Model Checking Biological Systems described using Ambient Calculus*

Radu Mardare, Corrado Priami, Paola Quaglia, and Oleksandr Vagin

Dipartimento di Informatica e Telecomunicazioni, Università di Trento, Italy

Abstract. We propose a way of performing model checking analysis for biological systems. The technics were developed for a CTL* logic built upon Ambient Calculus.
We introduce *labeled syntax trees* for ambient processes and use them as possible worlds in a Kripke structure developed for a propositional branching temporal logic. The accessibility relation over labeled syntax trees is generated by the reduction over corresponding Ambient Calculus processes.
Providing the algorithms for calculating the accessibility relation between states, we open the perspective of using model checking algorithms developed for temporal logics in analyzing any phenomena described in Ambient Calculus.

1 Introduction

Ambient Calculus [13] is a useful tool to construct mathematical models for complex systems because of its facilities in expressing hierarchies of locations and their mobility. At the same time properties as "the protein has split", or "there is a path of computation where the complexAB precedes the proteinA" are not expressible inside the calculus. Only a logic built on top of it can handle such properties.

Modal logics and especially temporal logics have emerged in many domains as a good compromise between expressiveness and abstraction. Many of them support useful computational applications as model checking. For the particular case of temporal logics, these technics were developed up to the construction of some tools able to perform such analysis (see, e.g. SMV [4], NuSMV [2], HyTech [1], VIS [5]).

This paper presents a propositional branching temporal logic for Ambient Calculus that is the representative calculus for the paradigm of calculi expressing hierarchies of locations and their mobility. We believe that the same sort of logic can be constructed for other calculi in this paradigm like, e.g., BioAmbients Calculus [7], or Brane Calculi [9].

The main feature of our logic is that the final state of any computation can be reconstructed by just having information about the initial state and the history

* Work partially supported by the FET project IST-2001-32072 DEGAS under the pro-active initiative on Global Computing.

of the computation. The spatial structure of a state is fully described by a set of atomical propositions, while the possible states are described using, in addition, a temporal modality. In this respect our approach is different from those used in Ambient Logic [12, 11], or Spatial Logic [10], giving us the advantages of simplicity and expressivity that a CTL* logic have w.r.t. the cited modal logics.

The rest of the paper is organized as follows. We first present a couple of simple case studies coming from biology. They are used to comment on the advantages of applying temporal logics to the Ambient Calculus specification of phenomena related to life sciences. Section 3 introduces the theoretical underpinning of our logic: *labeled syntax trees*. In Section 4 we define a branching temporal logic for Ambient Calculus, and show how to run simple reachability properties on our case studies. The final section concludes the presentation with a description of an implementation of our logic. The platform actually consists in the development of a suitable interface to NuSMV [2].

2 Case studies from biology

The advantage of using a temporal logic is relevant in the representation of biological phenomena because it gives us the power to predict over the future. Consider the model of *the trimetric GTP binding proteins (G-proteins)* that plays an important role in the signal transduction pathway for numerous hormones and neurotransmitters [3, 6]. It consists of five processes: a regulatory molecule RM, a receptor R, and three domains that are bound together composing the protein α, β and γ. Data sent by RM to R determine a communication between the receptor R and the protein that causes the breakage of the boundary of α, β and γ. We ca express this in Ambient Calculus by the following specification:

$$RM \stackrel{def}{=} open\ n.RM,\ R \stackrel{def}{=} n[\langle GTP\rangle|R],$$
$$Protein \stackrel{def}{=} (GDP)(\alpha|\beta|\gamma), \text{ where } GDP \text{ is a name that appear in } \alpha \text{ only,}$$
bounded by the input prefix
$$RM|R|Protein \equiv open\ n.RM \mid n[\langle GTP\rangle|R] \mid (GDP)(\alpha|\beta|\gamma) \rightarrow$$
$$RM \mid R \mid \langle GTP\rangle \mid (GDP)(\alpha|\beta|\gamma) \rightarrow$$
$$RM|R|(\alpha|\beta|\gamma)(GDP/GTP) \rightarrow$$
$$RM|R|(\alpha)(GDP/GTP)|\beta|\gamma$$

where we denoted by $(\alpha)(GDP/GTP)$ the process obtained by substituting GDP with GTP inside α.

W.r.t the above example we are interested to express properties like, e.g., *for all possible future paths, sometime in the future, we will have the interaction that will generate the split of the protein*. One could also want to express that the protein will not be split before the interaction between R and RM will be performed (*a property will not be satisfied until an other one will be*). Both properties above are examples of 'temporal' properties which cannot be expressed using other modal logics.

Consider now the interaction between a Virus and a Macrophage. The Macrophage is recognizing the Virus by its characteristics. Once it is recognized, the Virus is

moved by Macrophage inside itself were it is destroyed. We decided to describe the Macrophage as an ambient named n that contains a process $Digest$ able to destroy the virus; the virus is an ambient k' that contains inside a process $Infect$. The Macrophage recognizes the virus by the name k' and by its structure (i.e. Macrophage knows the names k, k'' that define the structure of the virus). Using these information, Macrophage manages to put in parallel the processes $Infect$ and $Digest$ and in this way annihilates the action of the virus. We can describe this action in Ambient Calculus in a way similar with the description of the action of a firewall:

$$Macrophage \stackrel{def}{=} n[k[out\ n.in\ k'.in\ n.0]|open\ k'.open\ k''.Digest]$$
$$Virus \stackrel{def}{=} k'[open\ k.k''[Infect]]$$
$$Virus|Macrophage \equiv$$
$$k'[open\ k.k''[Infect]]|n[k[out\ n.in\ k'.in\ n.0]|open\ k'.open\ k''.Digest]$$
$$\rightarrow^* k'[open\ k.k''[Infect]]|k[in\ k'.in\ n.0]|n[open\ k'.open\ k''.Digest]$$
$$\rightarrow^* k'[open\ k.k''[Infect]|k[in\ n.0]]|n[open\ k'.open\ k''.Digest]$$
$$\rightarrow^* k'[k''[Infect]|in\ n.0]|n[open\ k'.open\ k''.Digest]$$
$$\rightarrow^* n[k'[k''[Infect]]|open\ k'.open\ k''.Digest]$$
$$\rightarrow^* n[k''[Infect]|open\ k''.Digest]$$
$$\rightarrow^* n[Infect|Digest]$$

For this situation, we can be interested if our system succeeds, in all possible time paths, to achieve the state where the processes $Infect$ and $Digest$ are in parallel inside the ambient n (that represent the Macrophage), such that the virus to be annihilated. If our system succeeds to do this, we can say that is an appropriate one, otherwise we have to reconsider our approach. Such properties, we will argue further, can be naturally expressed in a temporal logic.

Another reason for using temporal logics to model Ambient Calculus is the possibility of performing model checking for our calculus, by reusing some software already developed for these logics such as SMV [4], NuSMV [2], HyTech [1], VIS [5].

