Available online at www.sciencedirect.com

ScienceDirect

Theoretical Computer Science 366 (2006) 121 – 143

Theoretical Computer Science

www.elsevier.com/locate/tcs

A coalgebraic approach to the semantics of the ambient calculus

Daniel Hausmann, Till Mossakowski, Lutz Schröder*

Department of Computer Science, University of Bremen, Germany

Abstract

Recently, various process calculi have been introduced which are suited for the modelling of mobile computation and in particular the mobility of program code; a prominent example is the ambient calculus. Due to the complexity of the involved spatial reduction, there is—in contrast to the situation in standard process algebra—up to now no satisfying coalgebraic representation of a mobile process calculus. Here, we discuss a coalgebraic denotational semantics for the ambient calculus, viewed as a step towards a generic coalgebraic framework for modelling mobile systems. Crucial features of our modelling are a set of GSOS style transition rules for the ambient calculus, a hardwiring of the so-called hardening relation in the functorial signature, and a set-based treatment of hidden name sharing. The formal representation of this framework is cast in the algebraic–coalgebraic specification language CoCasl.

Keywords: Process algebra; Ambient calculus; Coalgebra; CoCasl

Coalgebra has in recent years gained importance as a framework for modelling reactive systems at an appropriate level of generality [20]. Here, coalgebra serves as a basis for the semantics of processes, giving rise to generic notions in particular of bisimilarity, coinduction, corecursion, and modal logic [18]. In analogy to the (largely algebraic and order theoretic) denotational semantics of programming languages and logics, it is desirable to find a coalgebraic denotational semantics for process calculi, which then profit from the above-mentioned generic semantic notions and results. Such a denotational semantics also adds clarity to the calculi themselves and facilitates the comparison and, possibly, unification of process calculi.

For (the finitely branching fragment of) the classical process calculus CCS, a coalgebraic semantics has been defined in [16]; further work in similar directions is found e.g. in [8,9]. Here, we present work on a coalgebraic denotational semantics for mobile process calculi, in particular the ambient calculus [3]. This poses rather more involved problems than in the case of classical calculi, since the spatial structure interacts with the dynamic structure of processes in a complex way.

The two crucial steps in the design of the coalgebraic model are (not necessarily in this order) the identification of a suitable signature functor that determines the underlying type of transition systems, and the presentation of the calculus in the form of transition rules that allow a corecursive definition of the process building operations. Our solution for the latter problem is modelled on labelled transition systems (LTS) for the ambient calculus defined in [6,12]. The hardening relation introduced in [6] serves to single out active top-level processes to be involved in spatial reductions. Additional labels, introduced in [12], lead to a system that comes closer to a coalgebraic format. Here, we concentrate on the pre-action and internal action part of the system of [12], which we slightly adapt to fit the GSOS style [24]. The present work constitutes a first step towards a coalgebraic formulation of the full system of [12], which captures also

* Corresponding author. Tel.: +49 4212184683; fax: +49 4212183054.
E-mail address: lschrode@tzi.de (L. Schröder).

0304-3975/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2006.07.006

stuttering invariance and the perfect firewall equation. The subsystem considered here is nevertheless of independent interest, as we show that it is equivalent to the LTS of [6].

The central role of the hardening relation is reflected in our design of a coalgebraic functorial signature for the ambient calculus, which contains a component that corresponds to non-deterministic splitting of processes. The final coalgebra for this functor serves as a semantic domain for the interpretation of ambient calculus processes; the transition system can then be translated into corecursive definitions of functions into this domain. The problem that arises from the necessity of sharing restricted names between the two components into which a process is split by the hardening relation is solved by means of an explicit closure under α-equivalence of hidden names; this avoids disclosing the shared names but still allows the desired interaction between the components. In this way, it becomes possible to treat name hiding in a purely set-theoretic framework.

One benefit of these results is that one can now model the ambient calculus in a set-based formal specification language such as the algebraic–coalgebraic language CoCasl [16]. This allows e.g. for an integration of the ambient calculus into the Bremen heterogeneous tool set Hets [14,15], including the proof support for CoCasl presently under development, like circular coinduction [7]. For illustrative purposes, we present excerpts from a CoCasl specification of the ambient calculus that implements our corecursive definitions.

The material is organized as follows. Some basic concepts of coalgebra are recalled in Section 1. An introduction to the ambient calculus is given in Section 2. Finally, our coalgebraic semantics for the ambient calculus is presented in Section 3, beginning with the transition rules in GSOS format (Section 3.1), continuing with the presentation of the signature functor for the ambient calculus and the corecursive equations for the process building operations (Section 3.2), and finishing with a discussion of the formal specification in CoCasl (Section 3.3). Most results are stated in the main text without proof; the corresponding proofs are gathered in Appendix C (Appendices A and B recall technical material from [6]).

1. Coalgebra

We briefly recall some basic notions of coalgebra relating to the modelling of reactive systems.

Definition 1. Let $T : \mathbf{Set} \to \mathbf{Set}$ be a functor (in this work, all functors will implicitly be set functors), referred to as the *signature functor*. A T*-coalgebra* (X, ξ) consists of a set X of *states* and an *transition map* $\xi : X \to TX$. A *morphism* $(X_1, \xi_1) \to (X_2, \xi_2)$ of T-coalgebras is a map $f : X_1 \to X_2$ such that $\xi_2 \circ f = Tf \circ \xi_1$. A T-coalgebra Z is called *final* if there exists, for each T-coalgebra X, a unique morphism $X \to Z$.

Intuitively, the transition map describes the successor states and observations of a state, organized in a data structure given by T. These data encode the observable *behaviour* of a system, and morphisms of coalgebras preserve this behaviour. Indistinguishability of behaviours is formally captured as follows.

Definition 2. Two states x, y in coalgebras (X, ξ), (Y, ζ), respectively, are called *behaviourally equivalent* if there exist T-coalgebra morphisms $f : X \to W, g : Y \to W$ such that $f(x) = g(y)$. A binary relation $R \subseteq X \times Y$ between T-coalgebras (X, ξ) and (Y, ζ) is called a T*-bisimulation* if there exists a T-coalgebra structure on R that makes the projection functions $\pi_1 : R \to X$ and $\pi_2 : R \to Y$ into coalgebra morphisms. Two states are called T*-bisimilar* if they are related by some T-bisimulation.

If T preserves weak pullbacks (as will be the case for the functor appearing in the modelling of the ambient calculus in Section 3.2), then T-bisimilarity coincides with behavioural equivalence. In particular, bisimilarity is equality in the final T-coalgebra, and hence serves as a tool for proving identities between processes.

A very simple definition principle for functions, called *coiteration* or coinductive definition, is induced directly by the definition of the final coalgebra: a function f from a set X into the final T-coalgebra (Z, ζ) can be defined by giving a T-coalgebra structure ξ on X and stating that $f : (X, \xi) \to (Z, \zeta)$ is a coalgebra morphism, i.e. by specifying the corecursive equation $Tf\xi = \zeta f$. This method is sufficient for the definition of the semantics of classical process algebras such as CCS [16]; e.g. if $\zeta_a(x)$ denotes the set of a-successors of x for an action a in the final coalgebra,

then the parallel operator | is defined by

$$\zeta_\tau(x|y) = \{(x'|y) \mid x' \in \zeta_\tau(x)\} \cup \{(x|y') \mid y' \in \zeta_\tau(y)\} \cup \{(x'|y') \mid x' \in \zeta_l(x), y' \in \zeta_{\bar{l}}(y), l \in \mathcal{L}\},$$
$$\zeta_l(x|y) = \{(x'|y) \mid x' \in \zeta_l(x)\} \cup \{(x|y') \mid y' \in \zeta_l(y)\} \quad (l \in \mathcal{L}),$$

where τ denotes the silent action and $\mathcal{L}$ is the set of labels. In the definition of our coalgebraic semantics of the ambient calculus, we will need a more complex form of corecursion to be explained in Section 3.

2. The ambient calculus

The ambient calculus [3] models mobile computing (i.e. in mobile computing devices, like laptops or mobile phones) as well as mobile computations (i.e. processes that move among devices, like applets). A central issue is the handling of administrative domains and their boundaries (e.g. protected by firewalls). The ambient calculus hence comprises agents, their ambients, and mobility of these ambients.

Space is understood to be hierarchical in the ambient calculus, and the hierarchical fragmentation of space is represented using the notion of ambient: An ambient is a single entity with a clear separation from its environment. It may contain processes or further ambients. Thus ambients may be nested or they may be residing in parallel on the same level. The dynamic change of the position of ambients in space over time is represented in the ambient calculus by several reduction rules which utilize so-called *capabilities*. The capabilities model the opportunity for processes to enter, leave, or open ambients.

The syntax of the ambient calculus is defined as follows: For a set $\mathcal{N}$ of names (m, n will range over names in the sequel), the set of processes for the ambient calculus $\mathcal{AC}$ is defined inductively as the least set which is closed under

- the nil process $\mathbf{0}$,
- parallel composition of processes $P|Q$,
- capability prefixing $M.P$, where $M \in \{\mathbf{in}\ n, \mathbf{out}\ n, \mathbf{open}\ n\}$,
- the ambient operator $n[P]$,
- name restriction $(\nu n)P$,
- replication $!P$.

The set of free names $fn(P)$ of an ambient calculus process P is, roughly speaking, the set of names that appear in the process either in ambient operators or in the prefixing of capabilities, minus the set of all names that appear in name restrictions. Restriction of a sequence of i names $\overrightarrow{p} = (p_1, \ldots, p_i)$ is denoted for any process P as $(\nu\overrightarrow{p})P = (\nu p_1) \ldots (\nu p_i)P$. The empty restriction is written as $(\nu)P$; furthermore, $(\nu\overrightarrow{p})(\nu\overrightarrow{q})P = (\nu\overrightarrow{p}, \overrightarrow{q})P$.

The semantics of the ambient calculus is defined by the *reduction relation* $\longrightarrow \subset \mathcal{AC} \times \mathcal{AC}$, which is the least relation that satisfies the rules displayed in Fig. 1, where $\equiv \subset \mathcal{AC} \times \mathcal{AC}$ denotes *structural congruence*. The latter is defined as the smallest congruence relation satisfying the rules of Fig. 2. The *transitive reflexive closure* of $\longrightarrow$ is denoted by $\overset{*}{\longrightarrow}$. Since it follows from the rules in Fig. 2 that $(\nu n)(\nu n)P \equiv (\nu n)P$, restrictions can up to structural congruence be regarded as referring to sets of names.

We recall some notions from [6,12] related to equivalence of processes.

Definition 3.

- An ambient calculus process P *exhibits* a name n (written $P \downarrow n$) if there are names $\overrightarrow{m}$ and processes P', P'' with $n \notin \overrightarrow{m}$ such that $P \equiv (\nu\overrightarrow{m})(n[P'] \mid P'')$.
- An ambient calculus process P *converges* to a name n ($P \Downarrow n$) if it can evolve to a process that exhibits n, i.e. if there exists P' such that $P \overset{*}{\longrightarrow} P'$ and $P' \downarrow n$.
- A *context* $\mathcal{C}()$ is an ambient calculus process with zero or more holes. For a process P, $\mathcal{C}(P)$ denotes the process obtained by filling each hole of the context $\mathcal{C}$ with P.
- Two ambient calculus processes P, Q are *contextually equivalent* ($P \simeq Q$) if they exhibit the same convergence behaviour when plugged into an arbitrary context $\mathcal{C}()$, i.e. if $\mathcal{C}(P) \Downarrow n \Leftrightarrow \mathcal{C}(Q) \Downarrow n$ for all contexts $\mathcal{C}$ and all names n.
- A binary relation ρ is *barb-preserving* if $P\rho Q$ and $P \downarrow n$ imply $Q \Downarrow n$.

$$m[\mathbf{in}\, n.P|Q]|n[R] \longrightarrow n[m[P|Q]|R] \qquad \mathbf{open}\, n.Q|n[R] \longrightarrow Q|R$$

$$n[m[\mathbf{out}n.P|Q]|R] \longrightarrow m[P|Q]|n[R] \qquad \frac{P \longrightarrow Q}{P|R \longrightarrow Q|R} \qquad \frac{P \longrightarrow Q}{n[P] \longrightarrow n[Q]}$$

$$\frac{P \longrightarrow Q}{(\nu n)P \longrightarrow (\nu n)Q} \qquad \frac{P \equiv P' \qquad P' \longrightarrow Q' \qquad Q' \equiv Q}{P \longrightarrow Q}$$

Fig. 1. The reduction relation of the ambient calculus.

$$P|\mathbf{0} \equiv P \qquad P|Q \equiv Q|P \qquad (\nu m)(\nu n)P \equiv (\nu n)(\nu m)P$$

$$(\nu n)\mathbf{0} \equiv \mathbf{0} \qquad (P|Q)|R \equiv P|(Q|R) \qquad (\nu n)(P|Q) \equiv P|(\nu n)Q \text{ if } n \notin fn(P)$$

$$!\mathbf{0} \equiv \mathbf{0} \qquad !P \equiv P|!P \qquad (\nu m)n[P] \equiv n[(\nu m)P] \text{ if } n \neq m$$

Fig. 2. The structural congruence of the ambient calculus.

3. A coalgebraic semantics for the ambient calculus

Following [3], we can equivalently define the reduction relation of the ambient calculus in terms of a LTS together with a hardening relation; the role of the latter is to single out active top-level processes, in order to prevent processes 'tagging along' in spatial reductions performed by parallel processes. Below, we present such a system in a somewhat modified form that closely follows [12]. This will allow us to embed the ambient calculus semantically into a coalgebraic framework; in particular, we give a suitable behaviour functor for the ambient calculus and define the process building operations of the ambient calculus as corecursive operations on the final coalgebra of this functor. The corecursive definitions will be formulated in mathematical notation; for selected examples, we will also present the formal specifications in CoCasl.

