

Towards a Coalgebraic Semantics of the Ambient Calculus

Daniel Hausmann, Till Mossakowski, and Lutz Schröder

BISS, Dept. of Computer Science, University of Bremen

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 work towards a unifying coalgebraic framework for the denotational semantics of mobile systems. The connection between the ambient calculus and a coalgebraic approach which uses an extension of labelled transition systems in the representation of the reduction relation is analyzed in more detail. The formal representation of this framework is cast in the algebraic-coalgebraic specification language COCASL.

Introduction

Coalgebra has in recent years gained importance as a framework which allows the modelling of reactive systems at an appropriate level of generality [14]. 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 [12]. 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 [10]; further work in similar directions is found e.g. in [6, 7]. Here, we present work leading towards a coalgebraic denotational semantics for mobile process calculi, in particular the ambient calculus [4]. 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.

An LTS semantics for the ambient calculus has been defined in [5]. This semantics involves the use of a *hardening* relation, which singles out active top-level processes to be involved in spatial reductions. The hardening relation inspires our design of a coalgebraic functorial signature for the ambient calculus. We

then give corecursive definitions of the process-building operations of a subset of the ambient calculus in the final coalgebra for this functor, and we show that the arising coalgebraic notion of bisimulation coincides with the one induced by the LTS semantics of [5]. As a framework supporting the formal specification of these concepts, we use the algebraic-coalgebraic language CoCASL [10].

The material is organized as follows. Section 1 provides a brief overview of CoCASL. An introduction to the ambient calculus is given in Section 2. Finally, our coalgebraic semantics for the ambient calculus is presented in Section 3.

1 CoCASL

The algebraic-coalgebraic specification language CoCASL has been introduced in [10] as an extension of the standard algebraic specification language CASL. For the basic CASL syntax, the reader is referred to [2, 11]. We briefly explain the CoCASL features relevant for the understanding of the present work.

