Available online at www.sciencedirect.com
SCIENCE DIRECT
ELSEVIER
Electronic Notes in Theoretical Computer Science 99 (2004) 3–29
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

A Logical Approach to Security in the Context of Ambient Calculus

Radu Mardare[1] and Corrado Priami[2]

Dipartimento di Informatica e Telecomunicazioni
University of Trento, Italy

Abstract

In this paper [2] we advocate the use of a CTL* logic, built upon Ambient Calculus to analyze security properties. Our logic is a more expressive alternative to Ambient Logic, based on a single modality, but still powerful enough to handle mobility and dynamic hierarchies of locations. Moreover, having a temporal logic to express properties of computation, we can reuse the algorithms for model checking temporal logics in analyzing models for security problems.
We resort to syntax trees of Ambient Calculus and enrich them with some labeling functions in order to obtain what we called labeled syntax trees. The labeled syntax trees will be used as possible worlds in a Kripke structure developed for a propositional branching temporal logic. The accessibility relation is generated by the reduction of Ambient Calculus considered as reduction between syntax trees.
Providing the algorithms for calculating the accessibility relation between states, we open the perspective of model checking Ambient Calculus by using our algorithms together with the algorithms for model checking temporal logic.

[b] Work partially supported by the IST-FET project DEGAS and the MIUR-COFIN01 project MEFISTO.

Keywords: ambient calculus, temporal logic, set theory, model checking.

1 Introduction

Ambient Calculus [5] is a useful tool to construct mathematical models for security problems because of its facilities in expressing hierarchies of locations and their mobility. Strongly based on Ambient Calculus was constructed the

[1] Email: mardare@dit.unitn.it
[2] Email: priami@dit.unitn.it

doi:10.1016/j.entcs.2004.02.001

Ambient Logic [4,3], a logic that can describe properties of mobile computations as well as the hierarchy of locations and modifications of this hierarchy in time.

The main idea of Ambient Logic is treating processes as spatio-temporal entities, thus were used two types of modalities - one for assertions about space and the other for assertions about time. We will prove here that a single modality suffices for describing the behavior of these entyties. The main intuition is that we only need to calculate the modification that each *movement* of the system is making over the initial state, because this information suffices to reconstruct the actual shape of the system using only a propositional branching temporal logic based on Ambient Calculus.

The advantage of using a temporal logic is relevant in security problems. Consider the model of the interaction between a firewall that keeps its name completely secret and an agent that intends to cross the firewall by means of previously arranged passwords $k,\ k',\ k''$ [5].

$$
\begin{gathered}
Firewall \stackrel{def}{=} (\nu n)n[k[out\ n.in\ k'.in\ n.0]|open\ k'.open\ k''.P] \\
Agent \stackrel{def}{=} k'[open\ k.k''[Q]] \\
Agent|Firewall \equiv \\
(\nu n)(k'[open\ k.k''[Q]]|n[k[out\ n.in\ k'.in\ n.0]|open\ k'.open\ k''.P]) \\
\rightarrow^* (\nu n)(k'[open\ k.k''[Q]]|k[in\ k'.in\ n.0]|n[open\ k'.open\ k''.P]) \\
\rightarrow^* (\nu n)(k'[open\ k.k''[Q]|k[in\ n.0]]|n[open\ k'.open\ k''.P]) \\
\rightarrow^* (\nu n)(k'[k''[Q]|in\ n.0]|n[open\ k'.open\ k''.P]) \\
\rightarrow^* (\nu n)(n[k'[k''[Q]]|open\ k'.open\ k''.P]) \\
\rightarrow^* (\nu n)(n[k''[Q]|open\ k''.P]) \\
\rightarrow^* (\nu n)n[Q|P]
\end{gathered}
$$

This computation describes correctly the desired interaction. Our verification was possible because the formula was not too big, the computation was made in a few steps, and only one path of reductions was possible. In a more complex situation, in which the computational path have branches, i.e. there are moments when more then one reduction is possible, each possibility meaning different possible futures, when are involved also other agents that may know, entirely or partially, the passwords, we want to check wether this situation is always reachable, or if not, which is the configuration of the system in the excepted case.

We cannot obtain an answer to these problems inside Ambient Calculus. Only a logic able to predict about computations could give us an answer. Ambient Logic could express the possibility that a spatial configuration $\mathcal{A}$ of

the system will appear in one of the possible futures, by the formula $\diamond\mathcal{A}$ [3], but it cannot say if this moment is the next one, or a moment very far in future, as it cannot say if this future is to be found in any possible temporal path. In our example we are interested to express that, for all possible future paths, sometime in the future, we will have the interaction between the processes P and Q. This is not possible using Ambient Logic. We can only say that, sometime in the future, this communication is, indeed, possible, but this does not exclude the possibility of existing a temporal path in which this interaction is not possible at all. It could also be important in other situations to know if a property is true in the next moment, or, as in the situation when we want a communication not to be possible until a password is recognized, that a property is false until another one becomes true. All these properties are expressible using a temporal logic.