3.1. The labelled transition system

In order to allow for a coalgebraic specification of the semantics, we design a variant of a subsystem of the LTS given in [12], with the aim of obtaining a presentation that strictly adheres to the so-called GSOS format [24]. This format requires in particular that the conclusion of each inference rule has the application of exactly one process-building operation of the ambient calculus on the left-hand side of the transition, and that no process building operations appear in the premises. This facilitates the subsequent corecursive definition of the operations. The system of [12] deviates from this format in three respects:

- the premises of some of the rules contain the ambient operator;
- the system mixes processes with so-called concretions; and
- the rules for the so-called *env-actions* have single process variables on the left of transitions in their conclusions.

Our system as presented below makes adaptations w.r.t. the first two points: we slightly redesign the rules, and we replace transitions into concretions by a hardening relation in the style of [6]. Moreover, we give GSOS rules for the replication operator, which is treated only in restricted form in [12]. The third point is left for future research; at present, we have to content ourselves to just omit the env-actions altogether. In the remaining system, it is apparently no longer possible to capture the notion of reduction barbed congruence [12] by a suitable notion of bisimulation (for details, cf. Remark 27). However, the remaining system is nevertheless strong enough to capture all reductions of the ambient calculus, being in fact equivalent to the system of [6] (Theorem 14).

As in [6], a *concretion* is an expression of the form $(\nu \overrightarrow{p})\langle P\rangle Q$, where $P, Q \in \mathcal{AC}$ and $\overrightarrow{p} = \{p_1, \ldots, p_n\}$. The process P is called the *prime*, and Q is called the *residue*. The intuition is that a process, which may have many top-level

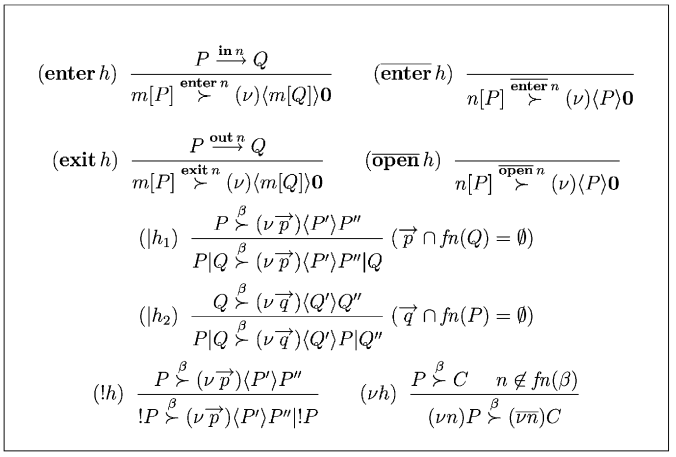

Fig. 3. The hardening relation.

processes, may harden to a concretion that singles out an active subprocess P, leaving behind the residue Q, where $\overrightarrow{p}$ is the set of private names shared by P and Q. This is necessary e.g. in order to ensure that only the left part of a process $m[\mathbf{in}\, n.P] \mid Q$ moves when the capability $\mathbf{in}\, n$ is exercised.

In order to keep track of the structure of an ambient calculus process over several inference steps, we use a labelled version of the hardening relation. So-called *intermediate capabilities* are used to store the information that a process is of a specific shape, and this information then appears as the premise of an inference rule which is used to derive a transition of the process. Thus, the LTS itself works entirely on processes, rather than also on concretions as in [12].

The hardening relation and the LTS are defined by mutual recursion; i.e. LTS relations may appear as assumptions in the rules for the hardening relation, and vice versa. Hardenings are of the form $P \overset{\beta}{\succ} C$, where P is a process, C is a concretion, and β is an element of the set $HAct = \{\mathbf{enter}, \overline{\mathbf{enter}}, \mathbf{exit}, \overline{\mathbf{open}}\} \times \mathcal{N}$ of *hardening labels*. The rules are given in Fig. 3, where $fn(\beta)$ denotes the set containing the single ambient name occurring in β and where for $C = (\nu \overrightarrow{p})\langle P'\rangle P''$, $(\overline{\nu n})C$ is defined as

$$(\overline{\nu n})C = \begin{cases} (\nu \overrightarrow{p})\langle P'\rangle (\nu n) P'' & \text{if } n \notin fn(P'), \\ (\nu \overrightarrow{p})\langle (\nu n) P'\rangle P'' & \text{if } n \notin fn(P''), \\ (\nu n, \overrightarrow{p})\langle P'\rangle P'' & \text{otherwise.} \end{cases}$$

Here, we assume $n \notin \overrightarrow{p}$; this can always be ensured by α-renaming.

Transitions are of the form $P \xrightarrow{\alpha} Q$, where P and Q are processes, and α is an element of the set $Act = \{\tau\} \cup \{\mathbf{in}, \mathbf{out}, \mathbf{open}\} \times \mathcal{N}$ of *transition labels*. Non-τ labels are called *capabilities*, ranged over by the metavariable M. The transition rules are displayed in Figs. 4 and 5, where

$$fn(\alpha) = \begin{cases} \emptyset & \text{if } \alpha = \tau, \\ \{n\} & \text{if } \alpha = M \text{ for the single ambient name } n \text{ in } M. \end{cases}$$

Note that while we do not impose structural congruence on terms for purposes of the hardening and transition relations, we continue to regard bound names as given only up to α-equivalence. Thus the hardening rule for parallel composition and the transition rules for opening and entering ambients can always be made applicable by α-renaming. Concerning the transition rule for exiting ambients, one can show that the premise implies $n \in fn(P)$, in particular $n \notin \overrightarrow{p}$.

Example 4. To illustrate the interplay of hardening and transitions, we demonstrate how the entering capability is exercised in a process of the form $(\nu m)(m[\mathbf{in}\, n.P]) \mid n[Q]$:

Since $\mathbf{in}\, n.P \xrightarrow{\mathbf{in}\, n} P$, it follows by rule $(\mathbf{enter}\, h)$ that $m[\mathbf{in}\, n.P] \overset{\mathbf{enter}\, n}{\succ} (\nu)\langle m[P]\rangle \mathbf{0}$. Since $m \notin fn(\mathbf{enter}\, n)$, we obtain $(\nu m)m[\mathbf{in}\, n.P] \overset{\mathbf{enter}\, n}{\succ} (\overline{\nu m})(\nu)\langle m[P]\rangle \mathbf{0}$ by rule (νh). By the definition of restriction of names on concretions,

$$(pre)\ \frac{}{M.P \xrightarrow{M} P} \qquad (|_1)\ \frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q} \qquad (|_2)\ \frac{P \xrightarrow{\alpha} P'}{Q|P \xrightarrow{\alpha} Q|P'}$$

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

$$(amb)\ \frac{P \xrightarrow{\tau} Q}{n[P] \xrightarrow{\tau} n[Q]} \qquad (\mathbf{exit})\ \frac{P \overset{\mathbf{exit}\, n}{\succ} (\nu \overrightarrow{p})\langle P'\rangle P''}{n[P] \xrightarrow{\tau} (\nu \overrightarrow{p})(n[P''] \mid P')}$$

$$(\mathbf{open}_1)\ \frac{P \xrightarrow{\mathbf{open}\, n} P' \quad Q \overset{\overline{\mathbf{open}}\, n}{\succ} (\nu \overrightarrow{q})\langle Q'\rangle Q''}{P|Q \xrightarrow{\tau} (\nu \overrightarrow{q})(P'|Q'|Q'')}\ (\overrightarrow{q} \cap fn(P) = \emptyset)$$

$$(\mathbf{open}_2)\ \frac{Q \xrightarrow{\mathbf{open}\, n} Q' \quad P \overset{\overline{\mathbf{open}}\, n}{\succ} (\nu \overrightarrow{p})\langle P'\rangle P''}{P|Q \xrightarrow{\tau} (\nu \overrightarrow{p})(P'|P''|Q')}\ (\overrightarrow{p} \cap fn(Q) = \emptyset)$$

$$(\mathbf{enter}_1)\ \frac{P \overset{\mathbf{enter}\, n}{\succ} (\nu \overrightarrow{p})\langle P'\rangle P'' \quad Q \overset{\overline{\mathbf{enter}}\, n}{\succ} (\nu \overrightarrow{q})\langle Q'\rangle Q''}{P|Q \xrightarrow{\tau} (\nu \overrightarrow{p})(\nu \overrightarrow{q})(n[Q'|P'] \mid P''|Q'')}\ (*)$$

$$(\mathbf{enter}_2)\ \frac{Q \overset{\mathbf{enter}\, n}{\succ} (\nu \overrightarrow{q})\langle Q'\rangle Q'' \quad P \overset{\overline{\mathbf{enter}}\, n}{\succ} (\nu \overrightarrow{p})\langle P'\rangle P''}{P|Q \xrightarrow{\tau} (\nu \overrightarrow{p})(\nu \overrightarrow{q})(n[Q'|P'] \mid P''|Q'')}\ (*)$$

Side condition $(*)$:

$$(fn(P') \cup fn(P'')) \cap \overrightarrow{q} = (fn(Q') \cup fn(Q'')) \cap \overrightarrow{p} = \emptyset$$

Fig. 4. The transition relation.

$$(!)\ \frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P'|!P}$$

$$(\mathbf{enter}_!)\ \frac{P \overset{\mathbf{enter}\, n}{\succ} (\nu \overrightarrow{p})\langle P'\rangle P'' \quad P \overset{\overline{\mathbf{enter}}\, n}{\succ} (\nu \overrightarrow{q})\langle Q'\rangle Q''}{!P \xrightarrow{\tau} (\nu \overrightarrow{p})(\nu \overrightarrow{q})(n[Q'|P'] \mid P''|Q'')\ |!P}\ (*)$$

Side condition $(*)$:

$$(fn(P') \cup fn(P'')) \cap \overrightarrow{q} = (fn(Q') \cup fn(Q'')) \cap \overrightarrow{p} = \emptyset$$

Fig. 5. The transition rules for replication.

$(\overline{vm})(v)\langle m[P]\rangle\mathbf{0} = (v)\langle (vm)m[P]\rangle\mathbf{0}$. Thus, the left part of the process has the hardening $(vm)(m[\mathbf{in}\ n.P]) \overset{\mathbf{enter}\, n}{\succ} (v)\langle (vm)m[P]\rangle\mathbf{0}$. By $(\overline{\mathbf{enter}}\, h)$, the right part has the hardening $n[Q] \overset{\overline{\mathbf{enter}}\, n}{\succ} (v)\langle Q\rangle\mathbf{0}$. By rule $(\mathbf{enter}_1)$, we thus obtain

$$(vm)(m[\mathbf{in}\ n.P]) \mid n[Q] \xrightarrow{\tau} n[Q \mid (vm)m[P]\ ],$$

where empty restrictions and nil processes are left out for sake of conciseness.

Remark 5. Since we require our LTS to adhere to the GSOS format, it is not possible to deal with replication by adding the standard rule

$$(!_s)\ \frac{P \mid !P \xrightarrow{\alpha} Q}{!P \xrightarrow{\alpha} Q}.$$

The reason is that the sources of all premises of a rule have to be single processes in order to be compatible to the GSOS format (see Remark 17). Instead, replication is axiomatized by the GSOS rules in Fig. 5.

Example 6. To illustrate the use of rule $(\mathbf{enter}_!)$, consider the process $P = n[\mathbf{in}\ n.\mathbf{0}]$. Obviously, $!P \longrightarrow n[\mathbf{in}\ n.\mathbf{0} \mid n[\mathbf{0}]\ ] \mid !P$. This reduction can be mimicked by our system: By $(\mathbf{enter}_!)$, $(\mathbf{enter}\, h)$, (pre), and finally $(\overline{\mathbf{enter}}\, h)$, we have $!P \xrightarrow{\tau} n[\mathbf{in}\ n.\mathbf{0} \mid n[\mathbf{0}]\ ] \mid !P$.

Gordon and Cardelli present a different LTS for the ambient calculus in [6] (recalled in Appendices A and B). We write $P \xrightarrow{\alpha}_{\mathrm{GC}} Q$ to indicate that the process P can reduce to the process Q by a transition with label α which is justified by a rule of the LTS from [6]. (Unlike the system in [6], our system does not take input and output primitives into account; the treatment of these features in our framework is the subject of further investigation.)

Lemma 7 (*Gordon and Cardelli [6]*). *If* $P \succ (\nu\overrightarrow{p})\langle P'\rangle P''$ *then* $P \equiv (\nu\overrightarrow{p})(P' \mid P'')$ *and* $\overrightarrow{p} \subset fn(P')$.

Lemma 8. *Let* $\beta \in \{\mathbf{enter}\ n, \mathbf{exit}\ n\}$. *Then every hardening with label* β *is, up to structural congruence, of the form*

$$(\nu\overrightarrow{p})(m[P]|Q) \overset{\beta}{\succ} (\nu\overrightarrow{p})\langle m[P']\rangle Q,$$

where $P \xrightarrow{\alpha} P'$ *with* $\alpha = \mathbf{in}\ n$ *if* $\beta = \mathbf{enter}\ n$ *and* $\alpha = \mathbf{out}\ n$ *if* $\beta = \mathbf{exit}\ n$, $n \notin \overrightarrow{p}$, *and* $\overrightarrow{p} \subset fn(m[P']) \cap fn(Q)$.

Lemma 9. *Let* $\beta \in \{\overline{\mathbf{enter}}\ n, \overline{\mathbf{open}}\ n\}$. *Then every hardening with label* β *is, up to structural congruence, of the form*

$$(\nu\overrightarrow{p})(n[P']|P'') \overset{\beta}{\succ} (\nu\overrightarrow{p})\langle P'\rangle P'',$$

where $n \notin \overrightarrow{p}$ *and* $\overrightarrow{p} \subset fn(P') \cap fn(P'')$.