A simple but typical CoCASL specification is shown in Fig. 1. This specification defines the final finitely branching labelled transition system (LTS) over a given set of labels, exploiting both algebraic and coalgebraic aspects of CoCASL.

```

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. 1. CoCASL specification of the final LTS

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. 1 gives rise to operators $\{-\} : State \rightarrow Set$ etc. called *constructors*, while the cotype declaration produces an operator $next : State \times Label \rightarrow Set$, 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 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 $Process ::= cont(hd1 :?Elem; next :?Process)$
 $| fork(hd2 :?Elem; left :?Process; right :?Process)$

produces a process type that can in each step either just advance one step ($next$) or fork ($left/right$). It is shown in [10] 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. 1 corresponds to coalgebras for $TX = Label \rightarrow \mathcal{P}_\omega(X)$, where \mathcal{P}_ω 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. 1: The keyword **cofree** may also be used to restrict the models of an entire specification, delimited as in Fig. 1 by curly brackets, to final models over a given model of the preceding specification — in the case of Fig. 1 over a given set of labels. This concept is dual to the CASL construct **free**, also appearing in Fig. 1, 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. 1 over a given set of states (and, irrelevantly, a given set of labels). (Subtle differences between **cofree** and **free** are discussed in [10]; this is not relevant for the understanding of the present work.)

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 $TX = Label \rightarrow \mathcal{P}_\omega(X)$, i.e. the final finitely branching LTS. This cotype, or process type, has been used in [10] in order to define a coalgebraic denotational semantics for CCS, exploiting the fact that final coalgebras admit corecursive definitions; e.g. the parallel operator may be defined in CoCASL by the corecursive equation

- $next(l, s1 \parallel s2) = power[- \parallel -](next(l, s1) * s2 \cup s1 * next(l, s1))$

(omitting the silent action), where $power[- \parallel -]$ has previously been defined as the image function of $- \parallel - : State \times State \rightarrow State$, and $A * s$ denotes the cartesian product $A \times \{s\}$. In this work, we pursue similar goals for the ambient calculus.

The form of corecursion used above, also called coiteration or coinductive definition, is a very simple one which is based directly on the definition of the final coalgebra: the corecursive equation essentially expresses that $next$ is the unique morphism from a coalgebra determined by the right hand side of the equation into the final coalgebra. 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 [4] 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.

$$\begin{array}{c}
 \overline{m[\mathbf{in} \ n.P|Q] \mid n[R] \longrightarrow n[m[P|Q]]R} \quad \overline{n[m[\mathbf{out} \ n.P|Q] \mid R] \longrightarrow m[P|Q] \mid n[R]} \\
 \\
 \overline{\mathbf{open} \ n.Q \mid n[R] \longrightarrow Q|R} \quad \frac{P \longrightarrow Q}{P|R \longrightarrow Q|R} \quad \frac{P \longrightarrow Q}{n[P] \longrightarrow n[Q]} \quad \frac{P \longrightarrow Q}{(\nu n)P \longrightarrow (\nu n)Q} \\
 \\
 \frac{P \equiv P' \quad P' \longrightarrow Q' \quad Q' \equiv Q}{P \longrightarrow Q}
 \end{array}$$

Fig. 2. The reduction relation of the ambient calculus

$$\begin{array}{l}
P|\mathbf{0} \equiv P \quad P|Q \equiv Q|P \quad (\nu m)(\nu n)P \equiv (\nu n)(\nu m)P \\
(\nu n)\mathbf{0} \equiv \mathbf{0} \quad (P|Q)|R \equiv P|(Q|R) \quad (\nu n)(P|Q) \equiv P|(\nu n)Q \text{ if } n \notin fn(P) \\
!\mathbf{0} \equiv \mathbf{0} \quad !P \equiv P!P \quad (\nu m)n[P] \equiv n[(\nu m)P] \text{ if } n \neq m
\end{array}$$

Fig. 3. The structural congruence of the ambient calculus

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. 2, 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. 3.

Example 1. The following typical example shows two possible chains of reductions of the process $m[\mathbf{in} \ n. \ \mathbf{out} \ n. \ \mathbf{0}] \mid n[\mathbf{open} \ m. \ \mathbf{0}]$:

$$\begin{aligned}
m[\mathbf{in} \ n. \ \mathbf{out} \ n. \ \mathbf{0}] \mid n[\mathbf{open} \ m. \ \mathbf{0}] &\longrightarrow n[m[\mathbf{out} \ n. \ \mathbf{0}] \mid \mathbf{open} \ m. \ \mathbf{0}] \\
&\longrightarrow m[\mathbf{0}] \mid n[\mathbf{open} \ m. \ \mathbf{0}]
\end{aligned}$$

$$\begin{aligned}
m[\mathbf{in} \ n. \ \mathbf{out} \ n. \ \mathbf{0}] \mid n[\mathbf{open} \ m. \ \mathbf{0}] &\longrightarrow n[m[\mathbf{out} \ n. \ \mathbf{0}] \mid \mathbf{open} \ m. \ \mathbf{0}] \\
&\longrightarrow n[\mathbf{out} \ n. \ \mathbf{0} \mid \mathbf{0}]