3 The construction of the labeled syntax trees

In order to define the temporal logic, we reorganize the spatio-temporal information contained by an ambient process. This will be done by defining a special labelling function for the syntax trees of Ambient Calculus.

A syntax tree $S = (S, \rightarrow_S)$ for a process is a graph with $S = \Pi \cup \Gamma \cup \Omega = (\Pi_P \cup \Pi_A) \cup \Gamma \cup \Omega$ where

Π is a set that contain all the unspecified process nodes (hereafter atomical processes[1] and collected in the subset Π_P) and the ambient nodes (collected in the subset Π_A);

[1] We use these to denote unspecified processes found inside an ambient process; this is a necessary requirement in developing model checking for Ambient Calculus because we have to recognize and distinguish, over time, unspecified processes inside the target process. For instance P is an unspecified process in $n[in\ m.P]$

Γ is the set of capability nodes (we include here the input nodes and the nodes of variables over capabilities as well); and

Ω is the set of syntactical operator nodes (this set contains the parallel operators | and the prefix operators, $\bullet$). We identify the subset $\Omega' = \{\bullet_1 \in \Omega \mid \bullet_1 \rightarrow_S \mid \} \subseteq \Omega$ of the prefix nodes that are immediately followed in the syntax tree by the parallel operator because they play an important role in the spatial structure of the ambient process [2].

We consider also the possibility of having circular branches in our trees, when recursive definitions are involved. All the further discussion is including these cases as well.

The intuition behind the construction of a labeled syntax tree is to associate to each node of the syntax tree some labeles by two functions: *id* that gives to each node an identity, and *sp* that registers the spatial position of the node.

The identity function *id* associates a label (urelement or $\emptyset$):

1. to each unspecified process and to each ambient; this label will identify the node and will help us further to distinguish between processes that have the same name
2. to each capability, the identity of the process in front of which this capability is placed
3. $\emptyset$, to each syntactical node

The spatial function *sp* associates:

1. to each ambient the set of the identities of its children[3], while to unspecified processes associates the *id*-label.
2. to each capability, a natural number that counts the position of this capability in the chain of capabilities (if any) belonging to the same process
3. to each syntactical node the spatial function associates 0, except for the nodes in Ω' to which the function *sp* will associate the set of identities of the processes connected by the main parallel operator in the compound process that this point is prefixing. For example in the situation $c.(P|Q)$, $sp(\bullet) = \{id(P), id(Q)\}$.

We recall here some basic definitions of Set Theory and Graph Theory that are needed to formally define the functions *id* and *sp* above.

We choose to work inside Zermelo-Fraenkel system of Set Theory ZFC with the Foundation Axiom (FA), as being a fertile field that offers many tools for analyzing structures, as argued in [8]. This approach allows us to describe the spatial structure of ambient processes as equations in set theory, each such equation being then used as atomical proposition in our logic. In this way we will not

[2] These point operators are those that connect a capability with a process formed by a parallel composition of other processes bounded together by brackets, hereafter *complex processes*, as in $c.(P|Q)$

[3] We use the terms *parent* and *child* about processes, meaning the immediate parent and immediate child in Ambient Calculus processes.

use a modality in describing the hierarchy of locations, but only in describing the evolution of the hierarchy in time. Hereafter, we assume a class $\mho$ of urelements, set-theoretical entities which are not sets (they do not have elements) but can be elements of sets. The urelements together with the empty set $\emptyset$ will generate all the sets we will work with (sometimes sets of sets).

Definition 1. *A set a is* transitive *if all the elements of a set b, which is an element of a, also belong to a: $\forall b \in a$ if $c \in b$ then $c \in a$.*
The transitive closure *of a, denoted by $TC(a)$ is the smallest transitive set including a. The existence of $TC(a)$ could be justified as follows:*
$TC(a) = \cup\{a, \cup a, \cup\cup a, ...\}$

Definition 2. *The* support of a set *a, denoted by $supp(a)$ is $TC(a) \cap \mho$. The elements of $supp(a)$ are the urelements that are* somehow involved *in a.*

Definition 3. *If $a \subseteq \mho$ then $V(a) \stackrel{def}{=} \{b \mid b$ is a set and $supp(b) \subseteq a\}$. $V(a)$ is the class of all sets in which the only urelements that are somehow involved are the urelements of a.*

Definition 4. *Let $S_P = (S, \rightarrow_S)$ be the syntax tree associated with the ambient process P. We call* the structure graph *associated with P, the graph obtained by restricting the edge relation of the syntax tree to $\Pi \cup \Omega'$, i.e. the graph $T_P = (\Pi \cup \Omega', \rightarrow_T)$ defined by:*
for $n, m \in \Pi \cup \Omega'$ we have $n \rightarrow_T m$ iff $n \rightarrow_S^* m$ and $\nexists p \in \Pi \cup \Omega'$ such that $n \rightarrow_S^* p \rightarrow_S^* m$

Intuitively, the structure graph of a process is obtained by restricting the edge relation of its syntax tree to Π.

Definition 5. *A* decoration *of a graph $G = (G, \rightarrow_G)$ is an injective function $e : G \rightarrow V(\mho) \cup \mho$ such that for all $a \in G$ we have:*

- *if $\nexists b \in G$ such that $a \rightarrow_G b$ then $e(a) \in \mho$*
- *if $\exists b \in G$ such that $a \rightarrow_G b$ then $e(a) = \{e(b) |$ for all b such that $a \rightarrow_G b\}$.*

We now introduce a set of auxiliary functions that are the building blocks for *id* and *sp* (for the application of these and the following definitions see the Appendix).

Definition 6. *Let the next functions be defined on the subsets of nodes of the syntax tree $(S, \rightarrow)$ as follows:*

- *Let $sp_\Pi : \Pi \cup \Omega' \rightarrow V(\mho) \cup \mho$ be a decoration of the structure graph associated with our syntax tree.*
- *Let $id_\Pi : \Pi \rightarrow \mho$ be an injective function such that $id_\Pi(P) = sp_\Pi(P)$ for all $P \in \Pi_P$. Consider $U_P \stackrel{def}{=} id_\Pi(\Pi_P) \subset \mho$, $U_A \stackrel{def}{=} id_\Pi(\Pi_A) \subset \mho$*

– *Let* $sp_\Omega : \Omega \to \mho \cup V(\mho) \cup N$ *defined by (*N *is the class of natural numbers)*

$$sp_\Omega(s) = \begin{cases} sp_\Pi(s) \text{ iff } s \in \Omega' \\ 0 \qquad \text{iff } s \in \Omega \setminus \Omega' \end{cases}$$

Consider $O \stackrel{def}{=} sp_\Omega(\Omega') \subset V(\mho)$

– *Let* $id_\Omega : \Omega \to V(\mho) \cup \mho$ *defined by*

$$id_\Omega(s) = \emptyset$$

– *Let* $sp_\Gamma : \Gamma \to N$ *such that*

$$sp_\Gamma(c) = \begin{cases} 1 \qquad \text{iff } | \to \bullet \to c \text{ or } n \to \bullet \to c \text{ with } n \in \Pi \\ k+1 \text{ iff } \bullet_1 \to \bullet_2 \to c \text{ and } \bullet_1 \to c' \in \Gamma \text{ with } sp_\Gamma(c') = k \end{cases}$$

