that $P \searrow P'$. By Theorem 4 $P \xrightarrow{\tau} \equiv P'$. Since $P \approx_c Q$, there exits $Q'$ such that $Q \searrow_* Q'$ and $P' \equiv \approx_c \equiv Q'$. Now assume $P \approx_c Q$. If $P\downarrow_a$ then, by Proposition 3, and rule (HO CO-IN), $P \xrightarrow{a[\,\overline{\mathbf{in}}\,]R} S$ for some $R, S$. Since $P \approx_c Q$ we know that $Q \xRightarrow{a[\,\overline{\mathbf{in}}\,]R} S'$ for some $S' \approx_c S$, from which $Q\Downarrow_a$, as desired.

The failure of completeness is due to the fact that contexts are insensible to repeated entering and exiting. This phenomena is called *stuttering* in [25] and it is typical of movements which do not consume co-capabilities, as happen in our calculus. Let us recall the example of [25]. If $P+Q$ denotes the sum operator à la CCS (which can be encoded for example as $(\mathbf{\nu} a)(a[\,P \mid \overline{\mathbf{open}}\,] \mid a[\,Q \mid \overline{\mathbf{open}}\,] \mid \mathbf{open}\ a)$), then no context can distinguish between the processes:

$$\mathbf{in}\ a\,.\,\mathbf{out}\ a\,.\,\mathbf{in}\ a\,.\,\mathbf{0}$$
$$\mathbf{in}\ a\,.\,\mathbf{out}\ a\,.\,\mathbf{in}\ a\,.\,\mathbf{0} + \mathbf{in}\ a\,.\,\mathbf{0}$$

Notice that it is crucial for BoCa that the process after the movement has weight 0: in fact if we consider

$$P \equiv \mathbf{in}\ a\,.\,\mathbf{out}\ a\,.\,\mathbf{in}\ a\,.\,P'$$
$$Q \equiv \mathbf{in}\ a\,.\,\mathbf{out}\ a\,.\,\mathbf{in}\ a\,.\,P' + \mathbf{in}\ a\,.\,P'$$

where the weight of $P'$ is $k+1$ (so $P$ weights $k+1$ and $Q$ weights $2k+2$), then a context which discriminates these two processes is $a^{k+1}[\,[\,]\,]$, since $a^{k+1}[\,P\,]$ is well-formed while $a^{k+1}[\,Q\,]$ is not.