\end{aligned}$$

It is easy to see that the ambient calculus is non-deterministic and non-confluent (and due to the replication operation also potentially non-terminating).

3 A Coalgebraic Semantics for the Ambient Calculus

Following [4], we can equivalently define the reduction relation of the ambient calculus in terms of a labelled transition system 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 which 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. Besides making standard coalgebraic machinery available for the ambient calculus, this clarifies the observational aspects of the calculus. The corecursive definitions will be presented in CoCASL.

3.1 The LTS

We recall the definition of labelled transition systems:

Definition 2. A *labelled transition system* (LTS) is a triple (S, A, T) where S is a set of *states*, A is a set of *actions* and $T \subset S \times A \times S$ is the *transition relation*. For $P, Q \in S$ and $a \in A$, $(P, a, Q) \in T$ means that the system evolves by the action a from source P to target Q ; this is denoted in the form $P \xrightarrow{a} Q$.

In order to allow for a coalgebraic specification of the semantics, we design a variant of the LTS given in [4] in such a way that the conclusion of each inference rule of the LTS has the application of exactly one process-building operation of the ambient calculus on the left hand side of the conclusion (and no process building operations appear in the premises). This facilitates the subsequent corecursive definition of the operations. As indicated above, we need a hardening relation as in [5], extended by intermediate capabilities in the spirit of [8]:

As in [5], a *concretion* is an expression of the form: $(\nu \vec{p})\langle P \rangle Q$, where $P, Q \in \mathcal{AC}$ and $\vec{p} = \{p_1, \dots, p_n\}$. The process P is called the *prime*, the process Q is called the *residue*. The intuition is that a process, which may have many top-level processes, may harden to a concretion that singles out an active subprocess P , leaving behind the residue Q , where \vec{p} is the set of private names shared by P and Q .

$\frac{P \xrightarrow{\text{in } n} Q}{m[P] \xrightarrow{\text{enter } n} (\nu) \langle m[Q] \rangle \mathbf{0}}$	$\frac{}{n[P] \xrightarrow{\overline{\text{enter } n}} (\nu) \langle P \rangle \mathbf{0}}$
$\frac{P \xrightarrow{\text{out } n} Q}{m[P] \xrightarrow{\text{exit } n} (\nu) \langle m[Q] \rangle \mathbf{0}}$	$\frac{}{n[P] \xrightarrow{\overline{\text{open } n}} (\nu) \langle P \rangle \mathbf{0}}$
$\frac{\vec{p} \cap \text{fn}(Q) = \emptyset \quad P \xrightarrow{\alpha} (\nu \vec{p}) \langle P' \rangle P''}{P Q \xrightarrow{\alpha} (\nu \vec{p}) \langle P' \rangle P'' Q}$	$\frac{\vec{p} \cap \text{fn}(Q) = \emptyset \quad P \xrightarrow{\alpha} (\nu \vec{p}) \langle P' \rangle P''}{Q P \xrightarrow{\alpha} (\nu \vec{p}) \langle P' \rangle Q P''}$
$\frac{P \xrightarrow{\alpha} (\nu \vec{p}) \langle P' \rangle P''}{!P \xrightarrow{\alpha} (\nu \vec{p}) \langle P' \rangle P'' !P}$	$\frac{P \xrightarrow{\alpha} C \quad n \notin \text{fn}(\alpha)}{(\nu n)P \xrightarrow{\alpha} (\nu n)C}$

Fig. 4. The hardening relation

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 [8].

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. The hardening relation has the format $\succ_C^\alpha \mathcal{AC} \times HAction \times \mathcal{Y}$ for the set $HAction = \{\mathbf{enter}, \mathbf{enter}, \mathbf{exit}, \mathbf{open}\} \times \mathcal{N}$. The rules are given in Fig. 4, where $(\overline{\nu n})C$ for $C = (\nu \vec{p})\langle P' \rangle P''$ is defined as

$$(\overline{\nu n})C = \begin{cases} (\nu \vec{p})\langle P' \rangle (\nu n)P'' & \text{if } n \notin fn(P') \\ (\nu \vec{p})\langle m[(\nu n)P'''] \rangle P'' & \text{if } n \notin fn(P'') \text{ and } P' = m[P'''] \text{ (} n \neq m \text{)} \\ (\nu n, \vec{p})\langle P' \rangle P'' & \text{otherwise.} \end{cases}$$

The labelled transition system has the format $\xrightarrow{\alpha} \mathcal{AC} \times Action \times \mathcal{AC}$ for the set $Action = \{\tau\} \cup \{\mathbf{in}, \mathbf{out}, \mathbf{open}\} \times \mathcal{N}$ of transition labels. Its rules are displayed in Fig. 5. The rules imply that concretions in hardenings labelled **enter** or **exit** are always of the form $(\nu \vec{p})\langle n[P] \rangle Q$. Note that while we do not impose the full structural congruence on terms for purposes of the hardening and transition relations, we do assume that bound names are 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 suitably renaming bound names. Concerning the transition rule for exiting ambients, one can show that the premise implies $n \in fn(P)$, in particular $n \notin \vec{p}$.

