Space-Aware Ambients and Processes[⋆]

Franco Barbanera

Università di Catania, Italy

Michele Bugliesi

Università "Ca' Foscari", Italy

Mariangiola Dezani-Ciancaglini

Università di Torino, Italy

Vladimiro Sassone

University of Sussex

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 calculus with bounded capacities and weighed processes, where migration and activation require space. A type system controls the dynamics of the calculus by providing static guarantees that the intended capacity bounds are preserved throughout the computation. We investigate various term-level mechanisms to complement the typed control on the dynamics of space allocation and acquisition, and study their consequences on the semantic theory of the calculus.

[⋆] F. Barbanera acknowledges supported from MIUR project EOS, M. Bugliesi from EU-FET project 'MyThS' IST-2001-32617 and from MIUR Project n. 2005015785. M. Dezani-Ciancaglini from EU-FET project DART IST-2001-33477, and from MIUR Project McTafi, V. Sassone from EU-FET project 'MIKADO' IST-2001-32222.

Introduction

Resource control, in diverse incarnations, has recently been the focus of extensive foundational research in concurrent and distributed systems. Starting with the seminal work in [24], the topics considered include a wide range of themes, from the control of the location of channel names [33], to the guarantee that distributed agents will access resources only when allowed to do so [13,9,4,12].

Emerging computing paradigms, such as Global Computing, have created new challenges for this research. They envision scenarios in which mobile devices (e.g. smart cards, PDAs, embedded devices) roam across domains and networks in search of computational resources, thus creating the need, for the hosting networks and environments, to build reliable guarantees of a disciplined, and bounded, usage of such resources.

Our interest here is on the analysis of resources in computations of mobile agents, with specific focus on properties related to *space consumption* and *capacity bounds*. We formulate our analysis in one of the most prominent foundational models of mobility, the Ambient Calculus [6], which we extend by introducing a physical, yet abstract, notion of space. We inject this notion directly into the calculus by means of a new process constructor, the *slot*, which evolves out of the homonym notion of [10]: a slot is to be interpreted as a unit of computation space to be allocated to migrating ambients and, more generally, to running processes.

As in previous, related work on resource analysis (cf. Section 6), we employ a type system to control the usage and allocation of space. In our calculus, this is accomplished by a system of capacity types in which the types of ambients establish resource policies in terms of capacity bounds that control the movement and the spawning of processes inside the distributed computational ambients. Then, the typing rules enable us to certify statically that the capacity bounds imposed by such ambients are preserved by the execution of local processes and by the migration of mobile agents.

Distinctive of our approach is the choice of injecting the notion of space directly into the syntax of terms, rather than letting it surface only in the formalization of the dynamics of their (the terms') execution. Our choice is deliberate, and typical of the practice 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 open and inherently dynamic nature of the network deny us any substantial form of global knowledge. Therefore, to compensate for that, we introduce syntactic constructs to support the static analysis. In our calculus the possibility of dynamically checking particular space constraints is a consequence of the explicit presence of the slot primitive. Additional, and finer, control on the dynamics of space allocation/deallocation is provided by means of a naming mechanism for slots that we introduce with a refined version of the calculus. 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, whose associated bisimulation congruence is adequate with respect to (reduction) 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.

Structure of the paper. We begin Section 1 by giving motivations for the design of the calculus. In Section 2 we present the formal description of the new calculus, that we name BoCa (for Bounded Capacities) and illustrate it with a few examples. In Section 3 we introduce the system of capacity types, and prove it sound. In Section 4 we introduce the refined calculus, and in Section 5 we develop its operational and behavioural semantics, based on reduction barbed congruence. We also introduce a labelled transition system for this calculus and prove the resulting notion of labelled bisimilarity sound with respect to reduction barbed congruence. In Section 6 we discuss related work and we conclude with final remarks in Section 7.

A preliminary version of this paper appeared in [2].

1 Ambients, Processes and Space

The calculus of Mobile Ambients [6] (MA) introduced the notion of ambients acting at the same time as administrative domains and as computational environments. Processes live inside ambients, and inside ambients compute and interact. Ambients relocate themselves, carrying along all their contents: their migration, triggered by the processes they enclose, models mobility of entire domains and active computational loci. Two capabilities control ambient movements: **in** and **out**. These are performed by processes wishing their enclosing ambient to move to a sibling and, respectively, out of its parent. A third capability, **open**, can be used to dissolve ambient boundaries. To illustrate, in the configuration $a[\mathbf{open}\ b.\mathbf{in}\ c] \mid b[\mathbf{in}\ a.\mathbf{out}\ c] \mid c[\,]$ the ambient b may enter a, by exercising the *capability* **in** a, and reduce to the new configuration $a[\mathbf{open}\ b.\mathbf{in}\ c \mid b[\mathbf{out}\ c]] \mid c[\,]$. Then a may dissolve the boundary provided by b by exercising **open** b, reducing to $a[\textbf{in}\ c \mid \textbf{out}\ c] \mid c[\,]$. Two further reduction steps may then bring a into c and then back out reaching the final configuration $a[\,] \mid c[\,]$. In addition, ambients and processes may communicate. Communication is asynchronous and anonymous, and happens inside ambients. The configuration $(x)P \mid \langle M \rangle$ represents the parallel composition of two processes, the output process $\langle M \rangle$ "dropping" the message M, and the input process $(x)P$ reading the message M and continuing as $P\{x := M\}$.

BoCa extends the set of basic constructs from MA with an explicit representation of resource/space, formalized by the slot constructor, noted $\blacksquare$. To illustrate, the configuration

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

represents a system which is running process P and which has k resource/space units available to migrating agents willing to enter, and for P to spawn new local subprocesses. 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 k units may be activated on the system.

The allocation of resources in BoCa is the result of explicit requests by client processes. Moving ambients must declare their resource requirements by means of tags, as in $a^{\mathsf{k}}[\ P\ ]$; similarly, processes willing to spawn new subprocesses must explicitly signal the cost of the spawning, as in $\mathsf{k} \triangleright P$: in both cases, the tag k represents the activation cost of the associated processes at the target site. For mobile agents, migration implies a release of the space required for their computation at the source site and the acquisition of corresponding space at the target context, as formalized by the following reductions ($\blacksquare^{\mathsf{k}}$ is short for $\blacksquare \mid \ldots \mid \blacksquare$, k times):

$$a^{\mathsf{k}}[\ \textbf{in}\ b\,.\,P \mid Q\ ] \mid b[\ \blacksquare^{\mathsf{k}} \mid R\ ] \searrow \blacksquare^{\mathsf{k}} \mid b[\ a^{\mathsf{k}}[\ P \mid Q\ ] \mid R\ ]$$

$$\blacksquare^{\mathsf{k}} \mid b[\ P \mid a^{\mathsf{k}}[\ \textbf{out}\ b\,.\,Q \mid R\ ]\ ] \searrow a^{\mathsf{k}}[\ Q \mid R\ ] \mid b[\ P \mid \blacksquare^{\mathsf{k}}\ ]$$

The action of spawning a new local process is also made explicit in the calculus by the new process form $\mathsf{k} \triangleright P$, whose dynamics is defined as follows:

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

Here $\mathsf{k}\ \triangleright P$ is a "spawner" which launches P provided that the local context is ready to allocate enough (i.e., k) fresh resources for the activation. The tag k represents the "activation cost" for process P, while $\mathsf{k} \triangleright P$ is the "frozen code" of P which has no associated cost.

Clearly, to make sense of the above reductions, the tag k must provide a sound estimate of the resources needed by the associated processes, a property that we formalize in BoCa by a well-formedness condition which we impose, and assume as a pre-requisite, on all processes. The formalization of well-formedness requires, in turn, a precise definition of what is meant by "computational requirements" and "activation costs," in short, by *weight* or *size*, of processes and agents.

Various choices are at disposal. One may, for instance, measure the size of a process in terms of the process' count of ambient occurrences and/or output, as proposed in [7]. Alternatively, the weight of a process may be directly related to the number of its composing threads. Both choices appear controversial, however, as there is no clear-cut argument in favor of either one, and none appears to capture the import of mobility in the computational costs associated with execution.

Rather than committing to any of these solutions we chose to adopt a different, more foundational approach: we measure the weight/size of a process in terms of the amount of space (i.e. the number of slots) it may *potentially* unleash/release to its hosting environment in the course of the computation. The well-formedness condition can then be defined by requiring that the tags attached to processes be consistent with the weight of their associated processes. Thus, for instance, $a^{\mathsf{k}}[\,\blacksquare\,]$ and $\mathsf{k} \triangleright \mathbf{in}\ a\,.\,\blacksquare$ will turn out to be well-formed for $\mathsf{k} = 1$, as $a[\,\blacksquare\,]$ and $\mathbf{in}\ a\,.\,\blacksquare$ both weigh 1, and ill-formed otherwise.

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.

This informal description of the calculus is completed by presenting the 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.

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

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

Fig. 1. Weight function $w : \mathit{Processes} \rightharpoonup \omega$

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

2 The Calculus

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, h, 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 Section 3. As in other variants of Mobile Ambients, communication in BoCa is synchronous.

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

$$
\begin{aligned}
&\mathit{Processes}\ P &&::= \blacksquare \mid \mathbf{0} \mid P \mid P \mid (\boldsymbol{\nu} a : W)P \mid \pi\,.\,P \mid !\pi\,.\,P \mid M^{\mathsf{k}}[\, P\,]\\
&\mathit{Messages}\ M &&::= a \mid x \mid \mathbf{in}\ M \mid \mathbf{out}\ M \mid \mathbf{open}\ M \mid M\,.\,M\\
& &&\quad\mid \overline{\mathbf{open}} \mid \mathbf{get}\ M \mid \mathbf{get}^{\uparrow} \mid \mathbf{put} \mid \mathbf{put}^{\downarrow}\\
&\mathit{Guards}\quad G &&::= M \mid (x : W) \mid \langle M \rangle\\
&\mathit{Prefixes}\quad \pi &&::= G \mid \mathsf{k} \triangleright
\end{aligned}
$$

A (well-formed) term *P is a preterm such that $w(P) \neq \bot$, where w is the* partial weight *function defined in Figure 1.*

We rely on 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 k does not matter or can be inferred by the context. Also note that in $a^{\mathsf{k}}[\; P \;]$ the weight tag k refers to the ambient process $a[\; P \;]$ and *not* to the name a. We use $\_^{\mathsf{k}}$ as a shorthand for $\_ \mid \ldots \mid \_$ (k times) (then $\_^{0}$ is $\mathbf{0}$) and G^{k} as a shorthand for $G\,.\,\ldots\,.\,G$ (again 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. This restriction is costless: since $0 \triangleright P$ reduces to P for all P of weight 0 (rule (SPAWN) in Figure 2.1), we can always put a zero-weighted process in prefixed form.

The function $w : \mathit{Processes} \rightharpoonup \omega$ measures the weight of a process in terms of the the number of slots a process may *potentially* release to its hosting environment in the course of the computation. More precisely, the weight of process P is the count of the slots occurring in the *spawned* subprocesses of P, where a process is *spawned* if it does not occur under a spawner $\mathsf{k} \triangleright$. The well-formedness condition, in turn, formalizes the requirement that the tags attached to processes be consistent with the weight of their associated processes. We illustrate these ideas with few sample terms below:

- $a^{2}[\; \mathbf{in}\; c\,.\,\_ \mid b^{1}[\; \_ \;] \;]$ and $a^{2}[\; \_ \mid b^{1}[\; \_ \;] \;]$ are both well-formed and of weight 2 (by composing weights at different nesting levels);
- $1 \triangleright \mathbf{in}\; a\,.\,\_$ and $2 \triangleright a^{2}[\; \_ \mid b^{1}[\; \_ \;] \;]$ are both well-formed and of weight 0, as unspawned processes do not contribute any weight;
- $a^{\mathsf{k}}[\; \_ \;]$ and $\mathsf{k} \triangleright \mathbf{in}\; a\,.\,\_$ are well-formed only for $\mathsf{k} = 1$: their weight for $\mathsf{k} \neq 1$ is $\perp$, since the tags used are not consistent with the weight of the associated processes.