– *Let* $id_\Gamma : \Gamma \to V(\mho) \cup \mho$ *defined for* $c \in \Gamma$ *such that* $\bullet_c \to c$ *by*

$$id_\Gamma(c) = \begin{cases} id_\Pi(n) \text{ iff } \bullet_c \to n \text{ with } n \in \Pi \\ id_\Gamma(c') \text{ iff } \bullet_c \to \bullet' \text{ with } \bullet' \to c' \\ sp_\Omega(\bullet_c) \text{ iff } \bullet_c \in \Omega' \end{cases}$$

Summarizing we can define the identity function $id : \Pi \cup \Gamma \cup \Omega \to \mho \cup V(\mho)$ and the spatial function $sp : \Pi \cup \Gamma \cup \Omega \to \mho \cup V(\mho) \cup N$ by:

$$id(s) = \begin{cases} id_\Pi(s) \text{ iff } s \in \Pi \\ id_\Gamma(s) \text{ iff } s \in \Gamma \\ id_\Omega(s) \text{ iff } s \in \Omega \end{cases} \quad sp(s) = \begin{cases} sp_\Pi(s) \text{ iff } s \in \Pi \\ sp_\Gamma(s) \text{ iff } s \in \Gamma \\ sp_\Omega(s) \text{ iff } s \in \Omega \end{cases}$$

Observe that while the range of id is $\mho \cup V(\mho)$, the range of sp is $\mho \cup V(\mho) \cup N$ (we consider here natural numbers as cardinals[4] so that no structure anomaly emerges as long as $N \subset \mho \cup V(\mho)$). Hereafter, for the sake of the presentation, we will still consider natural numbers and not cardinals.

We identify the sets U_A of urelements chosen for ambients, U_P of urelements chosen for atomical processes, and the set of sets of urelements O that contain all the addresses of the elements in Ω'.

We now define *labeled syntax tree* for a given syntax tree of an ambient process.

Definition 7. *Let* $S_P = (S, \to)$ *be the syntax tree of the ambient process* P. *We call* the labeled syntax tree *of it the triplet* $Sl_P = (S, \to, \phi)$ *where* ϕ *is the function defined on the nodes of the syntax tree by*

$$\phi(s) = \langle id(s), sp(s) \rangle \text{ for all } s \in S.$$

Remark 1. It is obvious the central position of the function id in the previous definitions. For a particular ambient process, once we defined the function id, all the construction, up to the labeled syntax tree, can be done inductively on the structure of the ambient process. Because of this, our construction of the labeled syntax tree is unique up to the choice of urelements (i.e. of U_P and U_A).

[4] Informally, we treat 0 as $\emptyset$, 1 as $\{\emptyset\}$, 2 as $\{\emptyset, \{\emptyset\}\}$, 3 as $\{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}$ and so on.

Definition 8. *For a given labeled syntax tree $Sl = (S, \rightarrow, \phi)$ we define the functions:*

- $ur : \Pi \cup \Omega' \rightarrow U_P \cup U_A \cup O$ *by:*

$$ur(s) = \begin{cases} id(s) \text{ if } s \in \Pi \\ sp(s) \text{ if } s \in \Omega' \end{cases}$$

  This function associates to each node of the structure graph the set-theoretical identity defined by the labeled syntax tree
- *Let $e : U_P \cup U_A \cup O \rightarrow \mho \cup V(\mho)$ be the function defined by*

$$e(\nu) = sp(ur^{-1}(\nu))$$

  It associates to each ambient and compound process the set of addresses of its children.
- *$f : U_P \cup U_A \cup O \rightarrow \Lambda \cup \Pi$, where Λ is the set of names of ambients of Ambient Calculus, and Π is the set of atomical processes. For each $\nu \in U_P \cup U_A \subset \mho$, $f(\nu)$ is the name of the process with which ν is associated by id[5], and $f(\nu) = \langle 0, 0\rangle$ if $\nu \in O$. By the function f each urelement (or set of urelements) used as identity will receive the name of the ambient or atomical process that it is pointing to (the sets receive the name $\langle 0, 0\rangle$).*
- *$F : U_P \cup U_A \cup O \rightarrow \Gamma^*$ for each $\nu \in U_P \cup U_A \cup O$, $F(\nu) = \langle c_1, c_2, ...c_k\rangle$ where $c_i \in \Gamma$ such that $\forall i \in N$, $id(c_i) = \nu$, $sp(c_i) = i$ and $\nexists c_{k+1} \in \Gamma$ such that $id(c_{k+1}) = \nu$ and $sp(c_{k+1}) = k + 1$. In the case that, for ν we cannot find any such c_i, we define $F(\nu) = \langle \varepsilon, \varepsilon, ...\rangle$, ε being the null capability. We adopt the following enrichment of the relation of equality on capability chains $=_\Gamma$ defined by the next rules* [6]*:*
  - $\langle c_1, c_2, c_3, ...c_n\rangle - \langle c_1\rangle =_\Gamma \langle c_2, c_3, ...c_n\rangle$
  - $\langle \varepsilon, c_1, ..., c_k\rangle =_\Gamma \langle c_1, c_2, ..., c_k, \varepsilon\rangle =_\Gamma \langle c_1, ..., c_t, \varepsilon, c_{t+1}, ...c_k\rangle =_\Gamma \langle c_1, c_2, ..., c_k\rangle$,
  - $\langle \varepsilon, \varepsilon, ...\varepsilon\rangle =_\Gamma \emptyset$.

The function F associates with each of these the list of capabilities that exists in front of the process they point to.

Definition 9. *Let $S = (S, \rightarrow, \phi)$ be a labeled syntax tree of the ambient process P. We will call the* canonical labeled syntax tree *associated with P, denoted by $S^+ = (S^+, \rightarrow_+, \phi_+)$, the restriction of the labeled syntax tree to the set $S^+ = \{n|\ n \in S,\ f(n) \neq 0$ and $F(n) \neq \langle \varepsilon, ...\varepsilon\rangle\}$, where 0 is the null process and ε is the null capability.*

---

[5] informally we could say that, on $U_A \cup U_P$, we have $f = id^{-1}$, but this is not exact for the reason that id is an injective function while f is not. Because if we have two processes named P, then, for both, the value by f will be P, but, by id^{-1}, they point to different nodes in the syntax tree.

[6] these rules are allowed by the syntax of Ambient Calculus together with the rules of structural congruence over processes

Further we will analyze only canonical labeled trees (by extension canonical processes), these being those who evolves during the ambient calculus computations, so are those who really matters for our purpose.