Theorem 3. *The labelled transition system defined above is, up to structural congruence, sound and complete with respect to the reduction relation of the ambient calculus as recalled in Section 2. Formally: $P \xrightarrow{\tau} Q$ implies $P \longrightarrow Q$ (soundness) and $P \longrightarrow Q$ implies $\exists P'. P \xrightarrow{\tau} P' \wedge P' \equiv Q$ (completeness).*

Cardelli and Gordon present a different labelled transition system for the ambient calculus in [5]. We write $P \xrightarrow{\alpha}_{CG} Q$ 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 labelled transition system from [5]. (Unlike the system in [5], our system does not take input and output primitives into account; however, these features can easily be added to the theory.)

Theorem 4. *The labelled transition system defined above is sound and complete with respect to the labelled transition system of [5]: $P \xrightarrow{\alpha} Q$ implies $P \xrightarrow{\alpha}_{CG} Q$ (soundness) and $P \xrightarrow{\alpha}_{CG} Q$ implies $P \xrightarrow{\alpha} Q$ (completeness).*

3.2 Coalgebraic Semantics

The mobility aspects of the labelled transition system 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

$$\begin{array}{c}
\frac{}{M.P \xrightarrow{M} P} \quad \frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q} \quad \frac{P \xrightarrow{\alpha} P'}{Q|P \xrightarrow{\alpha} Q|P'} \quad \frac{P \xrightarrow{\alpha} P'}{P! \xrightarrow{\alpha} P'|P!} \\
\\
\frac{P \xrightarrow{\text{exit}^n} (\nu \vec{p}) \langle P' \rangle P''}{n[P] \xrightarrow{\tau} (\nu \vec{p}) (n[P''] | P')} \quad \frac{P \xrightarrow{\tau} Q}{n[P] \xrightarrow{\tau} n[Q]} \quad \frac{P \xrightarrow{\alpha} Q \quad n \notin \text{fn}(\alpha)}{(\nu n)P \xrightarrow{\alpha} (\nu n)Q} \\
\\
\frac{P \xrightarrow{\text{open}^n} P' \quad Q \xrightarrow{\overline{\text{open}}^n} (\nu \vec{q}) \langle Q' \rangle Q'' \quad \vec{q} \cap \text{fn}(P) = \emptyset}{P|Q \xrightarrow{\tau} (\nu \vec{q}) (P'|Q'|Q'')} \\
\\
\frac{Q \xrightarrow{\text{open}^n} Q' \quad P \xrightarrow{\overline{\text{open}}^n} (\nu \vec{p}) \langle P' \rangle P'' \quad \vec{p} \cap \text{fn}(Q) = \emptyset}{P|Q \xrightarrow{\tau} (\nu \vec{p}) (P'|P''|Q')} \\
\\
\frac{P \xrightarrow{\text{enter}^n} (\nu \vec{p}) \langle P' \rangle P'' \quad Q \xrightarrow{\overline{\text{enter}}^n} (\nu \vec{q}) \langle Q' \rangle Q'' \quad \vec{p} \cap \vec{q} = \emptyset}{P|Q \xrightarrow{\tau} (\nu \vec{p}) (\nu \vec{q}) (n[P'|Q'] | P''|Q'')} \\
\\
\frac{Q \xrightarrow{\text{enter}^n} (\nu \vec{q}) \langle Q' \rangle Q'' \quad P \xrightarrow{\overline{\text{enter}}^n} (\nu \vec{p}) \langle P' \rangle P'' \quad \vec{p} \cap \vec{q} = \emptyset}{P|Q \xrightarrow{\tau} (\nu \vec{p}) (\nu \vec{q}) (n[P'|Q'] | P''|Q'')}
\end{array}$$

Fig. 5. The transition relation

the concretions to which a process hardens. An further important aspect is the handling of name creation, which however poses certain technical problems in the coalgebraic setting. For the time being, we therefore give a coalgebraic semantics of a reduced calculus without name restriction (comparable in this respect e.g. to the Basic Sail calculus [13]). The combination of reduction and hardening suggests using the functor

$$AC = \lambda X. (\text{Action} \rightarrow \mathcal{P}_\omega(X)) \times (H\text{Action} \rightarrow \mathcal{P}_\omega(X \times X)).$$

Coalgebras $(\mathcal{A}, \psi : \mathcal{A} \rightarrow AC(\mathcal{A}))$ for this functor have a function $\psi = \langle \text{next}, \text{harden} \rangle$ as structure. The first part of this function ($\text{next} : \mathcal{A} \rightarrow (\text{Action} \rightarrow \mathcal{P}_\omega(\mathcal{A}))$) assigns to each state of the coalgebra a function which maps each transition label to the set of the corresponding successor states. The second function ($\text{harden} : \mathcal{A} \rightarrow (H\text{Action} \rightarrow \mathcal{P}_\omega(\mathcal{A} \times \mathcal{A}))$) assigns to each state a function which maps hardening labels to the set of the corresponding concretions of the state.

It should be noted that the crucial difference between AC and the standard functor $\mathcal{P}_\omega(A \times _)$ for LTS lies in the fact that the hardening part is essentially of the type $\lambda X. \mathcal{P}_\omega(X \times X)$; here, the peculiarity is captured that a process splits into two parts 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’.

A CoCASL specification of the final coalgebra of the functor AC is shown in Fig. 6. The specification is based on a specification of finite sets, which is

parametric in the type of elements and which needs to be instantiated with states and concretions. The cotype *State* is the mentioned final *AC*-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 *AC* needs to be split up into various datatype definitions; this makes for a somewhat more verbose, but also clearer and more readable specification style.