While formally convenient, and conceptually intuitive, our definition of process weight conveys the three different concepts associated with the standard allocate/use/free model of memory usage in programming languages. Specifically, an unguarded top-level slot is naturally interpreted as a piece of unallocated memory; spawning a process, in turn, corresponds to allocating new memory to that process, while a slot enclosed in an ambient or protected by a guard represents the usage of memory by that process. In principle, each piece of memory allocated to a process will be disposed by releasing the slots that determined that process' weight: this is a direct consequence of our notion of well-formedness. On the other hand, the execution of a well-formed process may also end-up creating garbage, i.e. allocating memory that is never released. For instance, the process

$$2 \triangleright \langle M \rangle (\_ \mid (\boldsymbol{\nu} a) a^{1}[\; \_ \;])$$

is well-formed, and requires two slots to be spawned. Once spawned, however, it will never release the slot allocated to the subprocess $(\boldsymbol{\nu} a)a^1[\,\rule{1ex}{0.4ex}\,]$.

2.1 Dynamics of terms

The dynamics of the calculus is given as usual in terms of structural congruence and reduction, both defined in Figure 2.1. Unlike other calculi, however, in BoCa these relations are only defined for proper (i.e. well-formed) terms, a fact we will leave implicit in the rest of the presentation. In particular, the congruence $!\pi\,.\,P \equiv \pi\,.\,P \mid !\pi\,.\,P$ only holds with P a proper term of weight 0. Thus, to duplicate arbitrary processes we need to first "freeze" them under a spawner, i.e. we decompose arbitrary duplication into "template replication" and "process activation." Rule (SPAWN) allows also for the spawning of processes of weight zero, since we have identified $\rule{1ex}{0.4ex}^0$ with $\mathbf{0}$.

The notion of structural congruences is mostly standard, with the exception of the law $\mathsf{k} \rhd (\rule{1ex}{0.4ex} \mid P) \equiv (\mathsf{k}-1) \rhd P$, that is specific to our calculus and regulates the interplay between slots and spawning. When used from left to right, this law transforms the term by removing all the unguarded occurrences of slots that appear directly in the scope of a spawning prefix: in doing that, the rule also redefines the weight of the process under spawning so as to preserve the well-formedness of the term. The process that results from the transformation can be thought of as a kind of normal form of the original process in which all unspawned slot occurrences are guarded or enclosed within an ambient. Process normalization is coherent with our interpretation of slots and spawning: for instance, by normalization one has $1 \rhd \rule{1ex}{0.4ex} \equiv \mathbf{0}$, which formalizes our intuition that spawning a free slot should indeed correspond to a no-op. More generally, and more interestingly, normalization enables reductions that would otherwise be blocked. To illustrate, consider the process $P = \mathsf{k} \rhd (\rule{1ex}{0.4ex} \mid P')$ and the context $\mathbf{E}\{\cdot\} = \cdot \mid \rule{1ex}{0.4ex}^{\mathsf{k}-1}$. As given, P cannot be spawned in $\mathbf{E}\{\cdot\}$ as the context provides one fewer slots than required by P. However, we can normalize P and obtain the desired reduction as follows: $\mathbf{E}\{P\} \equiv \mathbf{E}\{(\mathsf{k}-1) \rhd P'\} \searrow P'$.

The reduction relation $\searrow$ formalizes the intuitions discussed in Section 1. In addition, we remark that making the weight of an ambient depend explicitly on its contents allows a clean and simple treatment of ambient opening: opening simply does not require resources, it rather may release those enclosed within the opened ambient. Notice also that **open** is the only capability that requires a co-capability: this means that only consenting ambients can be opened. However, a closer look at the reductions shows that $\rule{1ex}{0.4ex}$ acts as a co-capability for mobility (for non-zero-weighted ambients).

A few further remarks are in order on the form of the transfer. As it can be

Fig. 2. Structural Congruence and Reduction

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

$$(\boldsymbol{\nu} a)(P \mid Q) \equiv (\boldsymbol{\nu} a)P \mid Q (a \notin \mathrm{fn}(Q)) \qquad a[\, (\boldsymbol{\nu} b)P \,] \equiv (\boldsymbol{\nu} b) a[\, P \,] \quad (a \neq b)$$

$$(\boldsymbol{\nu} a)(\boldsymbol{\nu} b)P \equiv (\boldsymbol{\nu} b)(\boldsymbol{\nu} a)P \qquad (M_1.M_2).P \equiv M_1.(M_2.P)$$

$$(\boldsymbol{\nu} a)\mathbf{0} \equiv \mathbf{0} \qquad \mathsf{k} \triangleright (\blacksquare \mid P) \equiv (\mathsf{k}-1) \triangleright P$$

$$!\pi\,.\,P \;\equiv\; \pi\,.\,P \mid !\pi\,.\,P$$

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

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

seen from rules (getS) and (getD), 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.

The format of the **put** and **get** capabilities is asymmetric as **put** offers slots to any (sibling or child) process requesting it, while **get** makes it request to a specifically named context (a sibling or the parent). We select these particular formats because they provide adequate balance between expressivity and control of interferences in the protocols for space allocation and distribution.

In particular, the current choice makes it easy and natural to express interesting programming examples and protocols (cf. the memory management example in Section 2.2, which exemplifies the balance of control and flexibility afforded by our choice). It also enables us to provide a simple encoding of named (and private) resources allocated for spawning (cf. the initial discussion in Section 4). Secondly, a new protocol is easily derived for transferring resources "upwards" from children to parents using the following pair of dual put and get.

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

Transfers affect the amount of resources allocated at different nesting levels in a system. We delegate to the type system of Section 3 to control that no nesting level suffers from resource over- or under-flows. More precisely, our type system associates to each ambient name a two non negative integers n, N such that $\mathsf{n} \leq \mathsf{N}$. The integers n and N can be interpreted, respectively, as the lower ad upper bounds for the weight an ambient is allowed to have. In a well-typed term and in all its reducts the weight k of the process inside an occurrence of a always satisfies $\mathsf{n} \leq \mathsf{k} \leq \mathsf{N}$ (see Theorem 4).

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. Let $\searrow_*$ be the reflexive and transitive closure of $\searrow$. We have:

Proposition 2 (Resource preservation) *If $w(P) \neq \bot$, and $P \searrow_* Q$, then $w(P) = w(Q)$.*

Notice that 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. We will return on this in Section 4, where we study the behavioral semantics of (an extension of) the calculus.

2.2 Examples

We illustrate the calculus with examples and encodings of systems that require usage and control of space. Before that, it is worth noticing that the Ambient Calculus [6] 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:

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

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

2.2.1 Parent-child swap

Given that ambients may have non-null weights, in BoCa this swap is possible only in case the father and child ambients have the same weight. We give the details for the case of weight 1: notice the use of the primitives for child to father slot transfer we defined in Section 2.1.

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

This example will come useful again to illustrate effect inference in Section 3.4.

2.2.2 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

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

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

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

where k is the weight of $P \mid R$. So, an ambient needs to *borrow* space from its parent in order to rename itself. We conjecture that renaming cannot be obtained otherwise.

2.2.3 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.

$$\mathrm{MEM\_MOD} \triangleq mem[\,\blacksquare^{256MB} \mid \overbrace{\mathbf{open}\ m \mid \ldots \mid \mathbf{open}\ m}^{256MB}\,]$$

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

$$\mathrm{USER} \triangleq u[\ \ldots.\mathrm{MALLOC} \mid \ldots.\mathbf{get}\ mem\ldots\mathbf{put} \mid \ldots\ ]$$

Notice that the format of the **get** capability provides fine control on how the space requests are served by the memory module, while keeping the encoding of the system simple. In particular, if a **get** request were not forced to specify a name, the encoding of the memory module would not guarantee the memory to take back spaces exactly from the users it previously allocated it to.

2.2.4 A cab trip

As a further example, we give a new version of the cab trip protocol from [31], 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.

$$
\begin{aligned}
&CALL(from, client) \triangleq \\
&\qquad call^{1}[\mathbf{out}\ client\,.\,\mathbf{out}\ from\,.\,\mathbf{in}\ cab\,.\,\overline{\mathbf{open}}\,.\,\mathbf{in}\ from\,. \\
&\qquad\qquad\qquad (load^{0}[\ \mathbf{out}\ cab\,.\,\mathbf{in}\ client\,.\,\overline{\mathbf{open}}\ ] \mid \rule{0.6em}{0.25em})]
\end{aligned}
$$

$$
\begin{aligned}
&TRIP(from, to, client) \triangleq \\
&\qquad trip^{0}[\ \mathbf{out}\ client\,.\,\overline{\mathbf{open}}\,.\,\mathbf{out}\ from\,.\,\mathbf{in}\ to\,.\,done^{0}[\ \mathbf{in}\ client\,.\,\overline{\mathbf{open}}\ ]\ ]
\end{aligned}
$$

$$
\begin{aligned}
&CLIENT(from, to) \triangleq \\
&\qquad (\boldsymbol{\nu} c)c^{1}[CALL(from, c) \mid \mathbf{open}\ load\,.\,\mathbf{in}\ cab\,.\,TRIP(from, to, c) \\
&\qquad\qquad \mid \mathbf{open}\ done\,.\,\mathbf{out}\ cab\,.\,bye^{0}[\ \mathbf{out}\ c\,.\,\mathbf{in}\ cab\,.\,\overline{\mathbf{open}}\,.\,\mathbf{out}\ to\ ]]
\end{aligned}
$$

$$
\begin{aligned}
&CAB \triangleq \\
&\qquad cab^{1}[\ \rule{0.6em}{0.25em} \mid \mathbf{!}(\mathbf{open}\ call\,.\,\mathbf{open}\ trip\,.\,\mathbf{open}\ bye)\ ]
\end{aligned}
$$

$$
\begin{aligned}
&SITE(i) \triangleq \\
&\qquad site_i[\ CLIENT(site_i, site_j) \mid CLIENT(site_i, site_l) \mid \cdots \mid \rule{0.6em}{0.25em} \mid \rule{0.6em}{0.25em} \mid \cdots\ ]
\end{aligned}
$$

$$
\begin{aligned}
&CITY \triangleq \\
&\qquad city[\ CAB \mid CAB \mid \cdots \mid \cdots \mid SITE(1) \mid \cdots \mid SITE(n) \mid \rule{0.6em}{0.25em} \mid \rule{0.6em}{0.25em} \mid \cdots\ ]
\end{aligned}
$$

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.

2.3 Discussion

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

To see the problem, consider again the cab trip example given above. On the one hand, comparing with [31], we notice that we can deal with the cab's space satisfactorily with no need for 3-way synchronisations. On the other hand, however, as already observed in [31], 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.

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):

$$a^1[\,\mathbf{in}\ b.P\,] \mid b[\,1 \triangleright Q \mid \rule{0.5em}{0.15em} \mid d[\,c^1[\,\mathbf{out}\ d.R\,]\,]\,]$$

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 Section 3 provides static guarantees that the resources available at a given site remain within the intended bounds. The mechanism for slot naming, in Section 4, yields new and effective primitives for the selective distribution of resources, and for protecting processes against the presence of races for resource acquisition.

3 Bounding Resources by Typing

As we mentioned above, the type system provides static guarantees for a simple behavioural property, namely that the transfers of space exercised within each ambient preserve the resource bounds imposed by the type of that ambient. To deal with this satisfactorily, we need to take into account that transfer capabilities can be acquired by way of exchanges. The type of capabilities must therefore inform on how a capability may affect the space of the ambient in which it is exchanged and exercised.

3.1 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:

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

The intervals ι provide a representation for the capacity bounds that ambients impose on their enclosed processes, while the effects ε and the thread effects ϕ help characterize the behavior of processes and capabilities, respectively. The syntax of types is defined in terms of the previous domains as follows:

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

$\mathit{Amb}\langle \iota, \varepsilon, \chi \rangle$ is the type of ambients with weight ranging in ι, and enclosing processes with ε effects and χ exchanges. The exchanges types χ are defined exactly as in companion type systems for ambient calculi, and include ambient types and capability types, with the type *Shh* indicating no exchange.

The interesting, and distinguishing part of the type system is in the computation of the types and effects associated with capabilities and processes. $\mathit{Proc}\langle \varepsilon, \chi \rangle$ is the type of processes with ε effects and χ exchanges. Specifically, for a process P of type $\mathit{Proc}\langle (\mathsf{d},\mathsf{i}), \chi \rangle$, the effect (d,i) bounds the number of slots delivered ($|\mathsf{d}|$) and acquired (i) by P as the cumulative result of exercising P's transfer capabilities. As for capabilities, $\mathit{Cap}\langle \phi, \chi \rangle$ is the type of (paths of) capabilities that have thread effect ϕ and that, when exercised, unleash processes with χ exchanges.