Other aspects concerning the definition of the labeled syntax tree for situations that involves the new name operator, the replication operator, or recursive processes can be found in [16]. Also we introduce an algebra of labeled trees in order to analyze their composition.

In [16] we proved that the function that associates to each ambient process the set $\langle U_P, U_A, O, e, f, F\rangle$ is generating a sound model for Ambient Calculus. Being this result we construct the logic as having these ordered sets as states. We say that a process satisfies a formula of our logic, if its ordered set (as state) satisfies it.

4 The Logic

The logic we construct is a branching propositional temporal logic, CTL^*[7]. The requirements for such a construction [14] are to organize a structure $\mathcal{M} = (S_0, \Sigma, \Re, \mathcal{L})$ where S_0 is the initial state of our model, Σ is the class of all possible states in our model, $\Re$ is the accessibility relation between states, $\Re \subseteq \Sigma \times \Sigma$, and $\mathcal{L} : \Sigma \to \mathcal{P}(\mathcal{A})$ is a function which associates to each state $S \in \Sigma$ a set of atomical propositions $\mathcal{L}(S) \subseteq \mathcal{P}(\mathcal{A})$ - the set of the atomical propositions true in the state S ($\mathcal{A}$ will be the class of atomical propositions and $\mathcal{P}$ the power-set operator).

We propose to use the ordered sets $S = \langle U_A, U_P, O, e, f, F\rangle$ as states in our logic. The choice of the initial state should depend on the purpose of our analysis. If we are interested in the future of an ambient calculus process P by itself, then its ordered set will be the initial state. But if P will interact with another process Q, or will become child of an ambient, or both like in $m[P|Q]$, then, even if we have a particular interest in P, the initial state should be the ordered set of $m[P|Q]$. For this purpose we defined computation operations over these ordered sets such that to be able, starting from the sets constructed for some initial processes to obtain the sets for other processes constructed in top of these, for more see [16].

The construction of Σ should be done in such a way to contain all the possible future states of the initial state. For this reason we take

$$\Sigma = \{S_i = \langle U_A^i, U_P^i, O^i, e_i, f_i, F_i\rangle \mid U_A^i = U_A^0,\ U_P^i = U_P^0,\ \textit{and}\ O_i = O_0\}$$

where $S_0 = \langle U_A^0, U_P^0, O_0, e_0, f_0, F_0\rangle$ is the initial state. The intuition is that no matter how the process will evolve, it is not possible to appear in it new elements then those that already exist in the initial state[8].

[7] we choose CTL^* because is more expressive then CTL, but a CTL is possible as well

[8] we include here also the situations where some ambients were dissolved by consuming, for example, *open* capability; we consider, in this case, that these ambients still exist in our process but they have an "empty position".

Our main idea is to define the atomic propositions such that to express the basic equations that defines the spatial relations between parts of our process. So, we could define the set of atomical propositions as:

$$\mathcal{A} = \{xiny | x \in U_P \cup U_A \cup O \text{ and } y \in U_A \cup O\}.$$

In our logic we want $xiny$ to be just an atomical proposition and x, y just letters. The cardinality of $\mathcal{A}$ is $card(U_P \cup U_A \cup O) \times card(U_A \cup O)$ which depends (polynomial) on the number of atomical processes and ambients in the ambient calculus process S_0.

Further, the interpretation function $\mathcal{L} : \Sigma \to \mathcal{P}(\mathcal{A})$ is defined by:

$$\mathcal{L}(S) = \{xiny \mid x \in e_y \text{ if } x \in U_P, \text{ or } e_x \in e_y \text{ if } x \in U_A \cup O\}$$

As it concerns the accessibility relation $\Re \subseteq \Sigma \times \Sigma$, following the previous intuition we could define it for two states S_0 and S_1, constructed for the processes P_0 and P_1, by $\langle S_0, S_1 \rangle \in R$ iff $P_0 \to P_1$ (i.e. P_1 can be reached from P_0 in one step of ambient calculus reduction).

Further, we could introduce the syntax of the CTL* logic in the usual way [14]. We inductively define a class of state formulae (formulae which will be true or false of states) and a class of path[9] formulae (true or false of paths), starting from $\mathcal{A}$. We accept, as basic operators the logical operators $\wedge$ and $\neg$, the temporal operators X (*next time*) and $\cup$ (*until*) and the path quantifier E (*for some futures*). We will derive from them all the usual propositional logic operators, the temporal operators G (*always*) and F (*sometimes*) and the path quantifier A (*for all futures*).

4.1 Semantics

Now we define $\models$ inductively. We write $\mathcal{M}, S_0 \models p$ to mean that the state formula p is true at state S_0 in the model $\mathcal{M}$, and $\mathcal{M}, x \models p$ to mean that the path formula p is true for the fullpath x in the structure $\mathcal{M}$. The rules are:

$\mathcal{M}, S_0 \models P$ iff $P \in \mathcal{L}(S_0)$, where $P \in \mathcal{A}$
$\mathcal{M}, S_0 \models p \wedge q$ iff $\mathcal{M}, S_0 \models p$ and $\mathcal{M}, S_0 \models q$
$\mathcal{M}, S_0 \models \neg p$ iff it is not the case that $\mathcal{M}, S_0 \models p$
$\mathcal{M}, S_0 \models Ep$ iff $\exists$ fullpath $x = (S_0, S_1, ...)$ in $\mathcal{M}$ with $\mathcal{M}, x \models p$
$\mathcal{M}, S_0 \models Ap$ iff $\forall$ fullpath $x = (S_0, S_1, ...)$ in $\mathcal{M}$ with $\mathcal{M}, x \models p$
$\mathcal{M}, x \models p$ iff $\mathcal{M}, S_0 \models p$
$\mathcal{M}, x \models p \wedge q$ iff $\mathcal{M}, x \models p$ and $\mathcal{M}, x \models q$
$\mathcal{M}, x \models \neg p$ iff it is not the case that $\mathcal{M}, x \models p$
$\mathcal{M}, x \models p \cup q$ iff $\exists i \left(\mathcal{M}, x^i \models q \text{ and } \forall j \left(j < i \text{ implies } \mathcal{M}, x^j \models p\right)\right)$
$\mathcal{M}, x \models Xp$ iff $\mathcal{M}, x^1 \models p$

[9] A *fullpath* is an infinite sequence $S_0, S_1, ...$ of states such that $(S_i, S_{i+1}) \in \Re$ for all i. We use the convention that if $x = (S_0, S_1, ...)$ denotes a fullpath, then x^i denotes the suffix path $(S_i, S_{i+1}, S_{i+2}, ...)$.