```

spec SET [sort Elem] =
free {
  type Set[Elem] ::= {} | {_}(Elem) | _ ∪ _ (Set[Elem]; Set[Elem])
  op _ ∪ _ : Set[Elem] × Set[Elem] → Set[Elem],
      assoc, comm, idem, unit { } }
then
  pred _ ∈ _ : Elem × Set[Elem]
  op _ − _ : Set[Elem] × Elem → Set[Elem]
  ... %% recursive Definitions of ∈, −

spec ACDOMAIN[sort Name] = SET [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)
  free type HAction ::= haction(HCap; Name)
  free type Concretion ::= conc(State; State)
  then SET [sort State] and SET [sort Concretion]
  then cotype State ::= (next : Action → Set[State];
                          harden : HAction → Set[Concretion])
}

```

Fig. 6. CoCASL specification of the semantic domain for the ambient calculus

The process building operations of the ambient calculus can then be defined as functions into the semantic domain *State* by mutual corecursion. The corecursive definitions are clear from the corresponding inference rules for the hardening relation and the transition relation as shown in Figs. 4–5. The CoCASL specifications of the operations are shown in Figs. 7–11. (The operator declarations should be considered redeclarations, following an initial declaration of all operators which is necessary for mutually corecursive definitions due to CASL’s linear visibility principle. Moreover, the label *in* would have to be introduced via a display annotation, since **in** is a reserved word.)

In the description of the successor and concretion sets, the definitions below use explicit equivalences for elementhood in the successor sets rather than equa-

tions involving application of the powerset functor to the defined functions. One can also apply the latter style in CoCASL, making extensive use of parametrized specifications as e.g. in the CCS semantics of [10]. This style is actually preferable, but would require rather more infrastructure than the available space permits to present here.

```

op   zero : State
vars a : Action; b : HAction
•     next(a, zero) = {}
•     harden(b, zero) = {}

```

Fig. 7. CoCASL specification of the nil process