Lemma 10. *If* $P \xrightarrow{M}_{\mathrm{GC}} P'$ *then* $fn(M) \subseteq fn(P)$.

Lemma 11. *Let M be a capability. Up to structural congruence, every M-transition is of the form*

$$(\nu\overrightarrow{p})(M.P'|P'') \xrightarrow{M} (\nu\overrightarrow{p})(P'|P''),$$

where $fn(M) \cap \overrightarrow{p} = \emptyset$.

Lemma 12. *Let M be a capability. If* $P \xrightarrow{M} Q$ *then* $P \xrightarrow{M}_{\mathrm{GC}} \equiv Q$.

Notation 13. For $\alpha \in Act$ we write $P \xrightarrow{\alpha} \equiv Q$ to denote $\exists P'.P \xrightarrow{\alpha} P' \wedge P' \equiv Q$, and $P \xrightarrow{\alpha}_{\mathrm{GC}} \equiv Q$ to denote $\exists P'.P \xrightarrow{\alpha}_{\mathrm{GC}} P' \wedge P' \equiv Q$.

Theorem 14. *The LTS in GSOS format is, up to structural congruence, sound and complete with respect to the LTS of* [6]: *For* $\alpha \in Action$, $P \xrightarrow{\alpha} Q$ *implies* $P \xrightarrow{\alpha}_{\mathrm{GC}} \equiv Q$ (*soundness*) *and* $P \xrightarrow{\alpha}_{\mathrm{GC}} Q$ *implies* $P \xrightarrow{\alpha} Q$ (*completeness*).

The proof (cf. Appendix C) uses Theorem 26, proved independently below.

Corollary 15. *The LTS in GSOS format is, up to structural congruence, sound and complete with respect to the reduction relation of the ambient calculus as recalled in Section* 2. *Formally*: $P \xrightarrow{\tau} \equiv Q$ *implies* $P \longrightarrow Q$ (*soundness*) *and* $P \longrightarrow Q$ *implies* $P \xrightarrow{\tau} \equiv Q$ (*completeness*).

Proof. A special case of Theorem 14 is that $P \xrightarrow{\tau} \equiv Q$ iff $P \xrightarrow{\tau}_{\mathrm{GC}} \equiv Q$, and [6], Theorem 9, states that $P \xrightarrow{\tau}_{\mathrm{GC}} \equiv Q$ iff $P \longrightarrow Q$. □

3.2. Coalgebraic semantics

The mobility aspects of the LTS defined above can be modelled in a coalgebraic manner. This amounts to designing a behaviour functor which captures the possible observations on an ambient calculus process. These observations apparently include not only the reductions in the LTS, but also the concretions to which a process hardens. A somewhat subtle point here is the treatment of hidden names in concretions. This may be implemented by closing the set of concretions of a process under α-equivalence (this is perhaps less elegant than treating name freshness using named

sets or similar approaches as e.g. in [5,21], but has the advantage of being specifiable in a purely set-theoretic framework such as CoCasl) while keeping the set of locally *hidden* names as an extra component of the hardening observation; since the name space $\mathcal{N}$ is countably infinite, we thus obtain countable sets of concretions for each state, although morally, the system remains finitely branching. The implementation of α-renaming requires an additional observation, the (finite) set of *free* names appearing in a process. We thus arrive at the functor A defined by

$$AX = (Act \to \mathcal{P}_\omega(X)) \times (HAct \to \mathcal{P}_{\omega_1}(X \times X \times \mathcal{P}_\omega(\mathcal{N}))) \times \mathcal{P}_\omega(\mathcal{N}),$$

where $\mathcal{P}_\omega$ and $\mathcal{P}_{\omega_1}$ are the finite and countable powerset functors, respectively. A coalgebra for this functor on a set X is of the form $\langle next, harden, names\rangle$, where $next(x) : Act \to \mathcal{P}_\omega(X)$ is a function which maps each transition label to the set of the corresponding successor states, $harden(x) : HAct \to \mathcal{P}_{\omega_1}(X \times X \times \mathcal{P}_\omega(\mathcal{N}))$ is a function which maps hardening labels to the set of the corresponding concretions of the state, and $names(x)$ is the above-mentioned set of free names.

Due to the size limits on the involved power sets, there exists a final A-coalgebra $\mathcal{A}$. Below, we will define the semantics

$$[\![P]\!] \in \mathcal{A}$$

of an ambient calculus term by means of corecursive definitions of the process building operations. (For the sake of readability, we will omit the semantic brackets $[\![\_]\!]$ in the equations themselves.)

It should be noted that the crucial difference between A and the standard functor $\mathcal{P}_\omega(A \times \_)$ for LTS lies in the fact that the hardening part is essentially of the type $\lambda X.\, \mathcal{P}_{\omega_1}(X \times X \times \mathcal{P}_\omega(\mathcal{N}))$; here, the peculiarity is captured that a process splits into two parts (with some shared local names) for purposes of further reduction. There is good indication that this feature is indeed the essence of 'mobility', since it is instrumental in the modelling of 'moving and leaving others behind'.

We explicitly fix a syntactic notion of name interchange (this is easier to handle than arbitrary renaming, a fact which has first been noticed in connection with nominal calculi [5]):

Definition 16. We denote the permutation on $\mathcal{N}$ interchanging two names p, q by (p, q). The effect of (p, q) on a process term P is to replace all free occurrences of names p by q and conversely; the resulting process term is denoted $(p, q) \cdot P$. Similarly, we denote the application of (p, q) to an action α, a hardening action β, or a name n by $(p, q) \cdot \alpha$, $(p, q) \cdot \beta$, and $(p, q) \cdot n$, respectively (in particular, e.g. $(p, q) \cdot n = n$ if $n \notin \{p, q\}$).

One of the benefits of the coalgebraic view is that the process building operations of the ambient calculus can now be defined in a corecursive style, and hence become more easily tractable in coinductive proofs. This means that we take the final A-coalgebra $\mathcal{A}$ as our semantic domain for the interpretation of processes. To warm up, we give a corecursive definition of the (purely auxiliary) semantic name interchange operation, which will later be employed in the corecursive definition of name restriction: for $p, q \in \mathcal{N}$, the name interchange function $switch_{pq}$ is given by the equations

$$\begin{aligned} next(switch_{pq}(P))(\alpha) &= switch_{pq}[next(P)((p, q) \cdot \alpha)], \\ harden(switch_{pq}(P))(\beta) &= (switch_{pq} \times switch_{pq})[harden(P)((p, q) \cdot \beta)], \\ names(switch_{pq}(P)) &= switch_{pq}[names(P)]. \end{aligned}$$

For an action α, we define

$$actnames(\alpha) = \begin{cases} \emptyset & \text{if } \alpha = \tau, \\ \{n\} & \text{if } \alpha = M \text{ for the single ambient name } n \text{ in } M. \end{cases}$$

The corecursive definitions of the process building operations are shown in Figs. 6–11.

Remark 17. A word of explanation is in order as to why the above equations actually constitute good corecursive definitions. The format of these equations deviates from the standard coiteration format in that the right-hand sides contain composite expressions of the language being interpreted, rather than just one application of the single operation

$$
\begin{aligned}
&\mathit{next}(\alpha, \mathbf{0}) = \emptyset \\
&\mathit{harden}(\beta, \mathbf{0}) = \emptyset \\
&\mathit{names}(\mathbf{0}) = \emptyset
\end{aligned}
$$

Fig. 6. Corecursive definition of the zero process.

$$
\begin{aligned}
&\mathit{next}(\alpha, \alpha'.P) = \begin{cases} \{P\}, & \text{if } \alpha = \alpha' \\ \emptyset, & \text{otherwise} \end{cases} \\
&\mathit{harden}(\beta, \alpha'.P) = \emptyset \\
&\mathit{names}(\alpha'.P) = \mathit{actnames}(\alpha') \cup \mathit{names}(P)
\end{aligned}
$$

Fig. 7. Corecursive definition of action prefixing.

$$
\begin{aligned}
&\mathit{next}(\tau, P \mid Q) = \\
&\quad \{(\nu\,\vec{q})(P' \mid Q' \mid Q'') \bullet P' \in \mathit{next}(\mathbf{open}\, n, P), \\
&\qquad (Q', Q'', \vec{q}) \in \mathit{harden}(\overline{\mathbf{open}}\, n, Q),\ \vec{q} \cap \mathit{names}(P) = \emptyset\} \\
&\cup\ \{(\nu\,\vec{p})(P' \mid P'' \mid Q') \bullet Q' \in \mathit{next}(\mathbf{open}\, n, Q), \\
&\qquad (P', P'', \vec{p}) \in \mathit{harden}(\overline{\mathbf{open}}\, n, P),\ \vec{p} \cap \mathit{names}(Q) = \emptyset\} \\
&\cup\ \{(\nu\,\vec{p})(\nu\,\vec{q})(n[Q' \mid P'] \mid P'' \mid Q'') \bullet \\
&\qquad (P', P'', \vec{p}) \in \mathit{harden}(\overline{\mathbf{enter}}\, n, P), \\
&\qquad (Q', Q'', \vec{q}) \in \mathit{harden}(\overline{\mathbf{enter}}\, n, Q), \\
&\qquad \vec{p} \cap (\mathit{names}(Q') \cup \mathit{names}(Q'')) = \emptyset, \\
&\qquad \vec{q} \cap (\mathit{names}(P') \cup \mathit{names}(P'')) = \emptyset\} \\
&\cup\ \{(\nu\,\vec{p})(\nu\,\vec{q})(n[Q' \mid P'] \mid P'' \mid Q') \bullet \\
&\qquad (P', P'', \vec{p}) \in \mathit{harden}(\overline{\mathbf{enter}}\, n, P), \\
&\qquad (Q', Q'', \vec{q}) \in \mathit{harden}(\mathbf{enter}\, n, Q), \\
&\qquad \vec{p} \cap (\mathit{names}(Q') \cup \mathit{names}(Q'')) = \emptyset, \\
&\qquad \vec{q} \cap (\mathit{names}(P') \cup \mathit{names}(P'')) = \emptyset\} \\
&\cup\ \{P' \mid Q \bullet P' \in \mathit{next}(\tau, P)\} \\
&\cup\ \{P \mid Q' \bullet Q' \in \mathit{next}(\tau, Q)\} \\
&\mathit{next}(\alpha, P \mid Q) = \\
&\quad \{P' \mid Q \bullet P' \in \mathit{next}(\alpha, P)\} \\
&\cup\ \{P \mid Q' \bullet Q' \in \mathit{next}(\alpha, Q)\} \quad (\alpha \neq \tau) \\
&\mathit{harden}(\beta, (P \mid Q)) = \\
&\quad \{(P', P'' \mid Q, \vec{p}) \bullet (P', P'', \vec{p}) \in \mathit{harden}(\beta, P), \vec{p} \cap \mathit{names}(Q) = \emptyset\} \\
&\cup\ \{(Q', P \mid Q'', \vec{p}) \bullet (Q', Q'', \vec{p}) \in \mathit{harden}(\beta, Q), \vec{p} \cap \mathit{names}(P) = \emptyset\} \\
&\mathit{names}(P \mid Q) = \mathit{names}(P) \cup \mathit{names}(Q)
\end{aligned}
$$

Fig. 8. Corecursive definition of composition.

$$
\begin{aligned}
&\mathit{next}(\alpha, !P) = \\
&\quad \{P' \mid !P \bullet P' \in \mathit{next}(\alpha, P)\} \\
&\cup\ \{(\nu\,\vec{p})(\nu\,\vec{q})(n[Q' \mid P'] \mid P'' \mid Q'') \mid !P \bullet \\
&\qquad (P', P'', \vec{p}) \in \mathit{harden}(\overline{\mathbf{enter}}\, n, P), \\
&\qquad (Q', Q'', \vec{q}) \in \mathit{harden}(\overline{\mathbf{enter}}\, n, P), \\
&\qquad \vec{p} \cap (\mathit{names}(Q') \cup \mathit{names}(Q'')) = \emptyset, \\
&\qquad \vec{q} \cap (\mathit{names}(P') \cup \mathit{names}(P'')) = \emptyset\} \\
&\mathit{harden}(\beta, !P) = \\
&\quad \{(P', P'' \mid !P, \vec{p}) \bullet (P', P'', \vec{p}) \in \mathit{harden}(\beta, P)\} \\
&\mathit{names}(!P) = \mathit{names}(P)
\end{aligned}
$$

Fig. 9. Corecursive definition of replication.

$$
\begin{aligned}
&next(\tau, n[P]) = \\
&\qquad \{(\nu\,\overrightarrow{p})(n[P''] \mid P') \bullet (P', P'', \overrightarrow{p}) \ \in harden(\mathbf{exit}\ n, P)\} \\
&\quad \cup\ \{n[P'] \bullet P' \ \in next(\tau, P)\} \\
&next(\alpha, m[P]) = \emptyset \qquad (\alpha \neq \tau) \\
&harden(\mathbf{enter}\ n, m[P]) = \\
&\qquad \{(m[P'], \mathbf{0}, \emptyset) \bullet P' \ \in next(\mathbf{in}\ n, P)\} \\
&harden(\overline{\mathbf{enter}}\ n, (m[P])) = \begin{cases} \{(P, \mathbf{0}, \emptyset)\}, & \text{if } m = n \\ \emptyset, & \text{otherwise} \end{cases} \\
&harden(\mathbf{exit}\ n, m[P]) = \\
&\qquad \{(m[P'], \mathbf{0}, \emptyset) \bullet P' \ \in next(\mathbf{out}\ n, P)\} \\
&harden(\overline{\mathbf{open}}\ n, (m[P])) = \begin{cases} \{(P, \mathbf{0}, \emptyset)\}, & \text{if } m = n \\ \emptyset, & \text{otherwise} \end{cases} \\
&names(n[P]) = \{n\} \cup names(P)
\end{aligned}
$$