Definition 10. *A state formula p (resp. path formula p) is* valid *provided that for every structure $\mathcal{M}$ and every state S (resp. fullpath x) in $\mathcal{M}$ we have $\mathcal{M}, s \models p$ (resp. $\mathcal{M}, x \models p$). A state formula (resp. path formula) p is* satisfiable *provided that for some structure $\mathcal{M}$ and some states S (resp. fullpath x) in $\mathcal{M}$ we have $\mathcal{M}, S \models p$ (resp. $\mathcal{M}, x \models p$).*

4.2 Describing the state of a system

Consider the example of the interaction between the Virus and Macrophage discussed before. If the mathematical model chosen to describe the interaction is appropriate, then our system should have the property that, independently of the path of time that it will choose, always we will meet, in the future, the situation $n[Infect|Digest]$. Our logic allows us to formulate all these as a logical statement. We have:

$$u[k'[open\ k.k''[Infect]]|n[k[out\ n.in\ k'.in\ n.0]|open\ k'.open\ k''.Digest]] \quad (1)$$

For 1 we choose the urelements: α for u, β for n, o for 0, κ for k, κ' for k', κ'' for k'', p for $Infect$ and q for $Digest$ with $\alpha, \beta, \kappa, \kappa', \kappa'', p, q, o \in \mathfrak{U}$. So, $U_A = \{\alpha, \beta, \kappa, \kappa', \kappa''\}$, $U_P = \{q, p, o\}$, $O = \emptyset$; f is defined by: $f(\alpha) = u$, $f(\beta) = n$, $f(o) = 0$, $f(\kappa) = k$, $f(\kappa') = k'$, $f(\kappa'') = k''$, $f(q) = Infect$, $f(p) = Digest$ and e is defined by:

$$\begin{array}{lll}
e(\alpha) = \{e(\kappa'), e(\beta)\} & \Longrightarrow \begin{cases} e(\kappa') \in e(\alpha) \\ e(\beta) \in e(\alpha) \end{cases} & \Longrightarrow \begin{cases} \kappa' in \alpha \text{ is true} \\ \beta in \alpha \text{ is true} \end{cases} \\
e(\kappa') = \{e(\kappa'')\} & \Longrightarrow \{\ e(\kappa'') \in e(\kappa') & \Longrightarrow \{\ \kappa'' in \kappa' \quad \text{is true} \\
e(\beta) = \{e(\kappa), p\} & \Longrightarrow \begin{cases} e(\kappa) \in e(\beta) \\ p \in e(\beta) \end{cases} & \Longrightarrow \begin{cases} \kappa in \beta \text{ is true} \\ p in \beta \text{ is true} \end{cases} \\
e(\kappa'') = \{q\} & \Longrightarrow \{\ q \in e(\kappa'') & \Longrightarrow \{\ q in \kappa'' \quad \text{is true} \\
e(\kappa) = \{o\} & \Longrightarrow \{\ o \in e(\kappa) & \Longrightarrow \{\ o in \kappa \quad \text{is true}
\end{array}$$

The property we are interested in could be expressed as

$$Macrophage|Virus \models AF(\beta in \alpha \bigwedge q in \beta \bigwedge p in \beta)$$

It says that in all time paths exists at least a reachable state for which n is a child of the master ambient $u = f(\alpha)$, $Infect = f(q)$ and $Digest = f(p)$ are children of the Macrophage ambient $n = f(\beta)$. Further, for checking the truth value of this statement, a model checker could be used. Proving that our logical formula is true it finally means that our mathematical model for describing our problem is a correct one. Vice versa, if is not valid, the model checker will give us a counter example that will show the conflict in our model.

4.3 Algorithms for the accessibility relation

The accessibility relation computation is based on analysis of the initial state structure and all its possible derivatives. What basically defines the possible

evolutions (in time) are the prefixes of the processes involved. For every type of Ambient Calculus reduction (i.e. for each type of capability, c, and for communication) we construct an algorithm able to verify if the conditions of reduction are fulfilled (c-condition algorithm) and an algorithm which computes the final state of the system (c-reduction algorithm). These algorithms are then used within a more general procedure (the general algorithm) that handles the full structure of the initial state.

In order to perform the analysis it is useful to arrange the information in the initial state in two matrices. Consider the following example:

$$u\,[m\,[in\ n.P]\ \mid n\,[\ Q\ ]]$$

If we choose $f(\alpha) = u$, $f(\beta) = m$,$f(\gamma) = n$,$f(p) = P$, and $f(q) = Q$, then the two functions and the matrix are:

matrix T_1

T_1	α	β	γ	p	q
α	0	1	1	0	0
β	0	0	0	1	0
γ	0	0	0	0	1
p	0	0	0	0	0
q	0	0	0	0	0

matrix T_2

T_2	f	F
α	u	ε
β	m	ε
γ	n	ε
p	P	$in\ m$
q	Q	ε

Example (*)

The functions F and f are bundled into T_2 matrix, while the matrix T_1 has one line for each element of $U_P \cup U_A \cup O$, one column for each element of $U_A \cup O$, and is made by setting the entry of column x and row y to 1, if the proposition $xiny$ is true. All the empty entries are set to 0.

In what follows we present the general algorithm and the algorithms for *in*-capability only, the rest of the cases being similar.

General algorithm Assume that the initial state S_1 is described by T_1 and T_2 matrices. The first step in the algorithm is to pick the first column of the F-part in T_2 matrix. For the Example (*) it would be Row = $\{in\ m\}$, just one element.

Now specific c-condition algorithm checks the possibility of using reduction rules of the ambient calculus semantics, and if all the necessary conditions hold then the specific c-reduction part is performed to compute the next state (by updating T_1 and T_2 matrices). In the other case another capability might be chosen in the cycle until either c-reduction algorithm is finally performed or the *Row* set is empty. The algorithm computes exact one state on-forward. See **Algorithm1**.

While the empty place ε is excluded from the set *Row* for the obvious reason, the *output* action is not accepted for avoiding overlapping actions with the accepted *input* action.

Algorithm 1 General form of the accessibility algorithm

1: $Row \Leftarrow \{c \mid c$ is the first column of F-part of the T_2 matrix $\}\backslash\{\varepsilon, output\}$
2: **while** $Row \neq \emptyset$ **do**
3: choose $c \in Row$
4: c-condition
5: **if** *condition* **then**
6: c-reduction
7: $Row \Leftarrow \emptyset$
8: **else**
9: $Row \Leftarrow Row\backslash\{c\}$
10: **end if**
11: **end while**

where c can be In, Out, $Open$ or $Communication$ in c-condition and c-reduction, which depends on chosen capability at the third line.

The notation $S_1 \models_{alg} S_2$ denotes that S_2 state is obtained from S_1 in one step using algorithm 1 instantiated with suitable c-condition and c-reduction parts. We can prove that the accessibility relation between states fulfill the condition:

$$S_1 \Re S_2 \text{ iff } S_1 \models_{alg} S_2.$$

In-condition, *In*-reduction algorithms

$$n\,[in\ m.P \mid Q] \mid m\,[R] \longrightarrow m\,[n\,[P \mid Q] \mid R]$$

The representation of the initial state of the process is the following:

matrix T_1

T_1	α	β	p	q	r
α	0	0	1	1	0
β	0	0	0	0	1
p	0	0	0	0	0
q	0	0	0	0	0
r	0	0	0	0	0

matrix T_2

T_2	f	F
α	n	ε
β	m	ε
p	P	$in\ m$
q	Q	ε
q	R	ε

The In-condition and In-reduction algorithms implement the in-reduction rule of Ambient Calculus semantics.