To understand the typing of capabilities, and the nature of thread effects it is useful to start with a simpler idea for computing capability types. A first approach to typing the transfer capabilities is to assign the effects $(0, 1)$ to each **get**, and $(-1, 0)$ to each **put**. These effects provide a sound (and accurate) estimate of the computational consequences of the corresponding capabilities as, indeed, a **get** acquires one slot (and delivers none), while a **put** delivers one slot (and acquires none). Such estimates are accurate also when the transfer capabilities are composed in parallel, as in that case the most accurate, sound estimate of the effect of the capabilities is obtained as the sum of the composing effects. For instance, for the effect of the parallel composition **put | get** a there is no better estimate than $(-1, 1)$, which is the point-wise sum of the effects $(-1, 0)$ and $(0, 1)$.

On the other hand, if the two capabilities are part of the same thread, then the previous estimate is overly loose. To see the point notice that $(-1, 0)$ is a sound (and tighter) estimate for **put. get** a, as the slot acquired by get simply "fills in" for the slot delivered by the previous put or, dually, the effect of the put compensates the slot acquired by the subsequent get. By the same reasoning, **get** a **. put** has the cumulative effect $(0, 1)$.

Generalizing to the case of an arbitrary process P, if P has effect (d, i), the effect of **put.** P will be $(\mathsf{d} - 1, \mathsf{i} - 1)$ $((\mathsf{d} - 1, 0)$ if $\mathsf{i} - 1 < 0)$, i.e. the effect of P "shifted downwards" by 1. Dually, the effect of **get** a **.** P will be $(\mathsf{d} + 1, \mathsf{i} + 1)$ $((0, \mathsf{i} + 1)$ if $\mathsf{d} + 1 > 0)$, i.e. the effect of P "shifted upwards" by 1. More formally, we may represent these "shifts" by resorting the following combinators (functions in Φ):

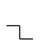
$$\begin{aligned}\mathsf{Put} &= \lambda(\mathsf{d}, \mathsf{i}).(\mathsf{d} - 1, \mathsf{max}(0, \mathsf{i} - 1))\\ \mathsf{Get} &= \lambda(\mathsf{d}, \mathsf{i}).(\mathsf{min}(0, \mathsf{d} + 1), \mathsf{i} + 1)\end{aligned}$$

Based on these definitions, if P has effect ε, the effect of **put.** P and **get** a **.** P may be computed as $\mathsf{Put}(\varepsilon)$ and $\mathsf{Get}(\varepsilon)$, respectively. The use of functional combinators also allows a rather elegant and concise representation of the cumulative effect of paths of capabilities in terms of functional composition. To illustrate the cumulative effect of the thread

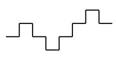
$$\mathbf{get}\ a\,\mathbf{.\,put.\,put.\,get}\ a\,\mathbf{.\,get}\ a\,\mathbf{.\,get}\ a\,\mathbf{.\,put}$$

is represented by the following effect expression (where $\circ$ denotes function composition in standard order, i.e. $f \circ g(x) = f(g(x))$):

$$\varepsilon \ = \ (\mathsf{Get} \circ \mathsf{Put} \circ \mathsf{Put} \circ \mathsf{Get} \circ \mathsf{Get} \circ \mathsf{Get} \circ \mathsf{Put})((0, 0)) \ = \ (-1, 2).$$

An alternative, somewhat more direct computation of the thread effect may be obtained by resorting to a graphical presentation of the effects of puts and gets [1] . Specifically, we **get** as "one step up" and of a **put** as "one step down" in the slot count and represent them graphically as and  respectively.

A sequence of **get**'s and **put**'s is then represented by the graph obtained by connecting the steps in the order in which they appear. For the previous example, the effect of the sequence

get a **. put. put. get** a **. get** a **. get** a **. put** is 

The cumulative effect of the sequence is computed by taking the lowest and highest levels of the graph, which in our example are -1 and 2 as expected.

[1] We gratefully acknowledge the anonymous referees for suggesting this representation.

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 ε declared by n to bound the effect of the unleashed process: that effect is then added to the effect of the continuation P. In analogy with the previous development, we introduce the following combinator:

$$\mathsf{Open}(\varepsilon) = \lambda(\mathsf{d}, \mathsf{i}).(\varepsilon + (\mathsf{d}, \mathsf{i}))$$

where $+$ denotes component-wise sum for effects, defined as follows: $(\mathsf{d}, \mathsf{i}) + (\mathsf{d}', \mathsf{i}') = (\mathsf{d}+\mathsf{d}', \mathsf{i}+\mathsf{i}')$. Then, if P has effect ε', the composite effect of **open** $n.P$ is computed as $\mathsf{Open}(\varepsilon)(\varepsilon') = \varepsilon + \varepsilon'$.

We could make the analysis more precise as done by Levi and Sangiorgi in [18]. In fact, thanks to the presence of $\overline{\textbf{open}}$, we know at which point in its life a given ambient can be opened, so we can give a tight estimate of the effect that remains by then (this is the effect unleashed by opening). We do not pursue this direction as it is orthogonal to our present development and it would complicate the type system.

3.2 The typing rules

The typing rules are collected in Figures 3 and 4, where we denote with id_Φ the identity element in the domain Φ. The rules rely on partial orders over intervals and effects. Such orders arise as expected, namely: $[\mathsf{n}, \mathsf{N}] \leq [\mathsf{n}', \mathsf{N}']$ when $\mathsf{n}' \leq \mathsf{n}$ and $\mathsf{N} \leq \mathsf{N}'$ and $(\mathsf{d}, \mathsf{i}) \leq (\mathsf{d}', \mathsf{i}')$ when $\mathsf{d}' \leq \mathsf{d}$ and $\mathsf{i} \leq \mathsf{i}'$.

The rules in Figure 3 derive judgements $\Gamma \vdash M : W$ for well-typed messages. The environment Γ 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 co-capability $\overline{\textbf{open}}$ have no effect, as reflected by the use of id_Φ in their type. The same is true also of the co-capability $\textbf{put}^{\downarrow}$. In fact, the weight of an ambient in which $\textbf{put}^{\downarrow}$ is executed does not change: the ambient loses one slot, but the weight of one of its sub-ambients increases (cf. reduction rule (GETD)).

The rules in Figure 4 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 $\mathsf{0}_\mathcal{E}$ denote the null

Fig. 3. Good Messages

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

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

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

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

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

effect $(0, 0)$: thus, rules ($\mathbf{0}$) and ($\_$) simply state that the inert process and the slot are processes with no effects.

Rule (*prefix*) computes the effects of prefixes, by applying the thread effect of the capability to the effect of the process, while rule (*par*) adds up the effects of two parallel threads. The constructs for input, output and restriction are standard. Rule (*amb*) governs the formation of ambient processes. The declared weight k of the ambient must reflect the weight of the enclosed process. Two further conditions ensure (*i*) that k modified by the effect (d, i) of the enclosed process lies within the interval $[\mathsf{n}, \mathsf{N}]$ declared by the ambient type (i.e. we get $\mathsf{n} \leq \mathsf{k} \leq \mathsf{N}$), and (*ii*) that the effect ε 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 $[\mathsf{max}(\mathsf{k} + \mathsf{d}, 0), \mathsf{k} + \mathsf{i}] \leq [\mathsf{n}, \mathsf{N}]$, where the use of $\mathsf{max}(\mathsf{k} + \mathsf{d}, 0)$ 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 (d, i) may only release effects $\varepsilon \geq (\mathsf{d} - \mathsf{i}, \mathsf{i} - \mathsf{d})$. The lower bound arises in a situation in which the ambient is opened right after the enclosed process has completed i **get**'s and is 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 $|\mathsf{d}|$ **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

Fig. 4. Good Processes

$$\frac{}{\Gamma \vdash \mathbf{0} : \mathit{Proc}\langle 0_{\mathcal{E}}, \chi\rangle}\ (\mathbf{0}) \qquad\qquad \frac{}{\Gamma \vdash \rule{0.6em}{0.25em} : \mathit{Proc}\langle 0_{\mathcal{E}}, \chi\rangle}\ (\rule{0.6em}{0.25em})$$

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

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

$$\frac{\Gamma, x : W \vdash P : \mathit{Proc}\langle \varepsilon, W\rangle}{\Gamma \vdash (x : W)P : \mathit{Proc}\langle \varepsilon, W\rangle}\ (\mathit{input}) \qquad \frac{\Gamma \vdash M : W \quad \Gamma \vdash P : \mathit{Proc}\langle \varepsilon, W\rangle}{\Gamma \vdash \langle M\rangle P : \mathit{Proc}\langle \varepsilon, W\rangle}\ (\mathit{output})$$

$$\frac{\Gamma, a : \mathit{Amb}\langle \iota, \varepsilon, \chi\rangle \vdash P : \mathit{Proc}\langle \varepsilon', \chi'\rangle}{\Gamma \vdash (\boldsymbol{\nu} a : \mathit{Amb}\langle \iota, \varepsilon, \chi\rangle)P : \mathit{Proc}\langle \varepsilon', \chi'\rangle}\ (\mathit{new\ pr})$$

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

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

two observations justify the condition $(\mathsf{d}-\mathsf{i}, \min(\mathsf{N}-\mathsf{n}, \mathsf{i}-\mathsf{d})) \leq \varepsilon$ in rule (*amb*). Notice that the more symmetric form of the previous condition, i.e. $(\max(\mathsf{n}-\mathsf{N}, \mathsf{d}-\mathsf{i}), \min(\mathsf{N}-\mathsf{n}, \mathsf{i}-\mathsf{d})) \leq \varepsilon$, is not sound, since we could derive $\{m : \mathit{Amb}\langle [0,0], (0,0), -\rangle\} \vdash m[\ \mathbf{put}\ ] : \mathit{Proc}\langle 0_{\mathcal{E}}, -\rangle$, while opening m may cause the loss of one slot.

Rule (*spawn*) simply checks that the process is well-formed and then defines the effect of $\mathsf{k} \triangleright P$ as the effect 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.

The following result complements Proposition 2 and shows that capacity bounds on ambients are preserved during computations, while the effects of processes reduce.

Theorem 3 (Subject Reduction) *Assume* $\Gamma \vdash P : \mathit{Proc}\langle \varepsilon, \chi\rangle$ *and* $P \searrow_* Q$. *Then* $\Gamma \vdash Q : \mathit{Proc}\langle \varepsilon', \chi\rangle$ *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. Rule (EXCHANGE) follows from a standard substitution lemma which can be easily proved. The interesting cases are those for rules (GETS), (GETD) and (OPEN). 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:

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

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:

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

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

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

has the following conditions:

$$[\mathsf{max}(\mathsf{k}+1+\mathsf{d}-1+\mathsf{d}',0), \mathsf{k}+1+\mathsf{max}(\mathsf{i}-1,0)+\mathsf{i}'] \le \iota$$

$$(\mathsf{d}-1+\mathsf{d}'-\mathsf{max}(\mathsf{i}-1,0)-\mathsf{i}', \mathsf{min}(\mathsf{N}-\mathsf{n}, \mathsf{max}(\mathsf{i}-1,0)+\mathsf{i}'-\mathsf{d}+1-\mathsf{d}')) \le \varepsilon,$$

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

$$[\mathsf{max}(\mathsf{k}+\mathsf{d}+\mathsf{d}',0), \mathsf{k}+\mathsf{i}+\mathsf{i}'] \le \iota$$

$$(\mathsf{d}+\mathsf{d}'-\mathsf{i}-\mathsf{i}', \mathsf{min}(\mathsf{N}-\mathsf{n}, \mathsf{i}+\mathsf{i}'-\mathsf{d}-\mathsf{d}')) \le \varepsilon.$$

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

For rule (GETD) let $\mathcal{D}$ be a derivation of:

$$\Gamma \vdash \mathbf{put}^{\downarrow}.\,P \mid \_ \mid a^{\mathsf{k}}[\,\mathbf{get}^{\uparrow}.\,Q \mid R\,] : \mathit{Proc}\langle \varepsilon\rangle$$

An inspection of the typing rules – (*put*$^{\downarrow}$), ($\_$), (*get*$^{\uparrow}$), (*prefix*), (*amb*), and (*par*) – shows that inside $\mathcal{D}$ there must be judgements of the shape:

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

for some $\iota, \varepsilon', \mathsf{i}, \mathsf{i}', \mathsf{d}, \mathsf{d}'$. By rules ($get^{\uparrow}$) and (*prefix*), $\Gamma \vdash Q : \mathit{Proc}\langle(\mathsf{d},\mathsf{i})\rangle$ implies $\Gamma \vdash \mathbf{get}^{\uparrow}.\, Q : \mathit{Proc}\langle(\mathsf{min}(0, \mathsf{d}+1), \mathsf{i}+1)\rangle$. Then the application of rule (*amb*) for deriving

$$\Gamma \vdash a^{\mathsf{k}}[\, \mathbf{get}^{\uparrow}.\, Q \mid R\,] : \mathit{Proc}\langle 0_{\mathcal{E}}\rangle$$

has the following conditions:

$$[\mathsf{max}(\mathsf{k} + \mathsf{min}(0, \mathsf{d}+1) + \mathsf{d}', 0), \mathsf{k} + \mathsf{i} + 1 + \mathsf{i}'] \leq \iota$$

$$(\mathsf{min}(0, \mathsf{d}+1) + \mathsf{d}' - \mathsf{i} - 1 - \mathsf{i}', \mathsf{min}(\mathsf{N} - \mathsf{n}, \mathsf{i} + 1 + \mathsf{i}' - \mathsf{min}(0, \mathsf{d}+1) - \mathsf{d}')) \leq \varepsilon',$$

where $\iota = [\mathsf{n}, \mathsf{N}]$. The conditions required to apply rule (*amb*) for the derivation of the judgement $\Gamma \vdash a^{\mathsf{k}+1}[\, \_ \mid Q \mid R\,] : \mathit{Proc}\langle 0_{\mathcal{E}}\rangle$ are

$$[\mathsf{max}(\mathsf{k} + 1 + \mathsf{d} + \mathsf{d}', 0), \mathsf{k} + 1 + \mathsf{i} + \mathsf{i}'] \leq \iota$$

$$(\mathsf{d} + \mathsf{d}' - \mathsf{i} - \mathsf{i}', \mathsf{min}(\mathsf{N} - \mathsf{n}, \mathsf{i} + \mathsf{i}' - \mathsf{d} - \mathsf{d}')) \leq \varepsilon'.$$

These last conditions can be shown to follow from the previous ones by easy algebraic manipulations. From the last judgement and from $\Gamma \vdash P : \mathit{Proc}\langle\varepsilon\rangle$, by rule (*par*) we conclude $\Gamma \vdash P \mid a^{\mathsf{k}+1}[\, \_ \mid Q \mid R\,] : \mathit{Proc}\langle\varepsilon\rangle$, as desired.

For rule (OPEN) let $\mathcal{D}$ be a derivation of:

$$\Gamma \vdash \mathbf{open}\ a.\, P \mid a[\, \overline{\mathbf{open}}.\, Q \mid R\,] : \mathit{Proc}\langle\varepsilon\rangle$$

Inside $\mathcal{D}$ there must be judgements of the shape $\Gamma \vdash a : \mathit{Amb}\langle\iota, \varepsilon'\rangle$, $\Gamma \vdash P : \mathit{Proc}\langle(\mathsf{d},\mathsf{i})\rangle$, $\Gamma \vdash Q : \mathit{Proc}\langle(\mathsf{d}',\mathsf{i}')\rangle$, and $\Gamma \vdash R : \mathit{Proc}\langle(\mathsf{d}'',\mathsf{i}'')\rangle$ for some $\iota, \varepsilon', \mathsf{i}, \mathsf{i}', \mathsf{i}'', \mathsf{d}, \mathsf{d}', \mathsf{d}''$. By rule (*par*) we can derive:

$$\Gamma \vdash P \mid Q \mid R : \mathit{Proc}\langle(\mathsf{d} + \mathsf{d}' + \mathsf{d}'', \mathsf{i} + \mathsf{i}' + \mathsf{i}'')\rangle.$$

By rules (*open*) and (*prefix*) we get:

$$\varepsilon = \varepsilon' + (\mathsf{d}, \mathsf{i}).$$

The conditions for using rule (*amb*) in deriving

$$\Gamma \vdash a[\, \overline{\mathbf{open}}.\, Q \mid R\,] : \mathit{Proc}\langle 0_{\mathcal{E}}\rangle$$

are:

$$[\mathsf{max}(\mathsf{k} + \mathsf{d}' + \mathsf{d}'', 0), \mathsf{k} + \mathsf{i}' + \mathsf{i}''] \leq \iota$$

$$(\mathsf{d}' + \mathsf{d}'' - \mathsf{i}' - \mathsf{i}'', \mathsf{min}(\mathsf{N} - \mathsf{n}, \mathsf{i}' + \mathsf{i}'' - \mathsf{d}' - \mathsf{d}'')) \leq \varepsilon',$$

where $\iota = [\mathsf{n}, \mathsf{N}]$ and $\mathsf{k} = w(Q \mid R)$. These conditions imply $(\mathsf{d}' + \mathsf{d}'', \mathsf{i}' + \mathsf{i}'') \leq \varepsilon'$, from which we $(\mathsf{d} + \mathsf{d}' + \mathsf{d}'', \mathsf{i} + \mathsf{i}' + \mathsf{i}'') \leq \varepsilon$, and this concludes our proof.

It follows as a direct corollary that no ambient may be subject to over/underflow during the computation. Given that the calculus per se admits no notion of over/underflow these notions are about the soundness of our type system to the extent that the latter is meant to enforces the allocation policies.

We use $\mathcal{C}[\cdot]$ to denote an arbitrary context (that is, a term with a "hole", not an evaluation context as defined in Fig.2.1).

Theorem 4 (Type Soundness) *If* $\Gamma, a : \mathit{Amb}\langle[\mathsf{n},\mathsf{N}], -, -\rangle \vdash P : \mathit{Proc}\langle\varepsilon,\chi\rangle$, $P \searrow_* \mathcal{C}[a^{\mathsf{k}}[\, R\,]]$, *and the showed occurrence of* a *is not in the scope of a binder for* a*, then* $\mathsf{n} \leq \mathsf{k} \leq \mathsf{N}$.

The proviso "not in the scope of a binder for a" is needed since the environment Γ 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.

3.3 Typed Examples

We illustrate the typing system at work with a typed version of the memory module of Section 2.2. We start with the *malloc* ambient

$$\textsc{malloc} \triangleq m[\, \mathbf{out}\ u\,.\,\mathbf{in}\ \mathit{mem}\,.\,\overline{\mathbf{open}}\,.\,\mathbf{put}\,.\,\mathbf{get}\ u\,.\,\mathbf{open}\ m\,]$$

Since there are no exchanges, we give the typing annotation and derivation disregarding the exchange component of the types. Let P_{malloc} denote the thread enclosed within the ambient m. If we let $m : \mathit{Amb}\langle[0,0],(-1,0)\rangle \in \Gamma$, an inspection of the typing rules for capabilities and paths shows that the following typing is derivable for any ambient type assigned to mem:

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

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

$$\textsc{mem\_mod} \triangleq \mathit{mem}[\, \blacksquare^{256\text{MB}} \mid \mathbf{open}\ m \mid \ldots \mid \mathbf{open}\ m\,]$$

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

For the remaining examples from Subsection 2.2, those that do not use the open and the transfer capabilities are easily seen to typecheck.

The parent-child swap process of Example 2.2.1 can be typed assuming the types $Amb\langle[0,1],(-1,1)\rangle$ and $Amb\langle[0,2],(-3,2)\rangle$ for ambients a and b respectively. The more demanding type for b is due to the encoding of capability $\mathbf{get}^{\downarrow}a$ using an ambient opening. We will show the correctness of this typing as an application of the effect inference in Subsection 3.4.

Finally, in the protocol of Example 2.2.4, one verifies that all the processes typecheck, by taking $W_1 = Amb\langle[1,1],\mathsf{0}_{\mathcal{E}}\rangle$, $W_0 = Amb\langle[0,0],\mathsf{0}_{\mathcal{E}}\rangle$, by declaring the c with W_1 and assuming the following types for the free names $call : W_1$, $cab : W_1$, $trip : W_0$, $load : W_0$, $done : W_0$, $bye : W_0$.

3.4 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 $mem : Amb[0,256\text{MB}]$ 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 $\mathcal{V}$). It is not difficult to infer effects for the whole calculus following the approach of [3], but it requires the introduction of variables ranging over communication types.

The ambient types give only the range ι 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 typable.

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:

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

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

Fig. 5. Effect Inference

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

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

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

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

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

$$\frac{\Gamma \vdash P : \mathit{Proc}\langle \varepsilon\rangle \Downarrow \Delta \quad w(P) = \mathsf{k}}{\Gamma \vdash \mathsf{k} \triangleright P : \mathit{Proc}\langle \varepsilon\rangle \Downarrow \Delta}\ \widehat{(\mathit{spawn})}$$

$$\frac{\Gamma \vdash \pi\,.\,P : \mathit{Proc}\langle 0_{\mathcal{E}}\rangle \Downarrow \Delta \quad w(\pi\,.\,P) = 0}{\Gamma \vdash \mathbf{!}\pi\,.\,P : \mathit{Proc}\langle 0_{\mathcal{E}}\rangle \Downarrow \Delta}\ \widehat{(\mathit{bang})}$$

expressions, that is:

$$\begin{array}{rl} \textit{Effects} & \varepsilon \in \mathcal{E} \triangleq \{(e, e') \mid e, e' \in \mathit{EXP}\} \\ \textit{Thread Effects} & \phi \in \Phi \triangleq \mathcal{E} \to \mathcal{E} \end{array}$$

The rules for messages in our inference system are the same as those in Figure 3, except for the rule (*open*) which becomes:

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

The inference rules are in Figure 5 and the judgements they derive have the shape $\Gamma \vdash P : \mathit{Proc}\langle \varepsilon\rangle \Downarrow \Delta$. This means that in our system, beside providing a process P with a process type $\mathit{Proc}\langle \varepsilon \rangle$, we produce for P also a set of inequality constraints Δ, 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.

To show that constraint satisfiability is decidable we rely on few simple transformations of the constraints generated by the inference system. First, we get rid of the occurrences of $\mathsf{min}()$ and $\mathsf{max}()$ by means of a standard linear programming technique (see for example [23]). Specifically, each constraint $\mathsf{min}(\mathsf{N} - \mathsf{n}, e' - e) \leq \mathtt{i}_a$ can be replaced by the following boolean formula:

$$(\mathsf{N} - \mathsf{n} \leq e' - e \ \wedge \ \mathsf{N} - \mathsf{n} \leq \mathtt{i}_a) \ \vee \ (e' - e \leq \mathsf{N} - \mathsf{n} \ \wedge \ e' - e \leq \mathtt{i}_a).$$

Similarly each constraint $\mathsf{n} \leq \mathsf{max}(\mathsf{k} + e, 0)$ is equivalent to the following disjunction of constraints (remember that n is non-negative by definition):

$$\mathsf{n} = 0 \ \vee \ \mathsf{n} \leq \mathsf{k} + e.$$

Thus, all the constraints generated by the inference system can equivalently be expressed in terms of a formula in quantifier free Presburger arithmetic, whose satisfiability is known to be a NP-complete problem.

More precisely, for a process with p occurrences of ambient constructors and q different ambient names we obtain a formula with ℓ inequality constraints (where $6\mathsf{p} \leq \ell \leq 7\mathsf{p}$ since the value of n is given) and $2\mathsf{q}$ unknows.

There are many algorithms and tools for solving the satisfiability problem of a quantifier free Presburger formula. In our case we can observe that:

- We can obtain a formula in disjunctive normal form with 2^{p} disjuncts, each one consisting of ℓ (where $4\mathsf{p} \leq \ell \leq 5\mathsf{p}$) inequality constraints. We can solve efficiently these disjuncts using an integer linear programming solver, like for example [25].
- The solution space of our formula can be easily determined by taking into account that the last two inequalities generated by the application of rule $\widehat{(amb)}$ imply respectively $\mathtt{d}_a \leq 0$ and $\mathtt{i}_a \geq 0$. Moreover we have $-\mathsf{h} \leq \mathtt{d}_a$ and $\mathtt{i}_a \leq \mathsf{h}$ where h is the number of put and get capabilities occurring in

the current process. In particular this gives $\mathtt{d}_a = \mathtt{i}_a = 0$ for all processes not containing put and get capabilities. Notice that h is linear in the dimension of the current process, while the general bound of the solution space depends on 2^{p} (since the number of inequality constraints is linear in p), which in our case is exponential in the dimension of the current process. Therefore we can translate our formula to a Boolean formula by encoding each integer variable as a vector of Boolean variables of length $\mathtt{log}\,\mathsf{h}$. The resulting Boolean formula can be checked using a Boolean satisfiability solver, see [1].