```

op   cap : Cap × Name × State → State
vars a : Action; b : HAction; c : Cap; n : Name; p : State
•     next(a, cap(c, n, p)) = {p} when a = action(c, n) else {}
•     harden(b, cap(c, n, p)) = {}

```

Fig. 8. CoCASL specification of capability prefixing

Remark 5. A word of explanation is in order as to why the equations in Figs. 7–11 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 being defined. According to the results of [16], 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

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

where T is the free monad (i.e. the term algebra functor) over Σ (cf. also [1]). The appearance of T in the target offers the possibility of using composite terms, as required. The semantic function $h : \Sigma S \rightarrow S$, 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

$$\zeta \circ h = Bh^* \circ \rho_S \circ \Sigma(id, \zeta), \quad (*)$$

op	$-- \parallel -- : State \times State \rightarrow State$
vars	$a : Action; b : HAction; n : Name; p, q, r : State; c : Concretion$
•	$r \in next(a, p \parallel q) \Leftrightarrow$ $(a = \tau \wedge \exists p', q', q'' : State \bullet$ $\quad r = p' \parallel q' \parallel q'' \wedge p' \in next(action(open, n), p) \wedge$ $\quad\quad\quad conc(q', q'') \in harden(haction(coopen, n), q)) \vee$ $(a = \tau \wedge \exists q', p', p'' : State \bullet$ $\quad r = p' \parallel p'' \parallel q' \wedge q' \in next(action(open, n), q) \wedge$ $\quad\quad\quad conc(p', p'') \in harden(haction(coopen, n), p)) \vee$ $(a = \tau \wedge \exists p', p'', q', q'' : State \bullet$ $\quad r = amb(n, p' \parallel q') \parallel p'' \parallel q'' \wedge$ $\quad\quad\quad conc(p', p'') \in harden(haction(enter, n), p) \wedge$ $\quad\quad\quad conc(q', q'') \in harden(haction(coenter, n), q)) \vee$ $(a = \tau \wedge \exists p', p'', q', q'' : State \bullet$ $\quad r = amb(n, p' \parallel q') \parallel p'' \parallel q'' \wedge$ $\quad\quad\quad conc(p', p'') \in harden(haction(coenter, n), p) \wedge$ $\quad\quad\quad conc(q', q'') \in harden(haction(enter, n), q)) \vee$ $(\exists p' : State \bullet r = p' \parallel q \wedge p' \in next(a, p)) \vee$ $(\exists q' : State \bullet r = p \parallel q' \wedge q' \in next(a, q))$
•	$c \in harden(b, (p \parallel q)) \Leftrightarrow$ $(\exists p', p'' : State \bullet c = conc(p', p'' \parallel q) \wedge conc(p', p'') \in harden(b, p)) \vee$ $(\exists q', q'' : State \bullet c = conc(q', p \parallel q'') \wedge conc(q', q'') \in harden(b, q))$

Fig. 9. CoCASL specification of parallel composition

op	$amb : Name \times State \rightarrow State$
vars	$a : Action; b : HAction; cap : Cap; m, n : Name; p, q : State; c : Concretion$
•	$q \in next(\tau, amb(n, p)) \Leftrightarrow$ $(\exists p', p'' : State \bullet$ $\quad q = amb(n, p'') \parallel p' \wedge$ $\quad\quad\quad conc(p', p'') \in harden(haction(exit, n), p)) \vee$ $(\exists p' : State \bullet q = amb(n, p') \wedge p' \in next(\tau, p))$
•	$next(action(cap, n), amb(m, p)) = \{\}$
•	$c \in harden(haction(enter, n), amb(m, p)) \Leftrightarrow$ $(\exists p' : State \bullet c = conc(amb(m, p'), zero) \wedge$ $\quad\quad\quad p' \in next(action(in, n), p))$
•	$harden(haction(coenter, n), (amb(n, p))) = \{conc(p, zero)\}$
•	$c \in harden(haction(exit, n), amb(m, p)) \Leftrightarrow$ $(\exists p' : State \bullet c = conc(amb(m, p'), zero) \wedge$ $\quad\quad\quad p' \in next(action(out, n), p))$
•	$harden(haction(coopen, n), (amb(n, p))) = \{conc(p, zero)\}$

Fig. 10. CoCASL specification of the ambient operator

<p>op $rep : State \rightarrow State$</p> <p>vars $a : Action; b : HAction; p, q : State; c : Concretion$</p> <ul style="list-style-type: none"> • $q \in next(a, rep(p)) \Leftrightarrow$ $(\exists p' : State \bullet q = p' \parallel rep(p) \wedge p' \in next(a, p))$ • $c \in harden(b, rep(p)) \Leftrightarrow$ $(\exists p', p'' : State \bullet c = conc(p', p'' \parallel rep(p)) \wedge$ $conc(p', p'') \in harden(b, p))$

Fig. 11. CoCASL specification of replication