The In-condition algorithm checks if there is an ambient with the same name as the one in-capability refer to (m in the particular case), at the same nested level as the *parent* process of the capability owner process; it checks also if there is no prefix in front of either ambient processes that will be involved in the reduction. If all the conditions hold then the In-reduction will be performed. It consists in updating T_1 and T_2 such that to represent the final state.

The Out-, $Open$- and $Communication$- condition/reduction algorithms differ from the above w.r.t. the Ambient Calculus reduction rules they describe. For the sake of space, we not discuss them here (for complete details, the reader is referred to [15]).

Algorithm 2 In-condition

$condition \Leftarrow false$
$UrBundle \Leftarrow f_{S_1}^{-1}(m)$
while $UrBundle \neq \emptyset$ **do**
  choose $\nu \in UrBundle$
  if $parent(parent(p)) = parent(\nu)$ AND
  $F_{S_1}(parent(p)) = \varepsilon$ AND
  $F_{S_1}(\nu) = \varepsilon$ AND
  $\forall \mu \in f_{S_1}^{-1}(\langle 0,0 \rangle), \nu \in \mu \Rightarrow F_{S_1}(\mu) = \varepsilon$ **then**
    $condition \Leftarrow true$
    $UrBundle \Leftarrow \emptyset$
  else
    $UrBundle \Leftarrow UrBundle \backslash \{\nu\}$
  end if
end while

Algorithm 3 In-Reduction

{update T_2}
$F_{S_2}(p) \Leftarrow F_{S_1}(p) - \langle in\ m \rangle$
$F_{S_2}(x) \Leftarrow F_{S_1}(x)$ for all $x \neq p$
$f_{S_2}(x) \Leftarrow f_{S_1}(x)$
if $f_{S_2}(p) = \langle 0,0 \rangle \wedge F_{S_2}(p) = \varepsilon$ **then**
  $\forall \mu \in p, F_{S_2}(\mu) \Leftarrow F_{S_2}(\mu) - \langle \star \rangle$
end if

{update T_1}
$\beta in \alpha \Leftarrow 0$
$\beta in \gamma \Leftarrow 1$
if $f_{S_2}(p) = \langle 0,0 \rangle \wedge F_{S_2}(p) = \varepsilon$ **then**
  $\forall \mu \in p, \mu inp \Leftarrow 0$
end if

5 Implementation Details

We present here the details of the implementation we developed for this logic in order to perform model checking analysis for Ambient Calculus. We use the NuSMV model checker for analyzing CTL* logic. Anyway, having the CTL* logic developed for Ambient Calculus, we can use for our purpose any model checker able to analysis temporal logics.

The implementation consists in the construction of a translator (in top of the algorithms presented before) that accepts as input a mobile ambient process and gives, as output, a model specification file for NuSMV model checker. Hereafter we sketch this construction.

The translator assigns to each atomical process or ambient (to each urelement), to each capability and to each ambient process name a natural number, and so it generates the constant definitions for the NuSMV model.

In order to adapt our approach to the requirements of the NuSMV software, we had to represent the matrices T_1 and T_2 by means of arrays.

For the matrix T_1, representing the urelements by natural numbers and using the function *parent* we obtain the representation in NuSMV model as follows:

matrix T_1

T_1	α	β	γ	δ
α	0	1	1	0
β	0	0	0	1
γ	0	0	0	0
δ	0	0	0	0

representation of T_1

parent[α]	parent[β]	parent[γ]	parent[δ]
$< no\_parent >$	α	α	β

In this case the translator converted the 4×4 matrix having 0 or 1 as entries into an array of 4 elements where each of them can have one of the values 0, 1, 2 or 3 (these values represent the identities of the processes and play the same role as the urelements).

For the representation of the matrix T_2, the translator generates the next arrays (functions):

$cap2proc$: $N_c \rightarrow N_p$
$cap2order$: $N_c \rightarrow N_c$
$nextCap$: $N_p \rightarrow N_c$
$cap2name$: $N_c \rightarrow N_n$
$proc2name$: $N_p \rightarrow N_n$
$enabled$: $N_p \rightarrow boolean$

where N_c, N_p and N_n are integers used to identify, respectively, a capability (N_c), a process (N_p) or a name (N_n).

The array $cap2proc$ stores the information that the capability with identity N_c is prefixing the process with identity N_p.

The array $cap2order$ points out the order in which the capabilities prefixing the same process can be used for reductions. For example, $cap2order[in\ \gamma] = out\ \gamma$ means that $out\ \gamma$ might be used only after $in\ \gamma$ was used.

The array $nextCap$ associates to a process the leftmost capability that prefix it. For instance, $nextCap[\delta] = in\ \gamma$ means that the capability $in\ \gamma$ is enabled in the process δ.

The arrays $cap2name$ and $proc2name$ handle the storing information about names that are used in a process formula. For instance, $cap2name[in\ \gamma] = m$ express that $in\ \gamma$ can only be applied in the case of an ambient with the name m, while $cap2name[\gamma] = m$ is used to express the fact that γ was chosen to name a process with the name m.

The array $enabled$ is used to block the action of some capabilities. For example, it is syntactically possible that the use of a capability to be conditioned by the use of another one which do not belong to the same process (so $cap2order$ is not enough). This is the case for $c_1.(P|c_2.Q)$, where c_2 cannot be consumed before c_1, but this case can arise in presence of communications as well. So, $enabled[in\ \gamma] = 1$ allows the capability to be used while $enabled[in\ \gamma] = 0$ forbid the use of the capability.

Using the procedure described above the translator is able to encode the information behind each state of the system. Further it generates the model for the initial state and for the possible next states using the functions already presented. The initial state consists in an assignment of values for variables. Then using the general algorithm it computes the models of the possible next states.

Fairness constraints generated by the syntactical structure of the ambient process are defined by the translator in order to avoid the stuck of the system and to prevent the appearance of impossible paths.

Finally the translator converts the property we want to verify in a form consistent with the one of the system. In this way the interface with NuSMV is complete.

6 Conclusions

The logic we constructed in top of Ambient Calculus opens the perspective of using model checking algorithms (or software) developed for temporal logics in analyzing mobile computations and, in this way, to predict over the future of the systems (biological systems) described using the calculus.

Having the description of the states, together with the algorithms for accessibility relation, all we have to do for having model checking for mobile computations, is to use, further, the algorithms for model checking CTL* (or the tools already constructed for this purpose). Here we presented the possibility of using NuSMV model checker.

Our ongoing research makes us confident in the possibility to construct such a logic for other calculi used for describing biological systems, e.g. BioAmbients Calculus, or Brane Calculi. In such a way, we could move towards predictions about the future of (the structures of) biological systems that can be described using these calculi.