Fig. 10. Corecursive definition of the ambient operator.

$$
\begin{aligned}
&next(\alpha, (\nu n)P) = \\
&\qquad \{(\nu n)R \bullet R \ \in next(\alpha, P),\ n \notin actnames(\alpha)\} \\
&harden(\beta, (\nu n)P) = \\
&\qquad \{(R, (\nu n)S, \overrightarrow{p}) \bullet (R, S, \overrightarrow{p}) \in harden(\beta, P),\ n \notin names(R) \cup \overrightarrow{p}\} \\
&\quad \cup\ \{((\nu n)R, S, \overrightarrow{p}) \bullet (R, S, \overrightarrow{p}) \ \in harden(\beta, P),\ n \notin names(S) \cup \overrightarrow{p}\} \\
&\quad \cup\ \{(switch_{nm}R, switch_{nm}S, \overrightarrow{p} \cup \{m\}) \bullet \\
&\qquad\quad (R, S, \overrightarrow{p}) \ \in harden(\beta, P),\ m \notin names(P) \cup \overrightarrow{p}, \\
&\qquad\quad n \in (names(R) \cap names(S)) \setminus \overrightarrow{p}\} \qquad \text{if } n \notin fn(\beta) \\
&harden(\beta, (\nu n)P) = \emptyset \qquad \text{otherwise} \\
&names((\nu n)P) = names(P) \setminus \{n\}
\end{aligned}
$$

Fig. 11. Corecursive definition of name restriction.

being defined. According to the results of [24], a semantics for a process calculus with signature functor Σ in coalgebras for a behaviour functor B can be defined by exhibiting an *abstract GSOS law*, i.e. a natural transformation

$$\rho : \Sigma(Id \times B) \to BT,$$

where T is the free monad (i.e. the term algebra functor) over Σ (cf. also [1]). Note that the Σ in the source corresponds to the different process operations on the left-hand sides of the corecursive equations. The B in the source means that the arguments of the corecursive equations are not given as processes, but as the possible observations on these. While we still use process variables in the left-hand sides of the corecursive equations, in the right-hand sides, these are never used directly, but only via their observations. The B in the target corresponds to the definition of the corecursive operations through their observations. Finally, the appearance of T in the target offers the possibility of using composite process terms, as required for the right-hand sides of the corecursive equations.

The semantic function $h : \Sigma S \to S$, where (S, ζ) is the final B-coalgebra, is the unique so-called ρ-model over (S, ζ), i.e. uniquely determined by the equation

$$\zeta \circ h = Bh^* \circ \rho_S \circ \Sigma\langle id, \zeta\rangle, \qquad (*)$$

where $h^* : TS \to S$ is the T-algebra determined by h. If the semantic function h is omitted from the notation, as done above, then Eq. $(*)$ becomes precisely the format of our corecursive definitions. This shows that our corecursive equations have a unique solution provided that ρ_S is part of a natural transformation ρ. If, as is the case here, Σ and B are κ-accessible for some regular cardinal κ and $|S| \geqslant \kappa$, then it suffices to check that ρ_S is natural for self-maps of S: we then obtain the components ρ_X for $|X| < \kappa$, natural in X, by restriction of ρ_Z, and from these ρ_X we can assemble all of ρ by taking κ-directed unions.

Verification of the naturality condition for ρ_S as given in the corecursive equations above is tedious but straightforward. The point is essentially that the rules adhere to similar restrictions as standard GSOS rules for the definition of LTS, in particular do not depend on equality of states, do not introduce new state variables in the conclusion, and never look ahead more than one step (the latter could in fact also be handled by means of so-called tree rules [24]).

The bialgebraic approach guarantees that the process building operations are really of an algebraic nature. In particular, equality on the final coalgebra, i.e. bisimilarity, is a congruence for the process building operations. Moreover, the semantics is *compositional*, i.e. the interpretation of composite terms is recursively derived from that of single operations [24]. (This does not, incidentally, contradict the previously diagnosed impossibility of a compositional LTS semantics for the ambient calculus [25], since our semantic domain is more than just an LTS.)

We explicitly record the agreement of the coalgebra structure arising from the corecursive definitions with the corresponding syntactic counterparts, in particular the LTS and the hardening relation defined in Section 3.1:

Lemma 18. *For every ambient calculus term P,*

$$\begin{aligned} next(\alpha, [\![P]\!]) &= \{[\![P']\!] \mid P \xrightarrow{\alpha} P'\},\\ harden(\beta, [\![P]\!]) &= \{([\![P'_1]\!], [\![P'_2]\!], \overrightarrow{p}) \mid P \overset{\beta}{\succ} (\nu\overrightarrow{p})\langle P'_1\rangle P'_2\}, \quad \textit{and}\\ names([\![P]\!]) &= fn(P). \end{aligned}$$

(Recall that processes are implicitly identified under α-equivalence, so that the set describing the hardenings of $[\![P]\!]$ is closed under α-renaming.)

The notion of A-bisimulation arising from the coalgebraic modelling according to the definitions recalled in Section 1 can be described as follows. A relation ρ between two A-coalgebras is an *A-simulation* if $x\rho y$ implies that

- for each $x' \in next(\alpha, x)$, there exists $y' \in next(\alpha, y)$ such that $x'\rho y'$;
- for each $(x', x'', \overrightarrow{p}) \in harden(x)$, there exists $(y', y'', \overrightarrow{p}) \in harden(y)$ such that $x'\rho y'$ and $x''\rho y''$; and
- $names(x) = names(y)$.

If, moreover, the inverse relation of ρ is also an A-simulation, then ρ is an A-bisimulation. States x and y are A-bisimilar (written $x \cong_A y$) if $x\rho y$ for some A-bisimulation. The relation $\cong_A$ is again an A-bisimulation.

The corresponding notion of A-bisimilarity can be brought into agreement with a natural notion of behavioural indistinguishability of ambient calculus terms:

Definition 19 (*Action bisimulation*). A relation ρ on ambient calculus processes is called an *action simulation* if for any two processes P, Q such that $P\rho Q$ the following hold:

(1) If $P \xrightarrow{\alpha} P'$ for some $\alpha \in Act$, then there exists a process Q' such that $Q \xrightarrow{\alpha} Q'$ and $P'\rho Q'$.

(2) If $P \overset{\beta}{\succ} (\nu\overrightarrow{p})\langle P'_1\rangle P'_2$ for some $\beta \in HAct$, then there exist processes Q'_1, Q'_2 such that

$$\begin{aligned} &Q \overset{\beta}{\succ} (\nu\overrightarrow{p})\langle Q'_1\rangle Q'_2,\\ &P'_1\rho Q'_1, \quad \text{and}\\ &P'_2\rho Q'_2. \end{aligned}$$

If, moreover, the inverse relation of ρ is also a action simulation, then ρ is an *action bisimulation*. If $P\rho Q$ for some action bisimulation ρ, then P and Q are called *action bisimilar*; the relation of action bisimilarity is denoted by $\cong_a$.

Proposition 20. *ρ is an action bisimulation iff $\{([\![P]\!], [\![Q]\!]) \mid P\rho Q\}$ is an A-bisimulation.*

Corollary 21. *Action bisimilarity and A-bisimilarity coincide, i.e.*

$$P \cong_a Q \quad \textit{iff} \quad [\![P]\!] \cong_A [\![Q]\!].$$

Corollary 22. *Action bisimilarity is a congruence.*

Proof. By the results of [24], the format of our corecursive definitions guarantees that A-bisimilarity is a congruence (cf. Remark 17); the claim then follows by Corollary 21. □

Remark 23. The bialgebraic setting allows for a more generous notion of bisimulation: one can use *bisimulation up to context* [1], i.e. simulating transitions need not end up in states related to the ones to be simulated under the original relation ρ itself, but only under the congruence closure of ρ w.r.t. the process building operations. In the proof of Theorem 26 below, we will need an even more liberal principle, where instead of the congruence closure of ρ we use the congruent *equivalence* relation $\bar{\rho}$ generated by ρ; let us call this form of bisimulation *relaxed bisimulation up to context*.

The soundness of relaxed bisimulation up to context as a proof principle for bisimilarity is proved by showing that $\bar{\rho}$ is a bisimulation. To prove the latter, one shows by induction over the derivation of $x\,\bar{\rho}\,y$ that transitions (or hardenings) of x can be simulated by y and conversely, ending up in states that are related under $\bar{\rho}$. The base of the induction is just the definition of relaxed bisimulation up to context. The steps for reflexivity and transitivity are straightforward; the step for symmetry is trivial, because the inductive claim is symmetric. The step for congruence relies on the fact that the rules are in GSOS format: if $x_i\,\bar{\rho}\,y_i$, $i = 1, \ldots, n$, and $\sigma(x_1, \ldots, x_n)$, where σ is a process building operator, has a transition (the case for hardenings is analogous) into a process term t derived by one of the GSOS rules, then the transitions (or hardenings) of the x_i appearing in the premise of the rule can, by the inductive assumption, be simulated by the y_i, ending up in states related under $\bar{\rho}$ to those for the x_i. We can then apply the same rule to obtain a transition for $\sigma(y_1, \ldots, y_n)$, ending up in a process term t' that is obtained from t by exchanging components for $\bar{\rho}$-related ones, hence $t\,\bar{\rho}\,t'$.

(These arguments can certainly be cast in a more general form; however, this leads beyond the scope of the present work.)

Lemma 24. *Let $\beta \in \{\overline{\textbf{enter}}\ n, \overline{\textbf{open}}\ n\}$ for a name n. A process P hardens under label β if and only if it exhibits the name n:*

$$(\exists C.\ P \overset{\beta}{\succ} C) \quad \textit{iff} \quad P \downarrow n.$$

Lemma 25. *Action bisimilarity is closed under exhibition of and convergence to names, i.e. if $P \cong_a Q$, then $P \downarrow n$ implies $Q \downarrow n$, and $P \Downarrow n$ implies $Q \Downarrow n$ with the same length of reduction.*

Theorem 26. *Structural congruence is contained in action bisimilarity.*

Remark 27. Fig. 12 shows some relations between various notions of bisimilarity and equivalence of ambient calculus processes. By Theorem 26, structural congruence $\equiv$ is contained in action bisimilarity $\cong_a$. By Theorem 14, the latter is contained in the bisimulation arising from Gordon and Cardelli's LTS [6]. Due to the hardening labels, it is clear that this containment is proper; however, if hardenings are included in Gordon and Cardelli's system, the bisimulations coincide. Action bisimilarity is also contained in *reduction barbed congruence* $\cong_p$ [12]: it is easy to see that $\cong_a$ is a barb-preserving (cf. Definition 3) reduction closed congruence, and $\cong_p$ is defined to be the largest such relation. The containment is proper: Due to the fact that action bisimilarity 'sees' the labels on transitions as well as on hardenings, it is clear from the outset that action bisimilarity $\cong_a$ does not prove the perfect firewall equation $(\nu n)n[P] = 0$, $n \notin fn(P)$ [6]—P may perform movements of so-called secret ambients, which are invisible from the point of view of reduction barbed congruence, but visible from the point of view of action bisimilarity. Hence, the congruence closure of the firewall equation $\mathbf{S}_\emptyset$ used in [3] is not contained in $\cong_a$, but is contained in $\cong_p$, since it is barb-preserving and reduction closed [3].

A related phenomenon is that action bisimilarity is, unlike reduction barbed congruence, sensitive to stuttering. It is a typical phenomenon in the coalgebraic modelling of process algebra that coalgebraic bisimilarity generally corresponds to strong notions of process equivalence; e.g. the coalgebraic notion of bisimilarity for CCS is strong bisimilarity (first steps towards a coalgebraic formulation of weak bisimilarity for CCS are taken in [19]). Various techniques are combined in [12] to obtain the desired properties of $\cong_p$ (in previous work on full abstraction results for the ambient calculus [10,11], extensions of the calculus by cocapabilities or passwords have played a crucial role); in particular, attention is restricted to env-actions, and for some of these, simulation is required only in a particular

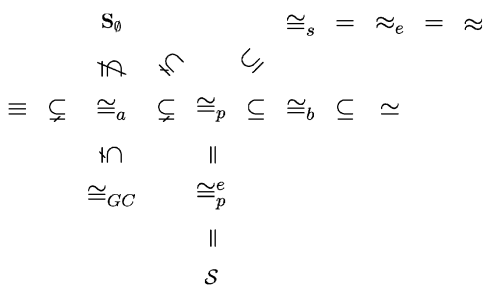

Fig. 12. Relations between different notions of ambient equivalence and bisimulation.

```
spec FINALLTS =
  sort Label
  then cofree {
      sort State
      then free {
          type Set ::= {} | {__}(State) | __ ∪ __(Set; Set)
          op __ ∪ __ : Set × Set → Set,
                  assoc, comm, idem, unit {}  }
      then cotype State ::= (next : Label → Set)  }
end
```

Fig. 13. CoCasl specification of the final LTS.

context. It is the subject of further research to determine whether these or similar techniques can be adapted to the LTS as presented here. We emphasize that the main concern of the present work is to provide a presentation of the ambient calculus in a format that makes it accessible to coalgebraic analysis; methods that allow extracting coarser process equivalences than coalgebraic bisimulation from the coalgebraic modelling are the subject of further research.