where $h^* : TS \rightarrow S$ is the T -algebra determined by h . If the semantic function h is omitted from the notation, as done above, then equation (*) 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 $|S| \geq \kappa$, 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 Figs. 4–5 is tedious but straightforward. The point is essentially that the rules adhere to similar restrictions as standard GSOS rules for the definition of labelled transition systems, 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 [16]).

Generally, GSOS semantics is *compositional*, i.e. the interpretation of composite terms is recursively derived from that of single operations [16]. This does not, incidentally, contradict the previously diagnosed impossibility of a compositional LTS semantics for the ambient calculus [17], since our semantic domain is more than just an LTS.

Remark 6. The reason that restriction is currently omitted in the specification is that it is as yet unclear how to model the sharing of private names in concretions: a concretion needs to contain information about the potential future interaction between the prime and the residue, but on the other hand should not leak information about the shared bound names.

We explicitly record the agreement between the coalgebraic specification and the LTS semantics given above:

Lemma 7. *The CoCASL specification is sound and complete with respect to the LTS defined in Section 3.1: for ambient calculus terms P and P' not involving restriction, $P \xrightarrow{\alpha} P'$ iff $P' \in next(\alpha, P)$ follows from the corecursive equations (omitting an obvious translation between ambient calculus terms and their representation in CoCASL). Furthermore, the LTS and the CoCASL specification*

agree w.r.t. hardening: $P \stackrel{\beta}{\succ} (\nu)\langle P'_1 \rangle P'_2$ iff $\text{conc}(P'_1, P'_2) \in \text{harden}(\beta, P)$ follows from the corecursive equations (again for terms without restriction, and omitting an obvious translation of fixed finite sets).

For the reduced calculus, the notion of bisimulation arising from the coalgebraic modelling can be brought into agreement with a natural notion of behavioral indistinguishability:

Definition 8 (Ambient bisimulation). A symmetric relation \simeq on ambient calculus processes without name restriction is called an *ambient bisimulation* if for any two processes P, Q such that $P \simeq Q$ the following hold:

1. $P \xrightarrow{\alpha} P' \Rightarrow (\exists Q'. Q \xrightarrow{\alpha} Q' \wedge P' \simeq Q')$ for any $\alpha \in \text{Act}$.
2. $P \stackrel{\beta}{\succ} (\nu)\langle P'_1 \rangle P'_2 \Rightarrow (\exists Q'_1, Q'_2. Q \stackrel{\beta}{\succ} (\nu)\langle Q'_1 \rangle Q'_2 \wedge P'_1 \simeq Q'_1 \wedge P'_2 \simeq Q'_2)$ for each $\beta \in \text{HAction}$.

If $P \simeq Q$ for some ambient bisimulation \simeq , then P and Q are called *ambient bisimilar*.

Recall that for any endofunctor G , a binary relation R between two G -coalgebras E and F is called a *bisimulation* if there exists a G -coalgebra structure on R that makes the projection functions $\pi_1 : R \rightarrow E$ and $\pi_2 : R \rightarrow F$ into coalgebra homomorphisms. If for two elements $e \in E$ and $f \in F$ and a bisimulation R it holds that $e R f$, then e and f are said to be *bisimilar*. In the final coalgebra, bisimilarity is equality.

From the preceding lemma, the following is easily shown:

Theorem 9. *Ambient bisimulation and coalgebraic bisimulation on AC-coalgebras coincide; formally: for $P, Q \in \mathcal{AC}$ not involving restriction, $P \simeq Q$ iff P and Q denote the same element of the final AC-coalgebra, i.e. iff $P = Q$ follows from the above CoCASL specification.*

4 Conclusion

We have described a transition semantics for the ambient calculus that correctly captures the ambient calculus in the sense that its reductions coincide with the reduction of the ambient calculus. Similar results have been obtained in [8] for the safe ambient calculus with passwords, and in [9] for the ambient calculus itself. In both cases, however, labelled transition systems are used which mix processes and concretions, while our system separates the two types of entities by keeping the reduction relation and the hardening relation apart. A consequence is that our semantics fits into a coalgebraic framework.