References

1. HyTech: The HYbrid TECHnology Tool. *http://www-cad.eecs.berkeley.edu/~tah/HyTech/*.
2. NuSMV: A new symbolic model checker. *http://nusmv.irst.itc.it/*.
3. Receptors directly activating trimetric g proteins. *http://courses.washington.edu/conj/gprotein/trimericgp.htm*.
4. The SMV system. *http://www-2.cs.cmu.edu/~modelcheck/smv.html*.
5. VIS homepage. *http://www-cad.eecs.berkeley.edu/~vis/*.
6. B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Molecular Biology of the Cell*. Garland Publishing, Inc., fourth edition, 2002.
7. A.Regev, E.M.Panina, W.Silverman, L.Cardelli, and E.Shapiro. Bioambients: An abstraction for biological compartments. *http://www.luca.demon.co.uk/, to appear in Theoretical Computer Science*, 2003.
8. J. Barwise and L. Moss. *Vicious Circles. On the Mathematics of Non-Wellfounded Phenomena*. CLSI Lecture Notes Number 60 Stanford: CSLI Publication, 1996.
9. L. Cardelli. Brane calculi. *http://www.luca.demon.co.uk/*.
10. L. Cardelli and L. Caires. A spatial logic for concurrency (part i). *Information and Computation, Vol.186/2, pages:194-235*, 2003.
11. L. Cardelli and A.D. Gordon. Ambient logic. *http://www.luca.demon.co.uk/, to appear in Mathematical Structures in Computer Science*.
12. L. Cardelli and A.D. Gordon. Anytime, anywhere. modal logics for mobile ambients. *Proceedings of the 27th ACM Symposium on Principles of Programming Languages*, pages 365–377, 2000.
13. L. Cardelli and A.D. Gordon. Mobile ambients. *Theoretical Computer Science, Special Issue on Coordination, D. Le Metayer Editor*, pages 177–213, June 2000.
14. E. A. Emerson. Temporal and modal logic. *Handbook of Theoretical Computer Science*, B: Formal Models and Sematics:995–1072, 1990.
15. R. Mardare and C. Priami. Computing the accessibility relation for ambient calculus. Technical report, Dipartimento di Informatica e Tlc, University of Trento, 2003. Available at `http://www.dit.unitn.it` following the link Publications.
16. R. Mardare and C. Priami. A propositional branching temporal logic for the ambient calculus. Technical report, Dipartimento di Informatica e Tlc, University of Trento, 2003. Available at `http://www.dit.unitn.it` following the link Publications.

A The construction of a labeled syntax tree

We present further the construction of a labeled syntax tree. Consider the ambient calculus program:

$$m[open\ n.Q|s[out\ m.in\ m.n[open\ t.(\ out\ s.(open\ s.P|R)|K)]\ ]\ ]\ |n[P]. \quad (2)$$

As a general rule, we embed our program into a *master ambient*[10] (the master ambient will have a fresh name). Our program becomes:

$$u[m[open\ n.Q|s[out\ m.in\ m.n[open\ t.(\ out\ s.(open\ s.P|R)|K)]\ ]\ ]\ |n[P]] \quad (3)$$

The syntax tree of this process is in Figure A.

For constructing the labeled syntax tree we will define ϕ. We define the identity function id as:

$id(u) = \alpha$, $id(m) = \beta$, $id(n) = \gamma$ (the child of u), $id(s) = \delta$, $id(n) = \mu$, $id(Q) = q$, $id(P) = p'$ (the child of that n which have γ as identity),
$id(P) = p$ (the child of that n which have μ as identity), $id(R) = r$, $id(K) = k$, where $\{\alpha, \beta, \gamma, \delta, \mu, p, q, p', r, k\} \subset \mho$.

Observe that in our situation $\Omega' = \{\bullet', \bullet''\}$ (see Figure A). The space function sp for $\Pi \cup \Omega'$ will be defined starting from the values of id for atomic processes and following the definition of decoration:

$$sp(u) = \{sp(m), sp(n)\} \text{ (here } n \text{ is the child of } u),\ sp(m) = \{sp(s), q\},$$
$$sp(n) = \{p'\} \text{ (the child of } u),\ sp(s) = \{sp(n)\},\ sp(n) = \{sp(\bullet')\},$$
$$sp(\bullet') = \{k, sp(\bullet'')\},\ sp(\bullet'') = \{p, r\}.$$

For capabilities the identity function have the values:

$$id(open\ n) = q,\ id(out\ m) = \mu,\ id(in\ m) = \mu,\ id(open\ t) = \{k, \{p, r\}\},$$
$$id(out\ s) = \{p, r\},\ id(open\ s) = p$$

and the spatial function:

$$sp(open\ n) = 1,\ sp(out\ m) = 1,\ sp(in\ m) = 2,\ sp(open\ t) = 1,\ sp(out\ s) = 1,$$
$$sp(open\ s) = 1$$

Concluding, the function ϕ will be defined as (we will denote $sp(x)$ by sp_x):

$\phi(u) = \langle \alpha, \{sp_m, sp_n\}\rangle$, $\phi(m) = \langle \beta, \{sp_s, q\}\rangle$,
$\phi(n) = \langle \gamma, \{p'\}\rangle$,(the child of u) $\phi(P) = \langle p', p'\rangle$(the child of n),
$\phi(open\ n) = \langle q, 1\rangle$, $\phi(Q) = \langle q, q\rangle$,
$\phi(s) = \langle \delta, \{sp_n\}\rangle$, $\phi(out\ m) = \langle \mu, 1\rangle$,
$\phi(in\ m) = \langle \mu, 2\rangle$, $\phi(n) = \langle \mu, sp_{\bullet'}\rangle$,
$\phi(\bullet') = \langle \emptyset, \{sp_{\bullet''}\}\rangle$, $\phi(open\ t) = \langle \{k, \{p, r\}\}, 1\rangle$,
$\phi(\bullet'') = \langle \emptyset, \{p, r\}\rangle$, $\phi(K) = \langle k, k\rangle$,
$\phi(out\ s) = \langle \{p, r\}, 1\rangle$, $\phi(R) = \langle r, r\rangle$,
$\phi(open\ s) = \langle p, 1\rangle$, $\phi(P) = \langle p, p\rangle$,
for all $\bullet \in \Omega \setminus \Omega'$, $\phi(\bullet) = \langle \emptyset, 0\rangle$, for all $| \in \Omega$, $\phi(|) = \langle \emptyset, 0\rangle$,

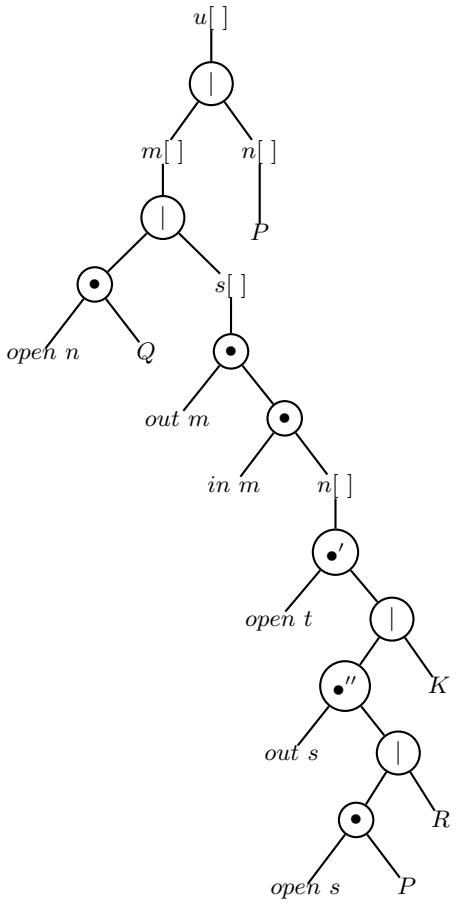

Fig. 1. Syntax tree of the process 3.