We illustrate the inference algorithm on the process of Example 2.2.1. First note that the effect of the capability $\mathbf{put}\,.\,\mathbf{in}\ a\,.\,\mathbf{get}^{\uparrow}$ is $\mathsf{Put} \circ \mathsf{Get}$. Then, applying the inference rules we obtain $\vdash \mathbf{put}\,.\,\mathbf{in}\ a\,.\,\mathbf{get}^{\uparrow} : \mathit{Proc}\langle(-1,0)\rangle \Downarrow \emptyset$ and $\{m : \mathit{Amb}^{-}\langle[0,1]\rangle\} \vdash \mathbf{open}\ m\,.\,\mathbf{put}\,.\,\mathbf{in}\ a\,.\,\mathbf{get}^{\uparrow} : \mathit{Proc}\langle(\mathtt{d}_m - 1, \mathtt{i}_m)\rangle \Downarrow \emptyset$. From $\{m : \mathit{Amb}^{-}\langle[0,1]\rangle\} \vdash \mathbf{get}\ a\,.\,\overline{\mathbf{open}} : \mathit{Proc}\langle(0,1)\rangle \Downarrow \emptyset$ we derive (valid ground inequalities are not shown)

$$\{m : \mathit{Amb}^{-}\langle[0,1]\rangle\} \vdash m^0[\ \mathbf{get}\ a\,.\,\overline{\mathbf{open}}\ ] : \mathit{Proc}\langle 0_{\mathcal{E}}\rangle \Downarrow \{\mathtt{d}_m \le -1, 1 \le \mathtt{i}_m\}.$$

It follows

$$\vdash (\boldsymbol{\nu} m : \mathit{Amb}^{-}\langle[0,1]\rangle)(\mathbf{open}\ m\,.\,\mathbf{put}\,.\,\mathbf{in}\ a\,.\,\mathbf{get}^{\uparrow} \mid m^0[\ \mathbf{get}\ a\,.\,\overline{\mathbf{open}}\ ]) : \\ \mathit{Proc}\langle(\mathtt{d}_m - 1, \mathtt{i}_m)\rangle \Downarrow \{\mathtt{d}_m \le -1, 1 \le \mathtt{i}_m\}$$

i.e., $\vdash \mathbf{get}^{\downarrow}a\,.\,\mathbf{put}\,.\,\mathbf{in}\ a\,.\,\mathbf{get}^{\uparrow} : \mathit{Proc}\langle(\mathtt{d}_m - 1, \mathtt{i}_m)\rangle \Downarrow \{\mathtt{d}_m \le -1, 1 \le \mathtt{i}_m\}$, where $\mathbf{get}^{\downarrow}a$ is as defined in Subsection 2.1.
Since

$$\vdash \mathbf{put}^{\uparrow}\,.\,\mathbf{out}\ b\,.\,\mathbf{get}\ b\,.\,\mathbf{put}^{\downarrow} \mid \_ : \mathit{Proc}\langle(-1,0)\rangle$$

we can derive

$$a : \mathit{Amb}^{-}\langle[0,1]\rangle \vdash a^1[\ \mathbf{put}^{\uparrow}\,.\,\mathbf{out}\ b\,.\,\mathbf{get}\ b\,.\,\mathbf{put}^{\downarrow} \mid \_\ ] \\ : \mathit{Proc}\langle 0_{\mathcal{E}}\rangle \Downarrow \{\mathtt{d}_a \le -1, 1 \le \mathtt{i}_a\}$$

Hence, it follows that

$$a : \mathit{Amb}^{-}\langle[0,1]\rangle \vdash \mathbf{get}^{\downarrow}a\,.\,\mathbf{put}\,.\,\mathbf{in}\ a\,.\,\mathbf{get}^{\uparrow} \mid a^1[\ \mathbf{put}^{\uparrow}\,.\,\mathbf{out}\ b\,.\,\mathbf{get}\ b\,.\,\mathbf{put}^{\downarrow} \mid \_\ ] \\ : \mathit{Proc}\langle(\mathtt{d}_m - 1, \mathtt{i}_m)\rangle \Downarrow \{\mathtt{d}_m \le -1, 1 \le \mathtt{i}_m, \mathtt{d}_a \le -1, 1 \le \mathtt{i}_a\}$$

By considering in the context a sound assumption for b, i.e. $b : \mathit{Amb}^{-}\langle[0,2]\rangle$, a further application of rule (amb) produces the following final set of constraints

$$\{\mathtt{d}_m \le -1, 1 \le \mathtt{i}_m, \mathtt{d}_a \le -1, 1 \le \mathtt{i}_a, 1 + \mathtt{i}_m \le 2, \\ \mathtt{d}_b \le \mathtt{d}_m - 1 - \mathtt{i}_m, \mathsf{min}(2, \mathtt{i}_m - \mathtt{d}_m + 1) \le \mathtt{i}_b\}$$

for the whole process

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

It is easy now to check that the set is satisfiable if we take $\mathtt{d}_m = -1$, $\mathtt{i}_m = 1$, $\mathtt{d}_a = -1$, $\mathtt{i}_a = 1$, $\mathtt{d}_b = -3$, $\mathtt{i}_b = 2$, and that we can thus type the process by assuming $Amb\langle[0,1],(-1,1)\rangle$ and $Amb\langle[0,2],(-3,2)\rangle$ for ambients a and b, respectively.

The type inference system can readily be extended so as to make it derive also the types of ambients in additions to the effects. In particular, it suffices to consider, for each name a, two additional integer variables $\mathtt{n}_a$, $\mathtt{N}_a$ and replace them respectively for n, N in rule (amb). This way we obtain a set of constraints where also $\mathtt{n}_a$ and $\mathtt{N}_a$ are unknown. If the constraints can be satisfied, then there is also a solution such that $0 \leq \mathtt{n}_a \leq \mathsf{k} \leq \mathtt{N}_a$, where k is the weight of the process P. We do not develop this extension further as we believe that, in order to specify resource control policies, ambient types ought to be given rather than inferred.

As last remark, we note that $Proc\langle\varepsilon\rangle \Downarrow \Delta$ can be seen as the *principal typing* of the processes P in the sense of [32]. As such, the type system provides for a compositional type analysis, where analyzing a fragment uses only the results for its subfragments, which can be analyzed independently in any order. Compositional analysis is appealing, as it helps with separate compilation and modularity and with making a complete/terminating analysis algorithm.

4 Controlling Resource Races

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 Subsection 2.3, 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 $\blacksquare_a^{\mathsf{k}} \triangleq a[\,\mathbf{put}^{\mathsf{k}}.\overline{\mathbf{open}} \mid \blacksquare^{\mathsf{k}}\,]$, where $\mathbf{put}^{\mathsf{k}}$ stands for $\underbrace{\mathbf{put}.\dots.\mathbf{put}}_{k}$, and

$$\mathsf{k} \triangleright (a, P) \;\triangleq\; (\boldsymbol{\nu} n)(n[\,(\mathbf{get}\ a)^{\mathsf{k}}.\mathsf{k} \triangleright \overline{\mathbf{open}}.P\,] \mid \mathbf{open}\ n.\mathbf{open}\ a)$$

Then, assuming $w(P) = \mathsf{k}$, one has $(\boldsymbol{\nu} a)(\blacksquare_a^{\mathsf{k}} \mid \mathsf{k} \triangleright (a, P))$ “behaves as” P, as desired.[2] A potential problem with this approach is that $\blacksquare_a^2$ is no longer

[2] Using the labelled transition system defined in Section 5.1 one can show that these two processes are barbed congruent in the sense of Definition 6.

congruent to $\boldsymbol{-}^1_a \mid \boldsymbol{-}^1_a$. This has the consequence that, e.g., $\boldsymbol{-}^1_a \mid \boldsymbol{-}^1_a \mid 2 \triangleright (a, P)$ reduces to $a[\ \overline{\textbf{open}}\ ] \mid P$ instead of the expected P, which yields garbage processes, and leaves room for interferences with other processes. It is also possible to encode a form of "resource renaming," by defining:

$$\{a/b\}.P \triangleq (\nu n)(a[\ \textbf{get}\ b.n[\ \textbf{out}\ a.\overline{\textbf{open}}.\textbf{open}\ b\ ] \mid \textbf{put}.\overline{\textbf{open}}\ ] \mid \textbf{open}\ n.P)$$

Then, a b-resource can be turned in to an a-resource: $\{a/b\}.P \mid \boldsymbol{-}_b \searrow_* P \mid \boldsymbol{-}_a$. Observe here that the messages $\textbf{put}.\overline{\textbf{open}}$ could be consumed before ambient n gets out of a, so that this ambient and process P might be blocked. As an example, consider the process $\{a/b\}.P \mid \boldsymbol{-}_b \mid 1 \triangleright (a, Q)$. There is a possible reduction leading to $P \mid Q$ as expected, but there is another one leading to $(\nu n)(n[\ \textbf{out}\ a.\overline{\textbf{open}}.\textbf{open}\ b\ ] \mid \textbf{open}\ n.P) \mid b[\ \overline{\textbf{open}}\ ] \mid Q$.

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 an a-slot for ambients named a. 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:

$$\begin{aligned}(\boldsymbol{\nu} p, q)(p[\ \textbf{in}\ b\,.\,\textbf{get}^{\mathsf{k}} q\,.\,\mathsf{k} \triangleright \overline{\textbf{open}}\,.\,a^{\mathsf{k}}[\ Q\ ]\ ]\\ \mid\ b[\ P \mid q[\ \boldsymbol{-}^{\mathsf{k}} \mid \textbf{put}^{\mathsf{k}}.\overline{\textbf{open}}\ ] \mid \textbf{open}\ p \mid \textbf{open}\ q\ ])\end{aligned}$$

where k is the weight of Q.

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 cannot be defined in the current calculus.

4.1 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 $\boldsymbol{-}_\eta$, where $\eta \in \mathcal{N} \cup \mathcal{V} \cup \{*\}$ is the unit name. To make the new calculus a conservative extension of the one presented in Subsection 2, 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 Section 2.

$$\begin{array}{llll}
\textit{Processes} & P ::= \text{▬}_\eta \mid M^{\mathsf{k}}[\, P\,]_\eta \mid \dots & \text{as in Section 2} \\
\textit{Capabilities} & C ::= \mathbf{get}\, M_\eta \mid \mathbf{get}^{\uparrow}{}_\eta \mid \dots & \text{as in Section 2} \\
\textit{Messages} & M ::= \dots & \text{as in Section 2} \\
\textit{Prefixes} & \pi ::= \mathsf{k}{\triangleright}_\eta \mid \dots & \text{as in Section 2}
\end{array}$$

Again, a (well-formed) *term* is a preterm such that in any subterm of the form $a^{\mathsf{k}}[P]$ or $\mathsf{k} \triangleright_\eta P$, the process P has weight k. The weight of a process can be computed by rules similar to those of Section 2. The anonymous slots ▬_* will be often denoted simply as ▬.

An alternative to the present definition of weight for the refined calculus would be to define $w_\eta(P)$ as the number of ▬_η in P, and then require $w_\eta(P) = \mathsf{k}$ in order for $\mathsf{k}{\triangleright}_\eta P$ to be well-formed. We do not follow such avenue in the present exposition, as it would complicate the spawning of processes containing slots with different tags.

The dynamics of the refined calculus is again defined by means of structural congruence and reduction. Structural congruence is exactly as in Figure 2.1, the top-level reductions are defined in Figure 6.