Reduction barbed congruence [12] comes in two variants: $\cong_p$ is defined using congruence w.r.t. *all* processes, while for $\cong_s$, this is restricted to so-called *systems*, which exclude prefixing and replication at the top level. While $\cong_s$ can be characterized by late and early bisimulation ($\approx$, $\approx_e$) of a suitable LTS, the characterization of $\cong_p$ is harder—$\cong_p$ is only characterized in terms of similar relations $\cong_p^e$ and $\mathcal{S}$ using fewer contexts. See [12] for details.

Barbed congruence $\cong_b$ is the context closure of the largest symmetric relation which is reduction closed and barb preserving [13,12]. Finally, *contextual equivalence* $\simeq$ is barb equivalence after applying a context and reduction (see Definition 3). It is straightforward to show that $\cong_p \subseteq \cong_b \subseteq \simeq$; it is at present unclear whether any of these inclusions is proper.

3.3. Formal specification of the ambient calculus in CoCasl

The algebraic–coalgebraic specification language CoCasl has been introduced in [16] as an extension of the standard algebraic specification language Casl. For the basic Caslsyntax, the reader is referred to [2,17]. We briefly explain the CoCasl features relevant for the understanding of the present work, before moving on to describe the CoCasl specification of the ambient calculus.

A simple but typical CoCasl specification is shown in Fig. 13. This specification defines the final finitely branching LTS over a given set of labels, exploiting both algebraic and coalgebraic aspects of CoCasl.

Several CoCasl features are nicely illustrated here. To begin, CoCasl offers a **cotype** construct which defines coalgebraic process types, dually to Casl's datatype construct **type**. Without further qualifications, type or cotype declarations essentially amount to just operator declarations; e.g. the type declaration in Fig. 13 gives rise to operators $\{\_\} : State \to Set$ etc. called *constructors*, while the cotype declaration produces an operator $next : State \times Label \to Set$, called a *selector* or *observer*. Like type declarations, cotype declarations may have several alternatives separated by |; while for types, this is just an enumeration of constructors, the effect of alternatives in a cotype declaration is the

```
spec FinSet [sort Elem] =
free {
  type  FinSet[Elem] ::=
        {} | {_}(Elem) | _ ∪ _(FinSet[Elem]; FinSet[Elem])
  op  _ ∪ _ : FinSet[Elem] × FinSet[Elem] → FinSet[Elem],
        assoc, comm, idem, unit {}  }
  then
  pred _ ∈ _ : Elem × FinSet[Elem]
  op  _ − _ : FinSet[Elem] × Elem → FinSet[Elem]
  ... %% recursive Definitions of ∈, −
spec CntSet [sort Elem] =
  cofree cotype Seq[Elem] ::= nil | (head : Elem; tail : Seq[Elem])
  pred _elem_ : Elem × Seq[Elem]
  var x : Elem; s : Seq[Elem]
  • x elem s ⇔ (x = head(s) ∨ x elem tail(s))
  then free {
      generated type CntSet[Elem] ::= mkSet(Seq[Elem])
      var s1, s2 : Seq[Elem]
      • mkSet(s1) = mkSet(s2) ⇔ ∀x : Elem . x elem s1 ⇔ x elem s2
      }
  then
  • op {} : CntSet[Elem] = mkSet(nil)
  ...
spec ACDomain[sort Name] = FinSet [sort Name] then
cofree {
  sort State
  free type Cap ::= in | out | open
  free type HCap ::= enter | coenter | exit | coopen
  free type Action ::= tau | action(Cap; name :?Name)
  free type HAct ::= haction(HCap; name : Name)
  free type Concretion ::= conc(State; State; FinSet[Name])
  then FinSet [sort State] and CntSet [sort Concretion]
  then cotype State ::= (next :    Action → FinSet[State];
                         harden : HAct → CntSet[Concretion];
                         names :  FinSet[Name])
  }
```

Fig. 14. CoCasl specification of the semantic domain for the ambient calculus.

generation of axioms emulating sum types, i.e. guaranteeing that the cotype is disjointly decomposed into the domains of the (partial) observers. E.g. writing

cotype *Process* ::= *cont*(*hd*1 :?*Elem*; *next* :?*Process*) | *fork*(*hd*2 :?*Elem*; *left* :?*Process*; *right* :?*Process*)

produces a process type that can in each step either just advance one step (*next*) or fork (*left*/*right*). It is shown in [16] that one can indeed define for each cotype signature a functor T such that models of the cotype correspond to T-coalgebras. E.g. the cotype *State* of Fig. 13 corresponds to coalgebras for $TX = Label \to \mathcal{P}_\omega(X)$, where $\mathcal{P}_\omega$ denotes the finite powerset functor, and the cotype *Process* above to coalgebras for $TX = Elem \times X + Elem \times X \times X$.

Cotypes can be qualified by keywords expressing further constraints. In particular, the keyword **cofree**, placed directly before the keyword **cotype**, restricts the models of a simple cotype such as *Process* to the final coalgebra (uniquely up to isomorphism), which in the case of *Process* consists of infinite *Elem*-labelled trees with branching degree either 1 or 2 at each node. In the context of this work, a more powerful mechanism is more important, which applies to complex cotypes such as *State* in Fig. 13: The keyword **cofree** may also be used to restrict the models of an entire specification, delimited as in Fig. 13 by curly brackets, to final models over a given model of the preceding specification—in the case of Fig. 13 over a given set of labels. This concept is dual to the Casl construct **free**, also appearing in Fig. 13, which restricts models of the following specification to be initial over a given model of the preceding specification, in the case of the type *Set* in Fig. 13 over a given set of states (and, irrelevantly, a given set of labels). (Subtle differences between **cofree** and **free** are discussed in [16]; this is not relevant for the understanding of the present work.)

$$
\begin{array}{ll}
\textbf{op} & cap \ : \ Cap \times Name \times State \to State \\
\textbf{vars} & a : Action; b : HAct; c : Cap; n : Name; P : State \\
\bullet & next(a, cap(c, n, P)) = \{P\} \ when \ a = action(c, n) \ else \ \{\} \\
\bullet & harden(\beta, cap(c, n, P)) = \{\} \\
\bullet & names(cap(c, n, P)) = \{n\} \ \cup names(P)
\end{array}
$$

Fig. 15. CoCasl specification of capability prefixing.

$$
\begin{array}{ll}
\textbf{op} & res \ : \ (Name \times State) \to State \\
\textbf{vars} & a : Action; b : HAct; n : Name; p : FinSet[Name]; P, Q : State; c : \\
 & Concretion \\
\bullet & Q \in next(a, res(n, P)) \Leftrightarrow \\
 & \quad \exists R : State \bullet Q = res(n, R) \wedge R \in next(a, P) \\
 & \quad \wedge \neg n \in actnames(a) \\
\bullet & c \in harden(b, res(n, P)) \Leftrightarrow \\
 & \quad (\exists R, S : State \bullet c = conc(R, res(n, S), p) \wedge \\
 & \qquad conc(R, S, p) \in harden(b, P) \wedge \neg n \in names(R) \cup p) \vee \\
 & \quad (\exists R, S : State \bullet c = conc(res(n, R), S, p) \wedge \\
 & \qquad conc(R, S, p) \in harden(b, P) \wedge \neg n \in names(S) \cup p) \vee \\
 & \quad (\exists R, S : State; m : Name \bullet \\
 & \qquad c = conc(rename(R, n, m), rename(S, n, m), p \cup \{m\}) \wedge \\
 & \qquad conc(R, S, p) \in harden(b, P) \wedge \\
 & \qquad \neg m \in names(P) \cup p \wedge n \in names(S) \cap names(R) - p) \\
\bullet & names(res(n, P)) = names(P) - n
\end{array}
$$

Fig. 16. CoCasl specification of name restriction.

Explicitly, this means that the type *Set* indeed consists of the set of constructor terms modulo associativity, commutativity, idempotence, and neutrality of {}, i.e. essentially of all finite subsets of *State*. The cotype *State* is thus really the final coalgebra for the functor $TX = Label \to \mathcal{P}_\omega(X)$, i.e. the final finitely branching LTS. This cotype, or process type, has been used in [16] in order to specify a coalgebraic denotational semantics for CCS as indicated in Section 1.

+A CoCasl specification of the final coalgebra for the ambient calculus functor A of Section 3.2 is shown in Fig. 14. The specification is based on specifications of finite and countable sets, parametric in the type of elements and instantiated here with states and concretions, respectively. While finite sets can be defined as a free algebra, countable sets are defined as a quotient of the cofree coalgebra of (finite and infinite) sequences.

The cotype *State* is the mentioned final A-coalgebra; it serves as a semantic domain for the interpretation of the ambient calculus operations. Since CoCasl does not have product and sum types, the presentation of A needs to be split up into various datatype definitions; this makes for a somewhat more verbose, but also clearer and more readable specification style.

The process building operations of the ambient calculus can then be defined corecursively as functions into the semantic domain *State* as laid out in the previous section. As examples, CoCasl specifications of capability prefixing and name restriction are shown in Figs. 15 and 16, respectively. The full CoCasl specifications are available under http://www.tzi.de/cofi/CASL-lib/CoCASL/AmbientCalculus.het.

Remark 28. Coalgebraic modal logic in the form included in the present CoCasl design (to be refined in future versions of the design) relies on extracting a single modal operator for each 'functor', i.e. for each datatype-valued observer [16]. The precise nature of the coalgebraic modal logic of the ambient calculus functor A, or any variants of it possibly suited for the analysis of weak notions of process equivalence, is the subject of future investigation. For now, we remark that the extracted binary modal operator [*harden*] for concretions is the naturally expected one: $[harden](\phi, \psi)$ holds in a state x iff for every concretion $conc(y, z, N)$ in $harden(x)$, y satisfies ϕ and z satisfies ψ. A similar modal operator for higher-order process calculi has appeared in [23].

4. Conclusion

We have described a transition semantics for the ambient calculus that correctly captures the ambient calculus in the sense that its silent reductions coincide with the reduction relation of the ambient calculus. Similar results have been obtained in [11] for the safe ambient calculus [10] with passwords, and in [6,12] for the ambient calculus itself. Our system corresponds to a subsystem of that in [12], adapted in a such a way that it adheres to the GSOS format [24] and hence fits into a coalgebraic framework.

The coalgebraic treatment of our transition semantics exhibits more structure than LTS. The coalgebraic structure is based on two kinds of observations: one can observe firstly the successor states in the traditional sense of process algebra, and secondly the set of ways ('concretions') in which a top-level process can be singled out for interaction with the ambient structure. Here, the set of concretions is particularly noteworthy; we expect that this part of the functor, essentially the composite of the powerset functor and the squaring functor, points to a fundamental aspect of mobile calculi—processes split up into parts that move and others that remain behind. Moreover, we have described a purely set-theoretic coalgebraic treatment of shared private names via anonymization through α-equivalence.

The corecursive definition of process building operations implies the possibility of proving algebraic laws about the ambient calculus using coinduction, based on notions of bisimulation arising from the coalgebraic semantics. This standard coalgebraic bisimilarity relation is finer than reduction barbed congruence [12], which in turn is finer than contextual equivalence of ambients [3]. The characterization of contextual equivalence in terms of a weakened notion of bisimilarity, in broad analogy e.g. to weak bisimilarity in CCS and possibly building on the results of [12], is the subject of further research.

We expect that the program of characterizing process and mobile calculi using coalgebras over certain functors will eventually lead to a systematic understanding of the nature of these calculi. The calculi are usually presented using some concrete syntax plus some transition rules; and there are many variations of the syntax and the rules whose impact on the nature of the respective calculus is not clear from the outset. By contrast, the representation using operations on a coalgebra for a certain functor immediately determines (through the functor) the fundamental observations that can be made, while the operations (that correspond to the syntax of the respective calculus) may vary without changing the fundamental nature of the meaning of processes. Hence, it is also expected that different calculi can be related and combined much more easily using the coalgebraic representation. Future work will substantiate this point by considering further mobile calculi. Moreover, the behaviour functor more or less automatically comes with an expressive modal logic (cf. [22] and references therein); further work will include the investigation of this logic, in particular in relation to ambient logic [4].