The coalgebraic treatment of the transition semantics (using the specification language CoCASL, or a corresponding functor) exhibits more structure than labelled transition systems. The coalgebraic structure is based on two kinds of observations: one can observe firstly the successor states in the 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, being 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. The coalgebraic treatment of private names, i.e. the security aspect of the ambient calculus, had to be left open for the time being; this problem is hoped to be resolved in future work. One should note that the first step in this direction has already been taken in the shape of our transition semantics, which does include name restriction: the transition rules adhere to a generalized GSOS format, which is an important precondition for a corecursive formulation.

The corecursive definition of process building operations implies the possibility to prove algebraic laws about the ambient calculus using coinduction, based on a notion of bisimulation arising from the coalgebraic semantics. An open problem that remains in this respect is the relation of this bisimilarity to the contextual equivalence of ambients [4] and to the notion of reduction barbed congruence [9].

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. [15] and references therein); further work will include the investigation of this logic in particular in relation to ambient logic [3]. In advance, we note that the hardening part of our behaviour functor will naturally give rise to a binary modality which appears also in ambient logic.

References

1. F. Bartels, *Generalised coinduction*, Math. Struct. Comput. Sci. **13** (2003), 321–348.
2. M. Bidoit and P. D. Mosses, *CASL user manual*, LNCS, vol. 2900, Springer, 2004.
3. L. Cardelli and A. Gordon, *Ambient logic*, Math. Struct. Comput. Sci., to appear.
4. ———, *Mobile ambients*, Theoret. Comput. Sci. **240** (2000), 177–213.
5. A. Gordon and L. Cardelli, *Equational properties of mobile ambients*, Math. Struct. Comput. Sci. **13** (2003), 371–408.

6. F. Honsell, M. Lenisa, U. Montanari, and M. Pistore, *Final semantics for the π -calculus*, Programming Concepts and Methods, Chapman & Hall, 1998, pp. 225–243.
7. B. Klin, *A coalgebraic approach to process equivalence and a coinduction principle for traces*, Coalgebraic Methods in Computer Science, ENTCS, vol. 106, Elsevier, 2004, pp. 201–218.
8. M. Merro and M. Hennessy, *Bisimulation congruences in safe ambients*, ACM SIGPLAN Notices **37** (2002), 71–80.
9. M. Merro and F. Zappa Nardelli, *Behavioural theory for mobile ambients*, Tech. Report RR-5375, INRIA, 2004.
10. T. Mossakowski, L. Schröder, M. Roggenbach, and H. Reichel, *Algebraic-coalgebraic specification in CoCASL*, J. Logic Algebraic Programming, to appear.
11. P. D. Mosses (ed.), *CASL reference manual*, LNCS, vol. 2960, Springer, 2004.
12. D. Pattinson, *Expressive logics for coalgebras via terminal sequence induction*, Notre Dame J. Formal Logic **45** (2004), 19–33.
13. D. Pattinson and M. Wirsing, *Making components move: a separation of concerns approach*, Formal Methods for Components and Objects (FMCO 02), LNCS, vol. 2852, Springer, 2003, pp. 487–507.
14. J. Rutten, *Universal coalgebra: a theory of systems*, Theoret. Comput. Sci. **249** (2000), 3–80.
15. L. Schröder, *Expressivity of coalgebraic modal logic: The limits and beyond*, Foundations of Software Science And Computation Structures, LNCS, vol. 3441, Springer, 2005, pp. 440–454.
16. D. Turi and G. Plotkin, *Towards a mathematical operational semantics*, Logic in Computer Science, IEEE Computer Society Press, 1997, pp. 280–291.
17. M. Vigliotti, *Reduction semantics for ambient calculi*, Ph.D. thesis, Imperial College, London, 2004.