The labeled syntax tree is in Figure A.
We can define now the functions ur, e, f and F.

$$ur(u) = \alpha,\ ur(m) = \beta,\ ur(n) = \gamma \text{ (the child of } u),\ ur(s) = \delta,\ ur(n) = \mu,$$
$$ur(Q) = q,\ ur(P) = p' \text{ (the child of } n),\ ur(P) = p,\ ur(R) = r,\ ur(K) = k,$$
$$ur(\bullet') = \{k, \{p, r\}\},\ ur(\bullet'') = \{p, r\}$$

We can define now the function f:

$$f(\alpha) = u,\ f(\beta) = m,\ f(\gamma) = n,\ f(\delta) = s,\ f(\mu) = n,\ f(q) = Q,\ f(p) = P,$$
$$f(p') = P,\ f(r) = R,\ f(k) = K,\ f(\{p, r\}) = \langle 0, 0\rangle,\ f(\{k, \{p, r\}\}) = \langle 0, 0\rangle$$

[10] This is a technical trick that is not disturbing our analysis because of the rule (RedAmb): $P \to Q \Rightarrow n[P] \to n[Q]$, [13], but it helps to treat the processes as a whole from the spatial point of view.

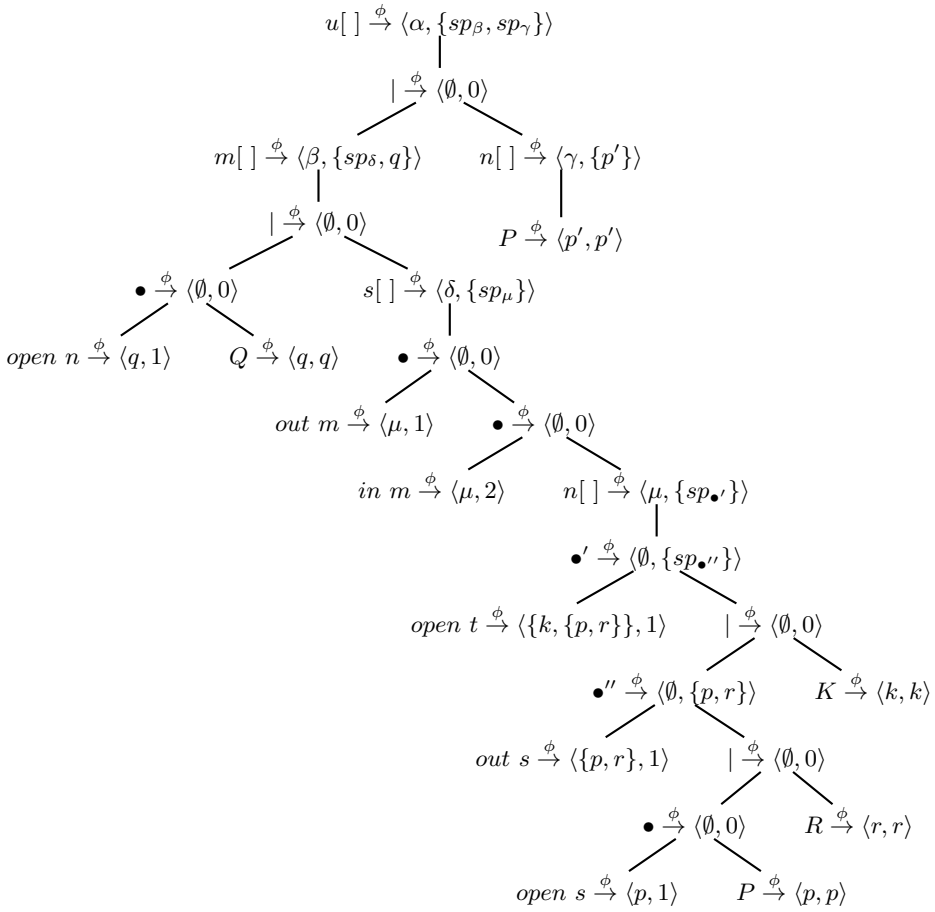

Fig. 2. Labeled syntax tree of 3.

We define, as before, $U_A = \{u \in \mho \mid f(u) \in \Lambda\}$ and $U_P = \{u \in \mho \mid f(u) \in \Pi\}$, which in our example became:
$U_P = \{p, q, r, k, p'\}$, $U_A = \{\alpha, \beta, \gamma, \delta, \mu\}$ and $O = \{\{k, \{p, r\}\}, \{p, r\}\}$.

The function e (as before, we denote $e(x)$ by e_x):

$$e_\alpha = \{e_\beta, e_\gamma\}\ ,\ e_\beta = \{e_\delta, q\},\ e_\gamma = \{p'\}\ ,\ e_\delta = \{e_\mu\},\ e_\mu = \{e_{\{k,\{p,r\}\}}\},$$
$$e_{\{k,\{p,r\}\}} = \{k, e_{\{p,r\}}\},\ e_{\{p,r\}} = \{p, r\}.$$

The function F:

$$F(\alpha) = \langle \varepsilon, \varepsilon, ...\rangle,\ F(\beta) = \langle \varepsilon, \varepsilon, ...\rangle,\ F(\gamma) = \langle \varepsilon, \varepsilon, ...\rangle\ F(\delta) = \langle \varepsilon, \varepsilon, ...\rangle$$
$$F(\mu) = \langle out\ m, in\ m, \varepsilon\rangle,\ F(q) = \langle open\ n, \varepsilon\rangle,\ F(p) = \langle \varepsilon, \varepsilon, ...\rangle,$$
$$F(p') = \langle open\ s, \varepsilon\rangle,\ F(r) = \langle \varepsilon, \varepsilon, ...\rangle,\ F(k) = \langle \varepsilon, \varepsilon, ...\rangle,\ F(\{p, r\}) = \langle out\ s, \varepsilon\rangle,$$
$$F(\{\{p, r\}, k\}) = \langle open\ t, \varepsilon\rangle.$$