Appendix A. The LTS of Gordon and Cardelli [6]

(Rules for communication omitted.)

$$(\textit{Trans Cap})\ \frac{P \succ (\nu \overrightarrow{p})\langle M.P'\rangle P''}{P \xrightarrow{M}_{\mathrm{GC}} (\nu \overrightarrow{p})(P'|P'')}\ (\mathit{fn}(M) \cap \overrightarrow{p} = \emptyset)$$

$$(\textit{Trans Amb})\ \frac{P \succ (\nu \overrightarrow{p})\langle n[Q]\rangle P' \qquad Q \xrightarrow{\tau}_{\mathrm{GC}} Q'}{P \xrightarrow{\tau}_{\mathrm{GC}} (\nu \overrightarrow{p})(n[Q']|P')}$$

$$(\textit{Trans In})\ \frac{\begin{array}{c} P \succ (\nu \overrightarrow{p})\langle n[Q]\rangle R \\ Q \xrightarrow{\mathbf{in}\, m}_{\mathrm{GC}} Q' \\ R \succ (\nu \overrightarrow{r})\langle m[R']\rangle R'' \end{array}}{P \xrightarrow{\tau}_{\mathrm{GC}} (\nu \overrightarrow{p}, \overrightarrow{r})(m[n[Q']|R']|R'')}\ (\overrightarrow{r} \cap \mathit{fn}(n[Q]) = \emptyset,\ \overrightarrow{p} \cap \overrightarrow{r} = \emptyset)$$

$$(\textit{Trans Out})\ \frac{\begin{array}{c} P \succ (\nu \overrightarrow{p})\langle n[Q]\rangle P' \\ Q \succ (\nu \overrightarrow{q})\langle m[R]\rangle Q' \\ R \xrightarrow{\mathbf{out}\, n}_{\mathrm{GC}} R' \end{array}}{P \xrightarrow{\tau}_{\mathrm{GC}} (\nu \overrightarrow{p})((\nu \overrightarrow{q})(m[R']|n[Q'])|P')}\ (n \notin \overrightarrow{q})$$

$$(\textit{Trans Open})\ \frac{P \succ (\nu \overrightarrow{p})\langle n[Q]\rangle P' \quad P' \xrightarrow{\mathbf{open}\, n}_{\mathrm{GC}} P''}{P \xrightarrow{\tau}_{\mathrm{GC}} (\nu \overrightarrow{p})(Q|P'')}$$

Appendix B. The hardening relation of Gordon and Cardelli [6]

(Rules for communication omitted.)

$$(\textit{Harden Action})\ \frac{\alpha \in \{\mathbf{in}\, n,\ \mathbf{out}\, n,\ \mathbf{open}\, n\}}{\alpha.P \succ (\nu)\langle \alpha.P\rangle \mathbf{0}}$$

$$(\textit{Harden Amb})\ \frac{}{n[P] \succ (\nu)\langle n[P]\rangle \mathbf{0}}$$

$$(\textit{Harden Par 1})\ \frac{P \succ (\nu \overrightarrow{p})\langle P'\rangle P''}{P|Q \succ (\nu \overrightarrow{p})\langle P'\rangle P''|Q}\ (\overrightarrow{p} \cap fn(Q) = \emptyset)$$

$$(\textit{Harden Par 2})\ \frac{Q \succ (\nu \overrightarrow{q})\langle Q'\rangle Q''}{P|Q \succ (\nu \overrightarrow{q})\langle Q'\rangle P|Q''}\ (\overrightarrow{q} \cap fn(P) = \emptyset)$$

$$(\textit{Harden Repl})\ \frac{P \succ (\nu \overrightarrow{p})\langle P'\rangle P''}{!P \succ (\nu \overrightarrow{p})\langle P'\rangle P''|!P}$$

$$(\textit{Harden Res})\ \frac{P \succ C}{(\nu n)P \succ \overline{(\nu n)}C}$$

$$\overline{(\nu n)}(\nu \overrightarrow{p})\langle P_1\rangle P_2 = \begin{cases} (\nu \overrightarrow{p})\langle P_1\rangle (\nu n) P_2 & \text{if } n \notin fn(P_1), \\ (\nu \overrightarrow{p})\langle m[(\nu n)P_1']\rangle P_2 & \text{if } P_1 = m[P_1'],\ \ m \neq n,\ \ n \in fn(P_1) \setminus fn(P_2), \\ (\nu n, \overrightarrow{p})\langle P_1\rangle P_2 & \text{otherwise,} \end{cases} \qquad (n \notin \overrightarrow{p}).$$

Appendix C. Proofs

Notation 29. Whenever dealing with a process of the form $(\nu \overrightarrow{p})P$ where $\overrightarrow{p}$ is a sequence of i names, we denote by (*Harden Res*)*,(*ν*)* and (*νh*)*, respectively, the i-fold application of the corresponding rule.

Proof of Lemma 8. Induction over the derivation of the hardening. The only interesting case is rule (*νh*); here, structural congruence is used to distribute restrictions in the process in the same way as prescribed by the definition of hardenings of concretions above. □

Proof of Lemma 9. Analogous to Lemma 8. □

Proof of Lemma 10. Let $P \xrightarrow{M}_{\mathrm{GC}} P'$. It is then immediate by the (*Trans Cap*) rule from [6], that $P \succ (\nu \overrightarrow{p})\langle M.P''\rangle P'''$ for some $\overrightarrow{p}$, M, P'', P''', where $fn(M) \cap \overrightarrow{p} = \emptyset$. By Lemma 7, $P \equiv (\nu \overrightarrow{p})(M.P''|P''')$. Thus, $fn(M) \subseteq fn(M) \cup (fn(P'') \cup fn(P''')) \setminus \{\overrightarrow{p}\} = fn(\nu \overrightarrow{p})(M.P''|P''') = fn(P)$. □

Proof of Lemma 11. Straightforward induction over the derivation of the transition, noting that we need only consider the first five rules in Fig. 4. □

Proof of Lemma 12. By Lemma 11, we have that $P \equiv (\nu \overrightarrow{p})(M.P' \mid P'')$, $Q \equiv (\nu \overrightarrow{p})(P' \mid P'')$ and $fn(M) \cap \overrightarrow{p} = \emptyset$. It remains to show that $(\nu \overrightarrow{p})(M.P' \mid P'') \xrightarrow{M}_{\mathrm{GC}} \equiv (\nu \overrightarrow{p})(P' \mid P'')$. This is shown by application of the rules (*Trans Cap*), (*Harden Res*), (*Harden Par 1*) and finally (*Harden Amb*) of [6]. □

Proof of Theorem 14. *Soundness*: By Lemma 12, the only case left to show is where $\alpha = \tau$. The proof is then by induction over derivations. Lemma 35 of [6] states that structural congruence $\equiv$ is a bisimulation w.r.t. $\xrightarrow{\alpha}_{\mathrm{GC}}$,

i.e.

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

We can thus freely use $P \xrightarrow{\alpha}_{\mathrm{GC}} Q$ where the induction hypothesis only yields $P \xrightarrow{\alpha}_{\mathrm{GC}} \equiv Q$, since the above fact guarantees that we can defer $\equiv$ to the end of the reduction. Moreover, it suffices to establish transitions $\xrightarrow{\alpha}_{\mathrm{GC}}$ for structurally congruent modifications of the source term.

The cases of the induction are as follows.

(1) The reduction is of the form $P|Q \xrightarrow{\tau} (\nu\overrightarrow{p})(\nu\overrightarrow{q})(n[Q'''|P'|P''|Q''])$ and was derived by rule (**enter**$_2$). Then $Q \overset{\mathbf{enter}\,n}{\succ} (\nu\overrightarrow{q})\langle Q'''\rangle Q''$, $P \overset{\overline{\mathbf{enter}}\,n}{\succ} (\nu\overrightarrow{p})\langle P'\rangle P''$, and $(fn(Q''') \cup fn(Q'')) \cap \overrightarrow{p} = (fn(P') \cup fn(P'')) \cap \overrightarrow{q} = \emptyset$. By Lemma 8, $Q \equiv (\nu\overrightarrow{q})(m[\overline{Q'}]|Q'')$ and $Q''' = m[\overline{Q'''}]$, where $\overline{Q'} \xrightarrow{\mathbf{in}\,n} \overline{Q'''}$, $n \notin \overrightarrow{q}$. By Lemma 9, $P \equiv (\nu\overrightarrow{p})(n[P']|P'')$ where $n \notin \overrightarrow{p}$. The required transition

$$(\nu\overrightarrow{p})(n[P']|P'')|(\nu\overrightarrow{q})(m[\overline{Q'}]|Q'') \xrightarrow{\tau}_{\mathrm{GC}} (\nu\overrightarrow{p}, \overrightarrow{q})(n[m[\overline{Q'''}]|P']|P''|Q'')$$

is derived as follows. By rule (*Trans In*), the following has to be shown:

- $(\nu\overrightarrow{p})(n[P']|P'')|(\nu\overrightarrow{q})(m[\overline{Q'}]|Q'') \succ (\nu\overrightarrow{q})\langle m[\overline{Q'}]\rangle(\nu\overrightarrow{p})(n[P']|P'')|Q''$. This follows (since $fn((\nu\overrightarrow{p})(n[P']|P'')) \cap \overrightarrow{q} = \emptyset$) by application of the rules (*Harden Par* 2), (*Harden Res*)*, (*Harden Par 1*), and finally (*Harden Amb*).
- $\overline{Q'} \xrightarrow{\mathbf{in}\,n}_{\mathrm{GC}} \overline{Q'''}$. This holds by induction since $\overline{Q'} \xrightarrow{\mathbf{in}\,n} \overline{Q'''}$.
- $(\nu\overrightarrow{p})(n[P']|P'')|Q'' \succ (\nu\overrightarrow{p})\langle n[P']\rangle P''|Q''$. This can be seen (since $fn(Q'') \cap \overrightarrow{p} = \emptyset$) by application of the rules (*Harden Par* 1), (*Harden Res*)*, (*Harden Par* 1), and finally (*Harden Amb*).

(2) The reduction is of the form $P|Q \xrightarrow{\tau} (\nu\overrightarrow{p})(\nu\overrightarrow{q})(n[P'|Q']|P''|Q'')$ and was derived by rule (**enter**$_1$). This case is analogous to the previous case.

(3) The reduction is of the form $P|Q \xrightarrow{\tau} (\nu\overrightarrow{p})(P'|P''|Q')$ and was derived by rule (**open**$_2$). Then $Q \overset{\mathbf{open}\,n}{\longrightarrow} Q'$, $P \overset{\overline{\mathbf{open}}\,n}{\succ} (\nu\overrightarrow{p})\langle P'\rangle P''$ and $\overrightarrow{p} \cap fn(Q) = \emptyset$. By Lemma 9, $P \equiv (\nu\overrightarrow{p})(n[P']|P'')$, $n \notin \overrightarrow{p}$, $\overrightarrow{p} \subset fn(P')$ and $\overrightarrow{p} \subset fn(P'')$. It remains to show that

$$(\nu\overrightarrow{p})(n[P']|P'')|Q \xrightarrow{\tau}_{\mathrm{GC}} (\nu\overrightarrow{p})(P'|P''|Q').$$

By rule (*Trans Open*), we have to show the following:

- $(\nu\overrightarrow{p})(n[P']|P'')|Q \succ (\nu\overrightarrow{p})\langle n[P']\rangle(P''|Q)$. This can be shown by applying the rules (*Harden Par* 1), (*Harden Res*)*, (*Harden Par* 1) and finally (*Harden Amb*), since $\overrightarrow{p} \cap fn(Q) = \emptyset$, $\overrightarrow{p} \subset fn(n[P'])$, and $\overrightarrow{p} \subset fn(P'')$.
- $P''|Q \xrightarrow{\mathbf{open}\,n}_{\mathrm{GC}} P''|Q'$. This holds since $Q \xrightarrow{\mathbf{open}\,n} Q'$ implies $P''|Q \xrightarrow{\mathbf{open}\,n} P''|Q'$ by rule ($|_2$). By induction, we get $P''|Q \xrightarrow{\mathbf{open}\,n}_{\mathrm{GC}} P''|Q'$.

(4) The reduction is of the form $P|Q \xrightarrow{\tau} (\nu\overrightarrow{q})(P'|Q'|Q'')$ and was derived by rule (**open**$_1$). This case is analogous to the previous case.

(5) The reduction is of the form $n[P'] \xrightarrow{\tau} (\nu\overrightarrow{p})(n[P']|P'')$ and was derived by rule (**exit**). Then $P \overset{\mathbf{exit}\,n}{\succ} (\nu\overrightarrow{p})\langle P'''\rangle P'$. By Lemma 8, $P \equiv (\nu\overrightarrow{p})(m[\overline{P''}]|P'')$ for some $\overline{P''}$ such that $\overline{P''} \xrightarrow{\mathbf{out}\,n} \overline{P'''}$ and $P''' = m[\overline{P'''}]$. Furthermore, $n \notin \overrightarrow{p}$, $\overrightarrow{p} \subset fn(m[\overline{P''}])$ and $\overrightarrow{p} \subset fn(P')$. It remains to show that

$$n[(\nu\overrightarrow{p})(m[\overline{P''}]|P')] \xrightarrow{\tau}_{\mathrm{GC}} (\nu\overrightarrow{p})(n[P']|m[\overline{P'''}]).$$

By rule (*Trans Out*), this amounts to showing the following:

- $n[(\nu\overrightarrow{p})(m[\overline{P''}]|P')] \succ \nu\langle n[(\nu\overrightarrow{p})(m[\overline{P''}]|P')]\rangle\mathbf{0}$. This follows immediately from rule (*Trans Amb*).
- $(\nu\overrightarrow{p})(m[\overline{P''}]|P') \succ (\nu\overrightarrow{p})\langle m[\overline{P''}]\rangle P'$. This can be shown by applying rules (*Harden Res*)*, (*Harden Par* 1) and (*Harden Amb*), since $\overrightarrow{p} \subset fn(m[\overline{P'''}]) \subset fn(m[\overline{P''}])$ and $\overrightarrow{p} \subset fn(P')$.
- $\overline{P''} \xrightarrow{\mathbf{out}\,n}_{\mathrm{GC}} \overline{P'''}$. This holds by induction.

(6) The reduction is of the form $n[P] \xrightarrow{\tau} n[Q]$ and was inferred by rule (*amb*). Then $P \xrightarrow{\tau} Q$. It follows by rule (*Trans Amb*) that

$$n[P] \xrightarrow{\tau}_{\mathrm{GC}} (\nu)(n[Q]|\mathbf{0})$$

if the following hold:
- $n[P] \succ (\nu)\langle n[P]\rangle\mathbf{0}$. This follows immediately from rule (*Harden Amb*).
- $P \xrightarrow{\tau}_{\mathrm{GC}} Q$. This holds by induction.