Fig. 6. Top-level reductions with named units

The reductions for ambient opening and exchanges are as in Figure 2.1, and the rules (ENTER) and (EXIT) have $\rho, \eta \in \{a, *\}$ as side condition. The omitted subscripts on ambients are meant to remain unchanged by the reductions.

$$\begin{array}{lrcl}
(\textsc{enter}) & a^{\mathsf{k}}[\, \mathbf{in}\; b\,.\,P \mid Q\,]_\rho \mid b[\, \text{▬}^{\mathsf{k}}_\eta \mid R\,] & \searrow & \text{▬}^{\mathsf{k}}_\rho \mid b[\, a^{\mathsf{k}}[\, P \mid Q\,]_\eta \mid R\,] \\
(\textsc{exit}) & \text{▬}^{\mathsf{k}}_\eta \mid b[\, P \mid a^{\mathsf{k}}[\, \mathbf{out}\; b\,.\,Q \mid R\,]_\rho\,] & \searrow & a^{\mathsf{k}}[\, Q \mid R\,]_\eta \mid b[\, P \mid \text{▬}^{\mathsf{k}}_\rho\,] \\
(\textsc{get}\mathrm{S}) & b^{\mathsf{h}+1}[\, \mathbf{put}.P \mid \text{▬}_\eta \mid Q\,] \mid a^{\mathsf{k}}[\, \mathbf{get}\, b_\eta.R \mid S\,] & \searrow & b^{\mathsf{h}}[\, P|Q\,] \mid a^{\mathsf{k}+1}[\, R \mid \text{▬}_\eta \mid S\,] \\
(\textsc{get}\mathrm{U}) & \mathbf{put}^{\downarrow}.\,P \mid \text{▬}_\eta \mid a^{\mathsf{k}}[\, \mathbf{get}^{\uparrow}{}_\eta\,.\,Q \mid R\,] & \searrow & P \mid a^{\mathsf{k}+1}[\, \text{▬}_\eta \mid Q \mid R\,] \\
(\textsc{spawn}) & \text{▬}^{\mathsf{k}}_\eta \mid \mathsf{k} \triangleright_\eta P & \searrow & P
\end{array}$$

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 move (as recorded in the tag associated with the ambient construct): this solves the problem we discussed at the beginning of Section 4. 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 Section 2. 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.

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

We have not defined this capability as a primitive of our calculus since it can be encoded using the new form of spawning as follows, for a fresh.

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

We will make substantial use of resource renaming in the examples of Section 4.2 and Section 4.3.

Notice that name rebinding is a dangerous capability, since it allows processes to give particular names to anonymous slots: $!\{{}^{y}/_{*}\}$. This enables possible malicious behaviour, like stealing all public resources by naming them with a private name. Such behaviour may be prevented by restricting the spawning $\mathsf{k}\triangleright_\eta$ to $\eta \in \mathcal{N} \cup \mathcal{V}$. 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: $!\{{}^{*}/_{y}\}$.

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 $\mathsf{k} \triangleright_\eta P \mid \blacksquare^{\mathsf{k}}_{\rho} \searrow P$ with the side condition $\eta = \rho$ *or* $\rho = *$. This behaviour, however, can be approximated closely enough by putting renaming processes in parallel with the spawning one, namely $\underbrace{\{\eta/*\} \mid \ldots \mid \{\eta/*\}}_{\mathsf{k}} \mid \mathsf{k} \rhd_\eta P$.

It is easy to check that the type system of Section 3 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 $\_\,_x$ can only be instantiated by a name. In particular, the typing rule for $\_\,_M$ is:

$$\frac{\Gamma \vdash M : Amb\langle -, -, - \rangle}{\Gamma \vdash \_\,_M : Proc\langle \mathsf{0}_{\mathcal{E}}, \chi \rangle}\ (\_\,_M)$$

The typing rule for $\_\,_\star$ is exactly that of $\_$, since they both denote anonymous slots.

The same properties we proved for the calculus of Section 2 hold for the variant with named slots discussed in this section.

Theorem 5 (Subject Reduction and Type Soundness) *For the processes and reduction relation of this section, we have:*

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

However, the presence of named slots allows much finer control on the dynamics of space allocation and acquisition. We illustrate two relevant applications in the next two subsections.

4.2 The cab trip revisited

Named slots allow us to avoid unwanted behaviours when encoding the cab trip example (see Figure 7). 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 or clients on that slot.

Fig. 7. Cab trip with named slots.

$$
\begin{aligned}
&\mathit{CALL}(\mathit{from}, \mathit{client}, \mathit{bye}) \triangleq\\
&\quad \mathit{call}^1[\textbf{out}\ \mathit{client}\,.\,\textbf{out}\ \mathit{from}\,.\,\textbf{in}\ \mathit{cab}\,.\,\overline{\textbf{open}}\,.\\
&\qquad \textbf{in}\ \mathit{from}\,.(\mathbf{\_}_{\mathit{client}} \mid \textbf{open}\ \mathit{arrived}\,.\,\{{}^{\mathit{bye}}\!/\!_{\mathit{client}}\}\,.\,\textbf{open}\ \mathit{bye})]\\[6pt]
&\mathit{TRIP}(\mathit{from}, \mathit{to}, \mathit{client}) \triangleq\\
&\quad \mathit{trip}^0[\textbf{out}\ \mathit{client}\,.\,\overline{\textbf{open}}\,.\,\textbf{out}\ \mathit{from}\,.\,\textbf{in}\ \mathit{to}\,.\\
&\qquad \mathit{arrived}^0[\ \overline{\textbf{open}}\,.\,\mathit{done}^0[\ \textbf{in}\ \mathit{client}\,.\,\overline{\textbf{open}}\ ]\ ]]\\[6pt]
&\mathit{CLIENT}(\mathit{from}, \mathit{to}) \triangleq\\
&\quad (\boldsymbol{\nu}\mathit{client} : W_1, \mathit{bye} : W_1)\,(\mathit{client}^1[\mathit{CALL}(\mathit{from}, \mathit{client}, \mathit{bye})\\
&\qquad \mid \textbf{in}\ \mathit{cab}\,.\,\mathit{TRIP}(\mathit{from}, \mathit{to}, \mathit{client})\\
&\qquad \mid \textbf{open}\ \mathit{done}\,.\,\textbf{out}\ \mathit{cab}\,.\\
&\qquad\quad 1 \rhd_{\mathit{call}} \mathit{bye}^1[\ \textbf{out}\ \mathit{client}\,.\,\textbf{in}\ \mathit{cab}\,.\,\overline{\textbf{open}}\,.\,\textbf{out}\ \mathit{to}\,.\,\mathbf{\_}_{\mathit{call}}\ ]] \mid \{{}^{*}\!/\!_{\mathit{client}}\})\\[6pt]
&\mathit{CAB} \quad\ \triangleq \mathit{cab}^1[\ \mathbf{\_}_{\mathit{call}} \mid !(\textbf{open}\ \mathit{call}\,.\,\textbf{open}\ \mathit{trip})\ ] \mid \{{}^{*}\!/\!_{\mathit{cab}}\}\\
&\mathit{SITE}(i) \triangleq \mathit{site}_i[\ \mathit{CLIENT}(\mathit{site}_i, \mathit{site}_j) \mid \mathit{CLIENT}(\mathit{site}_i, \mathit{site}_l) \mid \cdots \mid \mathbf{\_} \mid \mathbf{\_} \mid \cdots\ ]\\
&\mathit{CITY} \quad \triangleq \mathit{city}[\ \mathit{CAB} \mid \mathit{CAB} \mid \cdots \mid \mathit{SITE}(1) \mid \cdots \mid \mathit{SITE}(n) \mid \mathbf{\_} \mid \mathbf{\_} \mid \cdots\ ]
\end{aligned}
$$

- 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 $W_1 = \mathit{Amb}\langle[1,1], 0_{\mathcal{E}}\rangle$ and $W_0 = \mathit{Amb}\langle[0,0], 0_{\mathcal{E}}\rangle$. As in Subsection 3.3 the typing environment contains $\mathit{call} : W_1$, $\mathit{cab} : W_1$, $\mathit{trip} : W_0$, $\mathit{done} : W_0$. In the new encoding, instead, 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*. Using the labelled transition system given in Section 5 we proved 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 enters the cab the client just left. These properties can be formalised in a natural way. For the first one, for instance, we showed the full

Fig. 8. A travel agency.

$$\mathit{THE\ AGENCY} \triangleq$$

$$ag^{5}[\ \text{▬}^{2}_{cl} \mid \text{▬}_{req} \mid \text{▬}_{tkt}\ \mid$$

$$desk^{1}[\ \text{▬}_{req} \mid !(\mathbf{open}\ req.\,1 \triangleright_{fortkt}\ .\,tkt^{1}[\ \mathbf{out}\ desk.\,\mathbf{in}\ cl.\,\mathit{CONT}\,]\!|\ \{{}^{req}/_{tkt}\})]\ ]$$

where $\mathit{CONT} \triangleq (\overline{\mathbf{open}}.\,\mathbf{out}\ ag.\,\mathbf{in}\ plane.\,rdy^{0}[\ \mathbf{out}\ cl\,] \mid \text{▬}_{getoff} \mid \mathbf{open}\ getoff)$

$$\mathit{THE\ CLIENT} \triangleq$$

$$cl^{1}[\ \mathbf{in}\ ag.\,req^{1}[\ \mathbf{out}\ cl.\,\mathbf{in}\ desk.\,\overline{\mathbf{open}}.\,\text{▬}_{fortkt}\,]\!|\ \{{}^{tkt}/_{req}\} \mid \mathbf{open}\ tkt\,]$$

$$\mathit{THE\ AIRCRAFT}\ \triangleq$$

$$plane^{4}[\ \text{▬}^{2}_{cl} \mid \mathbf{open}\ rdy.\,\mathbf{open}\ rdy.\,\mathit{TRIP}.(\mathit{GETOFF} \mid \mathit{GETOFF})\,]$$

where $\mathit{GETOFF} \triangleq getoff^{1}[\ \mathbf{in}\ \mathrm{cl}.\,\overline{\mathbf{open}}.\,\mathbf{out}\ plane \mid \text{▬}\,]$ and TRIP is a path modelling the route of the aircraft

bisimilarity between the following processes:

$$(\boldsymbol{\nu} client, site_1, cab)city\,[cab[\ \mathbf{in}\ site_1\,.(\text{▬}_{client} \mid \cdots) \mid \cdots\ ]$$
$$\mid site_1[\ client[\ \mathbf{in}\ cab \cdots \mid \cdots\ ] \mid \cdots\ ] \mid \cdots]$$

and

$$(\boldsymbol{\nu} client, site_1, cab)(city[\ site_1[\ cab[\ client[\ \cdots\ ]\ ] \mid \cdots\ ] \mid \cdots\ ])$$

where $\cdots$ stand for various processes easily recoverable by looking at the given encoding.

4.3 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 ▬_{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 system are defined in Figure 8.

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.

5 Behavioural Semantics for BoCa

Our choice of behavioural semantics for BoCa is based on rather general notion of contextual equality defined in terms of *reduction barbed congruence*, a slight variant of Milner and Sangiorgi's *barbed congruence* [21], first studied by Honda and Yoshida in [16]. It is defined in terms of reduction and observation. 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) [11] this is defined as follows:

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

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 [18], 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 $a \notin \{\tilde{m}\}$):

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

$$(3) \quad P \downarrow_a^{slt} \quad \triangleq \quad P \equiv (\boldsymbol{\nu}\tilde{m})(a[\, \blacksquare \mid Q \,] \mid R)$$

$$(4) \quad P \downarrow_a^{put} \quad \triangleq \quad P \equiv (\boldsymbol{\nu}\tilde{m})(a[\, \mathbf{put}.\, P' \mid \blacksquare \mid Q \,] \mid R)$$

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 equivalence is standard in typed calculi, in that we require closure (only) by well-formed contexts. More precisely, we say that a relation $\mathcal{R}$ is

- *contextual* if $P\mathcal{R}Q$ implies that for all contexts $\mathcal{C}[\cdot]$ (i) $\mathcal{C}[P]$ is well-formed iff so is $\mathcal{C}[Q]$ and (ii) $\mathcal{C}[P]\mathcal{R}\mathcal{C}[Q]$ for all $\mathcal{C}[\cdot]$ such that $\mathcal{C}[P]$ is well-formed;
- *reduction closed* if $P\mathcal{R}Q$ and $P \searrow P'$ imply the existence of some Q' such that $Q \searrow_* Q'$ and $P'\mathcal{R}Q'$;
- *barb preserving* if $P\mathcal{R}Q$ and $P\downarrow_a$ imply $Q\Downarrow_a$, i.e. $Q \searrow_*\downarrow_a$.