(7) The reduction is of the form $!P \xrightarrow{\tau} (\nu\overrightarrow{p})(\nu\overrightarrow{q})(n[Q' \mid P''']] \mid P'' \mid Q'') \mid !P$ and was inferred by rule (**enter**$_!$). Then $P \overset{\mathbf{enter}\ n}{\succ} (\nu\overrightarrow{p})\langle P'\rangle P''$, $P \overset{\overline{\mathbf{enter}}\ n}{\succ} (\nu\overrightarrow{q})\langle Q'\rangle Q''$ and $(fn(P') \cup fn(P'')) \cap \overrightarrow{q} = (fn(Q') \cup fn(Q'')) \cap \overrightarrow{p} = \emptyset$. By Lemma 8, $P \equiv (\nu\overrightarrow{p})(m[\overline{P'}]|P'')$ and $P''' = m[\overline{P'''}]$, where $\overline{P'} \xrightarrow{\mathbf{in}\ n} \overline{P'''}$, $n \notin \overrightarrow{p}$. By Lemma 9, $P \equiv (\nu\overrightarrow{q})(n[Q']|Q'')$ where $n \notin \overrightarrow{q}$. The required transition

$$!P \xrightarrow{\tau}_{\mathrm{GC}} (\nu\overrightarrow{p}, \overrightarrow{q})(n[m[\overline{P'''}]|Q']|Q''|P'') \mid !P$$

is derived by rule (*Trans In*). The following remain to be shown:
- $!P \succ (\nu\overrightarrow{p})\langle m[\overline{P'}]\rangle P'' \mid !P$. This follows by application of the rules (*Harden Repl*), (*Harden Res*)*, (*Harden Par* 2), and finally (*Harden Amb*).
- $\overline{P'} \xrightarrow{\mathbf{in}\ n}_{\mathrm{GC}} \overline{P'''}$. This holds by induction since $\overline{P'} \xrightarrow{\mathbf{in}\ n} \overline{P'''}$.
- $P'' \mid !P \succ (\nu\overrightarrow{q})\langle n[Q']\rangle P'' \mid Q'' \mid !P$. This can be seen (since $fn(P'') \cap \overrightarrow{q} = \emptyset$) by application of the rules (*Harden Par* 1, (*Harden Repl*), (*Harden Res*)*, (*Harden Par* 1), and finally (*Harden Amb*).

The hardenings in the GC-LTS ensure that τ-transitions are closed under parallel composition, replication, and name restriction. The four remaining cases can thus be treated easily: The rules ($|_1$), ($|_2$), (ν), and (!) coincide directly with the rules (*Harden Par* 1), (*Harden Par* 2), (*Harden Res*), and (*Harden Repl*), respectively. Note that the rules (*Harden Par* 1) and (*Harden Par* 2) impose an additional requirement on the set of hidden names of the concretion. This requirement is met by rules ($|_1$) and ($|_2$) since we regard bound names as given only up to α-equivalence. If the rule (*Harden Res*) moves restrictions of names inside the prime or residue of the concretion, it is obvious that they can by moved out again after the τ-reduction by applying the according structural congruence rules.

Completeness: Again, by induction over the derivation. We make use of Theorem 26, proved independently below, stating that structural congruence is an action bisimulation. This fact implies that it suffices to establish transitions $\xrightarrow{\alpha}$ as well as hardenings for structurally congruent modifications of the source term.

First, let $\alpha = \tau$.

(1) The reduction is of the form $P \xrightarrow{\tau}_{\mathrm{GC}} (\nu\overrightarrow{p_1})(Q|P'')$ and was inferred by rule (*Trans Open*). Then $P \succ (\nu\overrightarrow{p_1})\langle n[Q]\rangle P'$ and $P' \xrightarrow{\mathbf{open}\ n}_{\mathrm{GC}} P''$. By Lemma 7, $P \equiv (\nu\overrightarrow{p_1})(n[Q]|P')$ and $\overrightarrow{p_1} \subset fn(n[Q])$. As $P' \xrightarrow{\mathbf{open}\ n}_{\mathrm{GC}} P''$, $P' \succ (\nu\overrightarrow{p_2})\langle \mathbf{open}\ n.R\rangle R'$ and $n \notin \overrightarrow{p_2}$. Again by Lemma 7, $P' \equiv (\nu\overrightarrow{p_2})(\mathbf{open}\ n.R|R')$ and $\overrightarrow{p_2} \subset fn(\mathbf{open}\ n.R)$; thus by rule (*Trans Cap*), $P'' \equiv (\nu\overrightarrow{p_2})(R|R')$. Putting everything together, we know that the source of the reduction is structurally congruent to $(\nu\overrightarrow{p_1})(n[Q]|(\nu\overrightarrow{p_2})(\mathbf{open}\ n.R|R'))$ and that the target of the reduction is structurally congruent to $(\nu\overrightarrow{p_1})(Q|(\nu\overrightarrow{p_2})(R|R'))$, where $n \notin \overrightarrow{p_2}$. To show that

$$(\nu\overrightarrow{p_1})(n[Q]|(\nu\overrightarrow{p_2})(\mathbf{open}\ n.R|R')) \xrightarrow{\tau} (\nu\overrightarrow{p_1})(Q|(\nu\overrightarrow{p_2})(R|R')),$$

it suffices by rule $(\nu)^*$ to show

$$n[Q]|(\nu\overrightarrow{p_2})(\mathbf{open}\ n.R|R') \xrightarrow{\tau} Q|(\nu\overrightarrow{p_2})(R|R').$$

This amounts by rule ($\mathbf{open}_2$) to showing that the following holds:
- $(\nu\overrightarrow{p_2})(\mathbf{open}\ n.R|R') \xrightarrow{\mathbf{open}\ n} (\nu\overrightarrow{p_2})(R|R')$. This can be shown by applying rules $(\nu)^*$, ($|_1$) and (*pre*).
- $n[Q] \overset{\overline{\mathbf{open}}\ n}{\succ} \nu\langle Q\rangle\mathbf{0}$. This follows immediately from rule ($\overline{\mathbf{open}}\ h$).

(2) The reduction is of the form $P \xrightarrow{\tau}_{\mathrm{GC}} (\nu\overrightarrow{p})((\nu\overrightarrow{q})(m[R']|n[Q'])|P')$ and was inferred by rule (*Trans Out*). Then $P \succ (\nu\overrightarrow{p})\langle n[Q]\rangle P'$, $Q \succ (\nu\overrightarrow{q})\langle m[R]\rangle Q'$ (where $n \notin \overrightarrow{q}$), and $R \xrightarrow{\mathbf{out}\ n}_{\mathrm{GC}} R'$. By Lemma 7, $P \equiv (\nu\overrightarrow{p})(n[Q]|P')$, $\overrightarrow{p} \subset fn(n[Q])$, and $Q \equiv (\nu\overrightarrow{q})(m[R]|Q')$, $\overrightarrow{q} \subset fn(m[R])$. It follows from $R \xrightarrow{\mathbf{out}\ n}_{\mathrm{GC}} R'$

that $R \succ (\nu\overrightarrow{r})\langle \mathbf{out}\ n.R''\rangle R'''$ where $n \notin \overrightarrow{r}$. Thus, again by Lemma 7, $R \equiv (\nu\overrightarrow{r})(\mathbf{out}\ n.R''|R''')$ and $\overrightarrow{r} \subset fn(\mathbf{out}\ n.R'')$, and by rule (*Trans Cap*), $R' \equiv (\nu\overrightarrow{r})(R''|R''')$. So far, we know that the source of the reduction is structurally congruent to $(\nu\overrightarrow{p})(n[(\nu\overrightarrow{q})(m[(\nu\overrightarrow{r})(\mathbf{out}\ n.R''|R''')]|Q')]|P')$ and that the target is structurally congruent to $(\nu\overrightarrow{p})((\nu\overrightarrow{q})(m[(\nu\overrightarrow{r})(R''|R''')]|n[Q'])|P')$, where $n \notin (\overrightarrow{q} \cup \overrightarrow{r})$. That

$$(\nu\overrightarrow{p})(n[(\nu\overrightarrow{q})(m[(\nu\overrightarrow{r})(\mathbf{out}\ n.R''|R''')]|Q')]|P') \xrightarrow{\tau} (\nu\overrightarrow{p})((\nu\overrightarrow{q})(m[(\nu\overrightarrow{r})(R''|R''')]|n[Q'])|P')$$

follows by application of the rules $(\nu)^*$, $(|_1)$, (**exit**), $(\nu h)^*$ ($n \notin \overrightarrow{q}$ and $\overrightarrow{q} \subset fn(m[R])$), $(|h_1)$, (**exit** h), $(\nu)^*$, $(|_1)$, and finally (*pre*).

(3) The reduction is of the form $P \xrightarrow{\tau}_{\mathrm{GC}} (\nu\overrightarrow{p}, \overrightarrow{r})(m[n[Q']|R']|R'')$ and was inferred by rule (*Trans In*). Then $P \succ (\nu\overrightarrow{p})\langle n[Q]\rangle R$, $R \succ (\nu\overrightarrow{r})\langle m[R']\rangle R''$ (where $\overrightarrow{r} \cap fn(n[Q]) = \emptyset$), $Q \xrightarrow{\mathbf{in}\ m}_{\mathrm{GC}} Q'$, and $\overrightarrow{p} \cap \overrightarrow{r} = \emptyset$). By Lemma 7, $P \equiv (\nu\overrightarrow{p})(n[Q]|R)$, $\overrightarrow{p} \subset fn(n[Q])$ and $R \equiv (\nu\overrightarrow{r})(m[R']|R'')$, $\overrightarrow{r} \subset fn(m[R'])$. Because of $Q \xrightarrow{\mathbf{in}\ m}_{\mathrm{GC}} Q'$, $Q \succ (\nu\overrightarrow{q})\langle \mathbf{in}\ m.Q''\rangle Q'''$, $m \notin \overrightarrow{q}$, and $Q' \equiv (\nu\overrightarrow{q})(Q''|Q''')$. Again by Lemma 7, $Q \equiv (\nu\overrightarrow{q})(\mathbf{in}\ m.Q''|Q''')$, $\overrightarrow{q} \subset fn(\mathbf{in}\ m.Q'')$. It also follows by Lemma 10 from $Q \xrightarrow{\mathbf{in}\ m}_{\mathrm{GC}} Q'$ that $m \in fn(Q)$. This yields, together with $\overrightarrow{r} \cap fn(n[Q]) = \emptyset$, that $m \notin \overrightarrow{r}$. Altogether, we have the following situation: The source of the reduction is structurally congruent to $(\nu\overrightarrow{p})(n[(\nu\overrightarrow{q})(\mathbf{in}\ m.Q''|Q''')]|(\nu\overrightarrow{r})(m[R']|R''))$ and the target of the reduction is structurally congruent to $(\nu\overrightarrow{p}, \overrightarrow{r})(m[n[(\nu\overrightarrow{q})(Q''|Q''')]|R']|R'')$, where $m \notin (\overrightarrow{r} \cup \overrightarrow{q})$. To show

$$(\nu\overrightarrow{p})(n[(\nu\overrightarrow{q})(\mathbf{in}\ m.Q''|Q''')]|(\nu\overrightarrow{r})(m[R']|R'')) \xrightarrow{\tau} (\nu\overrightarrow{p}, \overrightarrow{r})(m[n[(\nu\overrightarrow{q})(Q''|Q''')]|R']|R''),$$

it is by rule $(\nu)^*$ sufficient to show that

$$n[(\nu\overrightarrow{q})(\mathbf{in}\ m.Q''|Q''')]|(\nu\overrightarrow{pr})(m[R']|R'') \xrightarrow{\tau} (\nu\overrightarrow{r})(m[n[(\nu\overrightarrow{q})(Q''|Q''')]|R']|R'').$$

This holds by rule ($\mathbf{enter}_1$) (since $fn((\nu\overrightarrow{q})(Q''|Q''')) \cap \overrightarrow{r} = \emptyset$) if the following holds:

- $n[(\nu\overrightarrow{q})(\mathbf{in}\ m.Q''|Q''')] \overset{\mathbf{enter}\ m}{\succ} (\nu)\langle n[(\nu\overrightarrow{q})(Q''|Q''')]\rangle \mathbf{0}$. This can be shown by applying rules (**enter** h), $(\nu)^*$, $(|_1)$, and finally (*pre*).
- $(\nu\overrightarrow{r})(m[R']|R'') \overset{\overline{\mathbf{enter}}\ m}{\succ} (\nu\overrightarrow{r})\langle R'\rangle R''$. This can be shown by applying rules (νh) ($m \notin \overrightarrow{r}$, $\overrightarrow{p} \subset fn(m[R'])$), $(|h_1)$, and ($\overline{\mathbf{enter}}\ h$).

(4) The reduction is of the form and $P \xrightarrow{\tau}_{\mathrm{GC}} (\nu\overrightarrow{p})(n[Q']|P')$ and was inferred by rule (*Trans Amb*). Then $P \succ (\nu\overrightarrow{p})\langle n[Q]\rangle P'$ and $Q \xrightarrow{\tau}_{\mathrm{GC}} Q'$. By Lemma 7, $P \equiv (\nu\overrightarrow{p})(n[Q]|P')$, $\overrightarrow{p} \subset fn(n[Q])$. By applying the rules $(\nu)^*$, $(|_1)$, and (*amb*) backwards to

$$(\nu\overrightarrow{p})(n[Q]|P') \xrightarrow{\tau} (\nu\overrightarrow{p})(n[Q']|P'),$$

we arrive at $Q \xrightarrow{\tau} Q'$ which holds by induction since $Q \xrightarrow{\tau}_{\mathrm{GC}} Q'$.

It remains the case where $\alpha = M$ for a capability M: Let the reduction be of the form $P \xrightarrow{M}_{\mathrm{GC}} (\nu\overrightarrow{p})(P'|P'')$ (inferred by rule (*Trans Cap*)). Then $P \succ (\nu\overrightarrow{p})\langle M.P'\rangle P''$ and $fn(M) \cap \overrightarrow{p} = \emptyset$. By Lemma 7, $P \equiv (\nu\overrightarrow{p})(M.P'|P'')$ and $\overrightarrow{p} \subset fn(M.P')$. By applying rules $(\nu)^*$, $(|_1)$, and (*pre*) to

$$(\nu\overrightarrow{p})(M.P'|P'') \xrightarrow{M} (\nu\overrightarrow{p})(P'|P''),$$

we are done. □

Proof of Lemma 18. By induction over the structure of P.

The first equation is proved using the rather direct correspondence between the LTS rules in Fig. 4 and the corecursive definition of *next*. Here, it is crucial that the LTS has GSOS format: take the top-level process building operation of P. Then the set of rules that have this operation in the left-hand side of the conclusion defines the transition steps that P can perform. Now the corecursive definition of an operation just mimics the GSOS rule for the operation; in case there are several rules, their translations are just united.

The second equation (to be proved jointly with the first one in a parallel induction) is based on a correspondence between the rules in Fig. 3 and the corecursive definition of *harden*. Note that due to α-congruence, the set

$$\{([\![P_1']\!], [\![P_2']\!], p) \mid P \overset{\beta}{\succ} (\nu\overrightarrow{p})\langle P_1'\rangle P_2'\}$$

is closed under consistent renamings of the names $\overrightarrow{p}$. So is $harden(\beta, [\![P]\!])$—this is due to the explicit renaming at the only place where new names can be introduced into the third component of the hardening observation, namely in the last term of the definition of *harden* for restriction.

The third equation is obvious. □

Proof of Proposition 20. Follows easily from Lemma 18. □

Proof of Lemma 24. '*Only if*': Let $P \overset{\beta}{\succ} (\nu\overrightarrow{p})\langle P'\rangle P''$. By Lemma 9, $P \equiv (\nu\overrightarrow{p})(n[P'] \mid P'')$ and $n \notin \overrightarrow{p}$, i.e. $P \downarrow n$.

'*If*': Let $P \downarrow n$, i.e. $P \equiv (\nu\overrightarrow{p})(n[P'] \mid P'')$ for some $\overrightarrow{p}$, P', P'' such that $n \notin \overrightarrow{p}$. By rules ($\overline{\mathbf{enter}}\, h$) or ($\overline{\mathbf{open}}\, h$), respectively, we have $n[P'] \overset{\beta}{\succ} (\nu)\langle P'\rangle$. By rules ($|h_1$) and ($\nu h$), we obtain $(\nu\overrightarrow{p})(n[P'] \mid P'') \overset{\beta}{\succ} (\nu\overrightarrow{p})\langle P'\rangle P''$ as required. □

Proof of Lemma 25. For the first part, let $P \cong_a Q$ and let $P \downarrow n$. By Lemma 24, there is a concretion C such that $P \overset{\beta}{\succ} C$ for $\beta \in \{\overline{\mathbf{enter}}\, n, \overline{\mathbf{open}}\, n\}$. As $P \cong_a Q$, there exists a concretion C' such that $Q \overset{\beta}{\succ} C'$. By Lemma 24, $Q \downarrow n$.

For the second part, note that $P \Downarrow n$ means that there is a natural number m and a process P'' such that $P \xrightarrow{\tau}{}^{m} P''$ and $P'' \downarrow n$. The claim is then easily shown by induction on m, with the base case being just the first part of the lemma. □

Proof of Theorem 26. It suffices by Remark 23 to show that the structural congruence equations of Fig. 2 together constitute a relaxed action bisimulation up to context. Commutativity and associativity of parallel composition, interchange of restrictions ($(\nu n)(\nu m)P = (\nu m)(\nu n)P$), and the equations involving 0 are straightforward. Using the observation (proved by means of Lemma 9) that any hardening of $P|!P$ must come from a hardening of P (as is trivially the case for $!P$), one easily deals with the hardening part of the equation $!P \equiv P|!P$. Both cases of the transition part (i.e. the case that the transition was derived by rule (!) and the case that it was derived by rule ($\mathbf{enter}_!$)) are straightforward.

We treat the remaining two cases in more detail. We begin with scope extrusion, $(\nu n)(P|Q) \equiv P|(\nu n)Q$ ($n \notin fn(P)$).

Transitions: This is fairly straightforward, except that the definition of the restriction $(\overline{\nu n})C$ of a concretion C makes for a somewhat large number of case distinctions. We treat only an exemplary case: assume that a transition

$$(\nu n)(P \mid Q) \xrightarrow{\tau} (\nu n)(\nu\overrightarrow{p})(\nu\overrightarrow{q})(m[P'|Q']|P''|Q'')$$

has been derived by rules ($\mathbf{enter}_1$) and (ν) from $P \overset{\mathbf{enter}\ m}{\succ} (\nu\overrightarrow{p})\langle P'\rangle P''$ and $Q \overset{\overline{\mathbf{enter}}\ m}{\succ} (\nu\overrightarrow{q})\langle Q'\rangle Q''$, where $\overrightarrow{p} \cap \overrightarrow{q} = \emptyset$. Since $n \notin fn(P)$, $n \neq m$. Then $(\nu n)Q \overset{\overline{\mathbf{enter}}\ m}{\succ} (\overline{\nu n})(\nu\overrightarrow{q})\langle Q'\rangle Q''$. Here another case split occurs, according to whether Q' and Q'' contain the name n. The most complicated case is that $n \in fn(Q') \setminus fn(Q'')$, so that

$$(\overline{\nu n})(\nu\overrightarrow{q})\langle Q'\rangle Q'' = (\nu\overrightarrow{q})\langle(\nu n)Q'\rangle Q''.$$

By rule ($\mathbf{enter}_1$), we then have

$$P|(\nu n)Q \xrightarrow{\tau} (\nu\overrightarrow{p})(\nu\overrightarrow{q})(m[P'|(\nu n)Q']|P''|Q''),$$

with

$$(\nu n)(\nu\overrightarrow{p})(\nu\overrightarrow{q})(m[P'|Q']|P''|Q'') \equiv (\nu\overrightarrow{p})(\nu\overrightarrow{q})(m[P'|(\nu n)Q']|P''|Q'')$$

as required.

Free names: It is clear that $(\nu n)(P|Q)$ and $P|(\nu n)Q$ contain the same free names.

Hardenings: This is straightforward, because hardenings of parallel composites (unlike transitions) always come from one of the components.

The second remaining case is commutation of restriction with ambients, $(\nu m)n[P] \equiv n[(\nu m)P]$ $(n \neq m)$.

Transitions: If $(\nu m)n[P] \xrightarrow{\alpha} \bar{P}'$, then necessarily $\alpha = \tau$, $\bar{P}' = (\nu m)P'$, and $n[P] \xrightarrow{\tau} P'$. The latter transition arises by one of the rules (*amb*) and (**exit**). In the first case, we have $P' = n[Q]$ and $P \xrightarrow{\tau} Q$. By rules (ν) and (*amb*), $n[(\nu m)P] \xrightarrow{\tau} n[(\nu m)Q]$, and $(\nu m)n[Q] \equiv n[(\nu m)Q]$ as required. In the second case, $P' = (\nu \overrightarrow{p})(Q|n[Q'])$, and $P \overset{\textbf{exit}\ n}{\succ} (\nu \overrightarrow{p})\langle Q\rangle Q'$. Then $(\nu m)P \overset{\textbf{exit}\ n}{\succ} (\overline{\nu m})(\nu \overrightarrow{p})\langle Q\rangle Q'$. Again, we have a case split according to whether m occurs freely in Q and Q', respectively. We treat only the most complicated case, $m \in fn(Q')\backslash fn(Q)$. Then $(\overline{\nu m})(\nu \overrightarrow{p})\langle Q\rangle Q' = (\nu \overrightarrow{p})\langle Q\rangle(\nu m)Q'$, and hence $n[(\nu m)P] \xrightarrow{\tau} (\nu \overrightarrow{p})(Q|n[(\nu m)Q'])$, with the last process structurally congruent to $(\nu m)(\nu \overrightarrow{p})(Q|n[Q'])$ as required. Conversely, one shows quite similarly that $(\nu m)n[P]$ can simulate the transitions of $n[(\nu m)P]$.

Free names: It is clear that $(\nu m)n[P]$ and $n[(\nu m)P]$ have the same free names.

Hardenings: Hardenings of $(\nu m)n[P]$ are of the shape $(\nu m)n[P] \overset{\alpha}{\succ} (\overline{\nu m})C$, derived from hardenings $n[P] \overset{\alpha}{\succ} C$, where $m \notin fn(\alpha)$. The latter are derived by one of the first four hardening rules. In the case for the rule (**enter** h), we have $n[P] \overset{\textbf{enter}\ p}{\succ} (\nu)\langle n[Q]\rangle \mathbf{0}$, derived from $P \xrightarrow{\textbf{in}\ p} Q$. Then $(\nu m)P \xrightarrow{\textbf{in}\ p} (\nu m)Q$. Thus $n[(\nu m)P] \overset{\textbf{enter}\ p}{\succ} (\nu)\langle n[(\nu m)Q]\rangle \mathbf{0}$, with $(\nu)\langle n[(\nu m)Q]\rangle \mathbf{0} \equiv (\nu)\langle(\nu m)n[Q]\rangle \mathbf{0} = (\overline{\nu m})(\nu)\langle n[Q]\rangle \mathbf{0}$ as required. The case for rule (**exit** h) is analogous, and the other two cases are markedly simpler. Conversely, one checks in a quite similar way that $(\nu m)n[P]$ can simulate the hardenings of $n[(\nu m)P]$. □

Acknowledgements

The authors would like to thank Horst Reichel and Markus Roggenbach for collaboration on CoCasl, and Erwin R. Catesbeiana for volunteering his opinion on the CoCasl empty carrier problem.

References

[1] F. Bartels, Generalised coinduction, Math. Struct. Comput. Sci. 13 (2003) 321–348.
[2] M. Bidoit, P.D. Mosses. Casl User Manual, Lecture Notes in Computer Science, Vol. 2900, Springer, Berlin, 2004.
[3] L. Cardelli, A. Gordon, Mobile ambients, Theoret. Comput. Sci. 240 (2000) 177–213.
[4] L. Cardelli, A. Gordon, Ambient logic, Math. Struct. Comput. Sci. 2005, to appear.
[5] M. Gabbay, A. Pitts, A new approach to abstract syntax with variable binding, Formal Aspects Comput. 13 (2002) 341–363.
[6] A. Gordon, L. Cardelli, Equational properties of mobile ambients, Math. Struct. Comput. Sci. 13 (2003) 371–408.
[7] D. Hausmann, T. Mossakowski, L. Schröder, Iterative circular coinduction for CoCasl in Isabelle/HOL, in: M. Cerioli (Ed.), Fundamental Approaches to Software Engineering, Lecture Notes in Computer Science, Vol. 3442, Springer, Berlin, 2005, pp. 341–356.
[8] F. Honsell, M. Lenisa, U. Montanari, M. Pistore, Final semantics for the π-calculus, in: D. Gries, W.-P. de Roever (Eds.), Programming Concepts and Methods, Chapman & Hall, London, 1998, pp. 225–243.
[9] B. Klin, A coalgebraic approach to process equivalence and a coinduction principle for traces, in: J. Adámek, S. Milius (Eds.), Coalgebraic Methods in Computer Science, Electronic Notes in Theoretical Computer Science, Vol. 106, Elsevier, Amsterdam, 2004, pp. 201–218.
[10] F. Levi, D. Sangiorgi, Mobile safe ambients, J. ACM 25 (1) (2003) 1–69.
[11] M. Merro, M. Hennessy, Bisimulation congruences in safe ambients, ACM SIGPLAN Notices 37 (2002) 71–80.
[12] M. Merro, F.Z. Nardelli, Behavioural theory for mobile ambients, J. ACM 52 (6) (2005) 961–1023.
[13] R. Milner, D. Sangiorgi, Barbed bisimulation, in: W. Kuich (Ed.), Automata, Languages and Programming, Lecture Notes in Computer Science, Vol. 623, Springer, Berlin, 1992, pp. 685–695.
[14] T. Mossakowski, The heterogeneous tool set, Available under ⟨http://www.tzi.de/cofi/hets⟩, University of Bremen.
[15] T. Mossakowski, Heterogeneous specification and the heterogeneous tool set, Habilitation Thesis, University of Bremen, 2005.
[16] T. Mossakowski, L. Schröder, M. Roggenbach, H. Reichel, Algebraic-co-algebraic specification in CoCasl, J. Logic Algebraic Programming 67 (2006) 146–197.
[17] P.D. Mosses (Ed.), Casl Reference Manual, Lecture Notes in Computer Science, Vol. 2960, Springer, Berlin, 2004.
[18] D. Pattinson, Expressive logics for coalgebras via terminal sequence induction, Notre Dame J. Formal Logic 45 (2004) 19–33.
[19] J. Rothe, Behavioural equivalences for coalgebras, Ph.D. Thesis, University of Dresden, 2003.
[20] J. Rutten, Universal coalgebra: a theory of systems, Theoret. Comput. Sci. 249 (2000) 3–80.
[21] U. Schöpp, I. Stark, A dependent type theory with names and binding, in: J. Marcinkowski, A. Tarlecki (Eds.), Computer Science Logic, Lecture Notes in Computer Science, Vol. 3210, Springer, Berlin, 2004, pp. 235–249.

[22] L. Schröder, Expressivity of coalgebraic modal logic: the limits and beyond, in: V. Sassone (Ed.), Foundations of Software Science and Computation Structures, Lecture Notes in Computer Science, Vol. 3441, Springer, Berlin, 2005, pp. 440–454.
[23] B. Thomsen, A theory of higher order communicating systems, Inform. and Comput. 116 (1995) 38–57.
[24] D. Turi, G. Plotkin, Towards a mathematical operational semantics, in: Logic in Computer Science, IEEE Computer Society Press, 1997, pp. 280–291.
[25] M. Vigliotti, Reduction semantics for ambient calculi, Ph.D. Thesis, Imperial College, London, 2004.