Definition 6 (Reduction Barbed Congruence) *Reduction barbed congruence, noted $\cong$, is the largest equivalence relation that is contextual, and that, when restricted to closed processes, is reduction closed and barb preserving.*

The import of the process weight in the relation of behavioural equivalence is captured directly by the well-formedness requirement in Definition 6. 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, k and 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.

We show that the definition of equivalence is robust, as it is independent from the choice of the observation predicate. Let $\cong_i$ be the equivalence relation resulting from Definition 6 and from choosing the notion of observation as in (i) above (with $i \in [1..4]$).

Proposition 7 (Independence from barbs) $\cong_i = \cong_j$ *for all* $i, j \in [1..4]$.

PROOF. Since the relations differ only on the choice of barb, Proposition 7 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 $\mathcal{C}_{2,3}[\cdot] = [\cdot] \mid \textbf{open}\ a.b^1[\ \blacksquare\ ]$, and note that for all P such that b is fresh in P one has $P \Downarrow_a^{opn}$ if and only if $\mathcal{C}_{2,3}[P] \Downarrow_b^{slt}$. To show the remaining equivalences we use the following contexts:

$$\begin{aligned}
\mathcal{C}_{1,2}[\cdot] &= [\cdot] \mid b^0[\ \textbf{in}\ a\,.\,\textbf{out}\ a\,.\,\overline{\textbf{open}}\ ] \\
\mathcal{C}_{3,4}[\cdot] &= [\cdot] \mid b^1[\ \textbf{in}\ a\,.\,\textbf{put} \mid \blacksquare\ ] \\
\mathcal{C}_{4,1}[\cdot] &= [\cdot] \mid \textbf{get}^{\downarrow}a\,.\,b^0[\ ]
\end{aligned}$$

Fig. 9. Labels, concretions and outcomes

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

5.1 A LTS for BoCa

The universal quantification on contexts makes the definition of equivalence not effective for proving term congruences. For this reason we give a labelled transition system for the named calculus and sketch a proof of adequacy of the resulting notion of labelled bisimilarity with respect to the relation of equivalence given in Definition 6.

As we shall see in Subsection 5.2, a number of algebraic laws can be proved based on the resulting co-inductive characterization of reduction barbed congruence.

Notice that a LTS can readily be obtained for the calculus of Section 2 by simply erasing all the occurrences of η and ρ 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 [26,29,27]. As usual for ambient calculi [11,18,20,5], the labelled transitions have the form $P \xrightarrow{\alpha} O$, where P is a well-formed term, and

- the *label* α 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.

Figure 9 defines labels and concretions. In $(\boldsymbol{\nu}\tilde{p})\langle\!\langle P \rangle\!\rangle Q$ the process P represents the moving ambient and the process Q represents the remaining system not

Fig. 10. Commitments: Visible transitions

(Cap)
$$\frac{M \text{ capability}}{M.P \xrightarrow{M} P}$$

(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)
$$\frac{}{P \xrightarrow{\rule[0.3ex]{0.6em}{0.25em}^{0}_{\star}} P}$$

(Slot-1)
$$\frac{}{\rule[0.3ex]{0.6em}{0.25em}_{\eta} \xrightarrow{\rule[0.3ex]{0.6em}{0.25em}^{1}_{\eta}} \mathbf{0}}$$

(Slot-par)
$$\frac{P \xrightarrow{\rule[0.3ex]{0.6em}{0.25em}^{\mathsf{k}}_{\eta}} P_1 \quad Q \xrightarrow{\rule[0.3ex]{0.6em}{0.25em}^{1}_{\eta}} Q_1 \quad (\mathsf{k} \geq 1)}{P \mid Q \xrightarrow{\rule[0.3ex]{0.6em}{0.25em}^{\mathsf{k}+1}_{\eta}} P_1 \mid Q_1}$$

(Get)
$$\frac{M \in \{\mathbf{get}\, a_\eta, \mathbf{get}^{\uparrow}{}_{\eta}\}}{M\,.\,P \xrightarrow{M} \rule[0.3ex]{0.6em}{0.25em}_{\eta} \mid P}$$

(put)
$$\frac{M \in \{\mathbf{put}, \mathbf{put}^{\downarrow}\}\ P \xrightarrow{M} P'\ Q \xrightarrow{\rule[0.3ex]{0.6em}{0.25em}^{1}_{\eta}} Q'}{P \mid Q \xrightarrow{M\ \eta} P' \mid Q'}$$

(slot-inc)
$$\frac{M \in \{\mathbf{get}\, a_\eta, \mathbf{get}^{\uparrow}{}_{\eta}\} \quad P \xrightarrow{M} P'}{b^{\mathsf{k}}[\,P\,] \xrightarrow{*[\,M\,]} b^{\mathsf{k}+1}[\,P'\,]}$$

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

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

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

(In-Out)
$$\frac{P \xrightarrow{M} P' \quad M \in \{\mathbf{in}\ a, \mathbf{out}\ a\} \quad \eta, \rho \in \{b, *\}}{b^{\mathsf{k}}[\,P\,]_{\rho} \xrightarrow{\eta^{\mathsf{k}}[\,M\,]} (\boldsymbol{\nu})\boldsymbol{\langle} b^{\mathsf{k}}[\,P'\,]_{\eta} \boldsymbol{\rangle} \rule[0.3ex]{0.6em}{0.25em}^{\mathsf{k}}_{\rho}}$$

(Co-In)
$$\frac{P \xrightarrow{\rule[0.3ex]{0.6em}{0.25em}^{\mathsf{k}}_{\eta}} P'}{a^{\mathsf{h}}[\,P\,] \xrightarrow{a[\,\rule[0.3ex]{0.6em}{0.25em}^{\mathsf{k}}_{\eta}\,]} (\boldsymbol{\nu})\boldsymbol{\langle} P' \boldsymbol{\rangle}\mathbf{0}}$$

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

(Co-open)
$$\frac{P \xrightarrow{\overline{\mathbf{open}}} P'}{a[\,P\,] \xrightarrow{a[\,\overline{\mathbf{open}}\,]} P'}$$

Fig. 11. Commitments: τ transitions and structural transitions

(τ-Enter)

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

(τ-Exit)

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

(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)

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

(τ-Amb)

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

(τ-spawn)

$$\frac{P\xrightarrow{\mathsf{k}\rhd\eta}P_1 \quad Q\xrightarrow{\mathbf{-}^{\mathsf{k}}_{\eta}}Q_1}{P\mid Q\xrightarrow{\tau}P_1\mid Q_1}$$

(Repl)

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

(Par)

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

(Res)

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

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.

Fig. 12. Commitments: Higher-Order transitions

(HO Output)

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

(HO In/Co-In)

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

(HO Out)

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

Figures 10 and 11 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 $\mathrm{fn}(R) \cap \{\tilde{p}\} = \emptyset$.

The transitions are similar to the transitions defined for [18,5] 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. Distinctive of 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. Finally, rules (slot-inc) and (slot-dec) define the behavior of the put and get primitives.

We can show that the labelled transition semantics coincides with the reduction semantics, in the sense of the following theorem.

Theorem 8 *If $P \xrightarrow{\tau} Q$, then $P \searrow Q$. Conversely, if $P \searrow Q$, then $P \xrightarrow{\tau} Q'$*

for some $Q' \equiv Q$.

PROOF. The proof is entirely standard and we only sketch it.

We first need to extend the definition of structural congruence to concretions. That can be accomplished as follows:

- $(\boldsymbol{\nu}\tilde{p})\langle P\rangle Q \equiv (\boldsymbol{\nu}\tilde{p})\langle P'\rangle Q'$ if $P \equiv P'$ and $Q \equiv Q'$
- $(\boldsymbol{\nu}\tilde{p})\langle M\rangle P \equiv (\boldsymbol{\nu}\tilde{p})\langle M\rangle P'$ if $P \equiv P'$.

Then we establish a number of elementary results describing the structure of processes and outcomes involved in the labelled transitions. We give an illustrative case below:

If $P \xrightarrow{\mathbf{in}\ m} P'$, then there exist names $\tilde{p}$, with $\{m\} \cap \{\tilde{p}\} = \emptyset$, and processes P_1, P_2 such that $P \equiv (\boldsymbol{\nu}\tilde{p})(\mathbf{in}\ m.P_1 \mid P_2)$ and $P' \equiv (\boldsymbol{\nu}\tilde{p})(P_1 \mid P_2)$.

All these results follow as expected by rule induction. A further result relates labelled transitions and structural congruence.

If $P \xrightarrow{\alpha} O$ and $P \equiv Q$, then there exists O' such that $Q \xrightarrow{\alpha} O'$ and $O \equiv O'$.

This is proved by induction on the derivation of $P \equiv Q$. Then the proof of the theorem derives routinely by transition induction and a case analysis on the last rule applied in the derivation.

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

Proposition 9 $P{\downarrow_a}$ *if and only if* $P \xrightarrow{a[\ \overline{\mathbf{in}}\ ]} (\boldsymbol{\nu}\tilde{p})\langle Q\rangle R$ *for some* $\tilde{p}, Q, R$.

Following [20,5], in order to provide a characterization of reduction 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 Figure 12. 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 Λ be the set of all labels including the first-order labels of Figure 9 as well as the higher-order labels determined by the transitions in Figure 12. We denote with λ any label in the set Λ. As usual, we focus on weak bisimilarities based on weak transitions, and use the following notation:

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

Definition 10 (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 $Q \overset{\hat{\lambda}}{\Longrightarrow} Q'$ 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 11 (Full bisimilarity) *Two processes P and Q are full bisimilar, $P \approx_{\mathrm{c}} Q$, if $P\sigma \approx Q\sigma$ for every closing substitution σ.*

Note that the definition of bisimilarity only tests transitions from processes to processes. As expected the full bisimilarity is a congruence

Theorem 12 *$P \approx_{\mathrm{c}} Q$, is a congruence.*

PROOF. The proof that $\approx_{\mathrm{c}}$ is an equivalence relation is standard. It is also easy to show that $\approx_{\mathrm{c}}$ is preserved by input prefixes. For the remaining constructs we can safely restrict to closed processes in the language, and prove that $\approx$ is preserved by all contexts. We rely on the technique of [20,5], treating all the constructs but replication simultaneously, as follows. Specifically, we let $\mathcal{S}$ be the symmetric relation that contains $\approx$ and is closed by prefix, parallel composition, restriction and ambient, i.e.:

- $\approx \subseteq \mathcal{S}$
- $P \ \mathcal{S} \ Q$ implies $\pi.P \ \mathcal{S} \ \pi.Q$
- $P \ \mathcal{S} \ Q$ implies $P \mid R \ \mathcal{S} \ Q \mid R$ and $R \mid P \ \mathcal{S} \ R \mid Q$ for all processes R
- $P \ \mathcal{S} \ Q$ implies $n^{\mathsf{k}}[P] \ \mathcal{S} \ n^{\mathsf{k}}[Q]$ and $(\boldsymbol{\nu} n)P \ \mathcal{S} \ (\boldsymbol{\nu} n)Q$ for all names n (where $\mathsf{k} = w(P)$).

Then, we show that $\mathcal{S}$ is a bisimulation up to $\equiv$ [29] by induction on the formation of $\mathcal{S}$. The proof has the exact same structure as the corresponding proofs in [20,5].

Lastly we show that $\approx$ is preserved by replication using its transitivity and its closure under parallel composition.

Based on the previous result, we show that full bisimilarity is sound (but not complete) w.r.t. reduction barbed congruence.

Theorem 13 (Soundness of full bisimilarity) *If $P \approx_{\mathrm{c}} Q$, then $P \cong Q$.*

PROOF. Rule (WEIGHT) distinguishes processes of different weights. Then it is enough to show that $\approx_{\mathrm{c}}$ is reduction closed and barb preserving, up to $\equiv$. Assume that $P \approx_{\mathrm{c}} Q$ and $P \searrow P'$. By Theorem 8 $P \xrightarrow{\tau}\equiv P'$. Since $P \approx_{\mathrm{c}} Q$, there exits Q' such that $Q \searrow_* Q'$ and $P' \equiv\approx_{\mathrm{c}}\equiv Q'$. Now assume $P \approx_{\mathrm{c}} Q$. If $P\downarrow_a$ then, by Proposition 9, and rule (HO Co-In), $P \xrightarrow{a[\,\overline{\mathbf{in}}\,]R} S$ for some R, S. Since $P \approx_{\mathrm{c}} Q$ we know that $Q \overset{a[\,\overline{\mathbf{in}}\,]R}{\Longrightarrow} S'$ for some $S' \approx_{\mathrm{c}} S$, from which $Q\Downarrow_a$, as desired.

The failure of completeness is a consequence of the fact that contexts are insensitive to repeated entering and exiting. This phenomena is called *stuttering* in [28] and it is typical of movements which do not consume co-capabilities, as happen in our calculus. Let us recall an example given in [34], which shows the problem also for BoCa. No context can distinguish between the processes:

$$(\boldsymbol{\nu} n)(\mathbf{in}\ a\,.\,\mathbf{out}\ a\,.\,\mathbf{in}\ a\,.\,n[\,\overline{\mathbf{open}}\,] \mid !\mathbf{open}\ n\,.\,\mathbf{out}\ a\,.\,\mathbf{in}\ a\,.\,n[\,\overline{\mathbf{open}}\,])$$
$$(\boldsymbol{\nu} n)(\mathbf{in}\ a\,.\,n[\,\overline{\mathbf{open}}\,] \mid !\mathbf{open}\ n\,.\,\mathbf{out}\ a\,.\,\mathbf{in}\ a\,.\,n[\,\overline{\mathbf{open}}\,])$$

which are not fully bisimilar.

5.2 Algebraic laws

We highlight some algebraic laws that can be proved using our labelled transition system.

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

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

Indeed, all these processes are inert, hence behaviorally equivalent to the null process. Given that they have non-null weight, however, they are not congruent to the **0** process, but rather to what may be construed as a new process construct 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 if we protect the slots inside a private ambient. Specifically, we have:

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

$$(A_3) \qquad (\boldsymbol{\nu} a)(\_^{\mathsf{k}}_{a} \mid \mathsf{k}\ \rhd_{\,a} P) \;\cong\; P$$

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 [18].

For the calculus of Section 2, the mobility laws concerning **out** moves are weaker than the corresponding laws in [18], as our slots act uniformly as co-capabilities for ambient movements and process spawning. We have instead the following law for **in** moves:

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

For the calculus of Section 4 we get a law similar to that of [18] for **out** moves only for ambients with non-null weight: [3]

$$(A_6) \qquad (\boldsymbol{\nu} a, b)(\_^{\mathsf{k}}_{b} \mid a^{\mathsf{k}}[\,b^{\mathsf{k}}[\,\mathbf{out}\ a\,.\,P\,]\,]) \;\cong\; (\boldsymbol{\nu} a, b)(b^{\mathsf{k}}[\,P\,] \mid a^{\mathsf{k}}[\,\_^{\mathsf{k}}_{b}\,])$$

As a further remark, we note that there are congruences, like (A_4), between typable and untypable terms for a fixed environment. In fact, by assuming

[3] Ambients with non-null weight may be characterized by means of a typing system in which one requires that all ambient types have a strictly positive lower bound.

$w(P) = \mathsf{h}$, $b \notin \Gamma$, 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.

6 Related Work

Our approach is related to the work on *Controlled Mobile Ambients* (CMA) [31] and on *Finite Control Mobile Ambients* [7]. 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 consumption of resources which do not seem possible for CMA. Recently, [30] introduced the "controlled π-calculus" Cπ, where name creation ν represents the capability of allocating a resource to a fragment of code, and is matched up by a corresponding "garbage-collection" capability $\urcorner$ (delete). Static typing techniques are then built on top of such operators in order to express and guarantee resource bounds.

As to [7], 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.

Other relevant references to related work in this context include [14], which introduces a notion of resource type representing an abstract unit of space, and uses a linear type system to guarantee linear space consumption; [8] where quantitative bounds on time usage are enforced using a typed assembly language; and [17], which puts forward a general formulation of resource usage analysis.

7 Conclusion and Future Work

We have presented an ambient-like calculus centred around an explicit primitive representing a resource unit: the space “slot” ▬. 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 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 ▬ 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. We have given a corresponding type inference system and proved the inference problem decidable. The inference algorithm has been implemented in Prolog [19].

BoCa relies on (prefix) replication to express non-terminating computations. Using recursion instead of replication would, at least in principle, allow non-trivial recursive processes of weight greater than 0. This however requires the systematic use of spawning to “freeze” weight. In fact, if the weight of $\mathsf{rec}X.P$ is k, so are those of P and X. Therefore, if X occurs in P outside the scope of a spawning constructs, then no other component can contribute a positive weight to P. It follows that the only value k can have is zero. The situation is different if X is unspawned as in, e.g., $\mathsf{rec}X.((1 \triangleright X) \mid$ ▬$)$. The term here has weight 1, and no other weight would do. The interaction between recursion and spawning requires careful consideration. For instance, recursion would allow to insert space at any depth in a term, which is not immediately aligned with our intuition nor of course obviously simulated with replication. For this reason we have chosen to focus here exclusively on replication and spawning, which represent the more elementary and self-explanatory paradigm of code copying and activation. The fine comparison between recursion and replication appears therefore to be a non-trivial question, which we leave to future work.

Plans for future include further work in several directions. Extending the type inference as outlined at the end of Subsection 3.4 we will be able to dynamically type-check processes obtaining in this way a *proof carrying code* [22] approach to space control. 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 [15].

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 (which has already been considered in [31]) sees an ambient as a packaging abstraction whose weight may have a different interpretation from that of content. 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.

Finally, 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 comments from the anonymous referees of ASIAN 2003. We are also most grateful to the referees of the journal submission for the valuable criticism and constructive feedback on the presentation and the technical content.

References

[1] Fahiem Bacchus and Toby Walsh, editors. *Theory and Applications of Satisfiability Testing*, volume 3569 of *Lecture Notes in Computer Science*. Springer-Verlag, 2005.

[2] Franco Barbanera, Michele Bugliesi, Mariangiola Dezani-Ciancaglini, and Vladimiro Sassone. A Calculus of Bounded Capacities. In Vijay A. Saraswat, editor, *ASIAN'03*, volume 2896 of *Lecture Notes in Computer Science*, pages 205–223. Springer-Verlag, 2003.

[3] Franco Barbanera, Mariangiola Dezani-Ciancaglini, Ivano Salvo, and Vladimiro Sassone. A Type Inference Algorithm for Secure Ambients. In Marina Lenisa

and Marino Miculan, editors, *TOSCA'01*, volume 62 of *ENTCS*. Elsevier Science, 2002.

[4] Michele Bugliesi and Giuseppe Castagna. Secure Safe Ambients. In Chris Hankin and Dave Schmidt, editors, *POPL'01*, pages 222–235. ACM Press, 2001.

[5] Michele Bugliesi, Silvia Crafa, Massimo Merro, and Vladimiro Sassone. Communication and Mobility Control in Boxed Ambients. *Information and Computation*, 202(1):39–86, 2005.

[6] Luca Cardelli and Andrew D. Gordon. Mobile Ambients. *Theoretical Computer Science*, 240(1):177–213, 2000. Special Issue on Coordination, D. Le Métayer Editor.

[7] Witold Charatonik, Andrew D. Gordon, and Jean-Marc Talbot. Finite-control Mobile Ambients. In Daniel Le Métayer, editor, *ESOP'02*, volume 2305 of *Lecture Notes in Computer Science*, pages 295–313. Springer-Verlag, 2002.

[8] Karl Crary and Stephanie Weirich. Resource Bound Certification. In Mark Wegman and Thomas Reps, editors, *POPL'00*, pages 184–198. ACM Press, 2000.

[9] Rocco De Nicola, Gianluigi Ferrari, Rosario Pugliese, and Betti Venneri. Types for Access Control. *Theoretical Computer Science*, 240(1):215–254, 2000.

[10] Jens Chr. Godskesen, Thomas Hildebrandt, and Vladimiro Sassone. A Calculus of Mobile Resources. In Lubos Brim, Petr Jancar, Mojmir Kretinsky, and Antonin Kucera, editors, *CONCUR'02*, volume 2421 of *Lecture Notes in Computer Science*, pages 272–287. Springer-Verlag, 2002.

[11] Andrew D. Gordon and Luca Cardelli. Equational Properties of Mobile Ambients. *Mathematical Structures in Computer Science*, 13(3):371–408, 2003.

[12] Matthew Hennessy, Massimo Merro, and Julian Rathke. Towards a Behavioural Theory of Access and Mobility Control in Distributed Systems. *Theoretical Computer Science*, 322(3):615–669, 2004.

[13] Matthew Hennessy and James Riely. Resource Access Control in Systems of Mobile Agents. *Information and Computation*, 173:82–120, 2002.

[14] Martin Hofmann. The Strength of non Size-increasing Computation. In John Launchbury and John C. Mitchell, editors, *POPL'02*, pages 260–269. ACM Press, 2002.

[15] Martin Hofmann and Steffen Jost. Static Prediction of Heap Space Usage for First-Order Functional Programs. In Greg Morrisett and Alex Aiken, editors, *POPL'03*, pages 185–197. ACM Press, 2003.

[16] Kohei Honda and Nobuko Yoshida. On Reduction-based Process Semantics. *Theoretical Computer Science*, 152(2):437–486, 1995.

[17] Atsushi Igarashi and Naoki Kobayashi. Resource Usage Analysis. In John Launchbury and John C. Mitchell, editors, *POPL'02*, pages 331–342. ACM Press, 2002.

[18] Francesca Levi and Davide Sangiorgi. Controlling Interference in Ambients. *Transactions on Programming Languages and Systems*, 25(1):1–69, 2003.

[19] Ugo de' Liguoro and Giuseppe Falzetta. Effect System for BoCa: a Prolog Implementation. Granted by IST-2001-33477 (DART) Project, http://www.di.unito.it/ deligu/papers/dLF05.pdf, 2005.

[20] Massimo Merro and Matthew Hennessy. Bisimulation Congruences in Safe Ambients. In John Launchbury and John C. Mitchell, editors, *POPL'02*, pages 71–80. ACM Press, 2002.

[21] Robin Milner and Davide Sangiorgi. Barbed Bisimulation. In Werner Kuich, editor, *ICALP'92*, volume 623 of *Lecture Notes in Computer Science*, pages 685–695. Springer-Verlag, 1992.

[22] George C. Necula. Proof-Carrying Code. In Neil D. Jones, editor, *POPL'97*, pages 106–119. ACM Press, 1997.

[23] George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. Wiley, 1988.

[24] Benjamin Pierce and Davide Sangiorgi. Typing and Subtyping for Mobile Processes. *Mathematical Structures in Computer Science*, 6(5):409–454, 1996.

[25] William Pugh. The Omega Test: a Fast and Practical Integer Programming Algorithm for Dependence Analysis. *Supercomputing*, pages 4–13, 1991.

[26] Davide 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.

[27] Davide Sangiorgi. Bisimulation for Higher-Order Process Calculi. *Information and Computation*, 131(2):141–178, 1996.

[28] Davide Sangiorgi. Extensionality and Intensionality of the Ambient Logic. In Chris Hankin and Dave Schmidt, editors, *POPL'01*, pages 4–13. ACM Press, 2001.

[29] Davide Sangiorgi and Robin Milner. The Problem of "Weak Bisimulation up to". In Walter R. Cleaveland, editor, *CONCUR'92*, volume 630 of *Lecture Notes in Computer Science*, pages 32–46. Springer-Verlag, 1992.

[30] David Teller. Recollecting Resources in the π-calculus. In Jean-Jacques Lévy, Ernst W. Mayr, and John C. Mitchell, editors, *TCS'04*, pages 605–618. Kluwer, 2004.

[31] David Teller, Pascal Zimmer, and Daniel Hirschkoff. Using Ambients to Control Resources. In Lubos Brim, Petr Jancar, Mojmir Kretinsky, and Antonin Kucera, editors, *CONCUR'02*, volume 2421 of *Lecture Notes in Computer Science*, pages 288–303. Springer-Verlag, 2002.

[32] Joe 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 *Lecture Notes in Computer Science*, pages 913–925. Springer-Verlag, 2002.

[33] Nobuko Yoshida and Matthew Hennessy. Subtyping and Locality in Distributed Higher Order Mobile Processes (extended abstract). In Jos C.M. Baeten and Sjouke Mauw, editors, *CONCUR'99*, volume 1664 of *Lecture Notes in Computer Science*, pages 557–573. Springer-Verlag, 1999.

[34] Francesco Zappa Nardelli. *De la sémantique des Processus d'Ordre Supérieur*. PhD thesis, École Normale Supérieure, Paris, 2003.