Another argument for using temporal logics to model Ambient Calculus is the possibility of having model checking for our calculus reusing some software already developed for these logics such as SMV, NuSMV, SiMpLer, VIS.

Further we will enrich the syntax trees of the ambient processes with a decoration function that helps us interpreting them as states in a temporal logic. The action of the decoration function will group the nodes of the syntax tree by their nature and will associate with each one some identities that will depict the hierarchical structure of the ambient process.

2 Labeled syntax trees

In this section we define the *labeled syntax trees* for the ambient calculus processes starting from the syntax trees. This notion is crucial for the further construction, because some abstractions of the labeled syntax trees will be the states in the logic we are going to construct.

We first only consider processes without the new name operator (handled in the next section).

A syntax tree $S = (S, \rightarrow_S)$ for a process is a graph with $S = \mathfrak{P} \cup \mathfrak{C} \cup \mathfrak{O} = (\mathfrak{P}_P \cup \mathfrak{P}_A) \cup \mathfrak{C} \cup \mathfrak{O}$ where

$\mathfrak{P}$ is a set that contain all the unspecified process nodes (hereafter atomical processes [4] and collected in the subset $\mathfrak{P}_P$) and the ambient nodes (collected

[3] $\diamond$ is used for temporal possibility

[4] 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 $n[in\ m.P]$

in the subset $\mathfrak{P}_A$);

$\mathfrak{C}$ is the set of capability nodes (we include here the input nodes and the nodes of variables over capabilities as well); and

$\mathfrak{O}$ is the set of syntactical operator nodes (this set contains the parallel operators $|$ and the prefix operators, $\bullet$). We identify the subset $\mathfrak{O}' = \{\bullet_1 \in \mathfrak{O} \mid \bullet_1 \rightarrow_S | \} \subseteq \mathfrak{O}$ 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 [5].

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 $\emptyset$):

(i) 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

(ii) to each capability, the identity of the process in front of which this capability is placed

(iii) $\emptyset$, to each syntactical node

The spatial function *sp* associates:

(i) to each ambient the set of the identities of its children [6], while to unspecified processes associates the *id*-label.

(ii) 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

(iii) to each syntactical node the spatial function associates 0, except for the nodes in $\mathfrak{O}'$ 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

[5] 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 $c.(P|Q)$

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

the Foundation Axiom (FA), as being a fertile field that offers many tools for analyzing structures, as argued in [2]. 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 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 2.1 *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 2.2 *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 2.3 *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 2.4 *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 $\mathfrak{P} \cup \mathfrak{O}'$, i.e. the graph $T_P = (\mathfrak{P} \cup \mathfrak{O}', \rightarrow_T)$ defined by:*
for $n, m \in \mathfrak{P} \cup \mathfrak{O}'$ we have $n \rightarrow_T m$ iff $n \rightarrow_S^* m$ and $\nexists p \in \mathfrak{P} \cup \mathfrak{O}'$ 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 $\mathfrak{P}$.

Definition 2.5 *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 2.6 *Let the next functions be defined on the subsets of nodes of the syntax tree $(S, \rightarrow)$ as follows:*

- *Let* $sp_{\mathfrak{P}} : \mathfrak{P} \cup \mathfrak{O}' \to V(\mathfrak{V}) \cup \mathfrak{V}$ *be a decoration of the structure graph associated with our syntax tree.*
- *Let* $id_{\mathfrak{P}} : \mathfrak{P} \to \mathfrak{V}$ *be an injective function such that* $id_{\mathfrak{P}}(P) = sp_{\mathfrak{P}}(P)$ *for all* $P \in \mathfrak{P}_P$. *Consider* $U_P \stackrel{def}{=} id_{\mathfrak{P}}(\mathfrak{P}_P) \subset \mathfrak{V}$, $U_A \stackrel{def}{=} id_{\mathfrak{P}}(\mathfrak{P}_A) \subset \mathfrak{V}$
- *Let* $sp_{\mathfrak{O}} : \mathfrak{O} \to \mathfrak{V} \cup V(\mathfrak{V}) \cup \mathbb{N}$ *defined by*

$$sp_{\mathfrak{O}}(s) = \begin{cases} sp_{\mathfrak{P}}(s) & \textit{iff } s \in \mathfrak{O}' \\ 0 & \textit{iff } s \in \mathfrak{O} \setminus \mathfrak{O}' \end{cases}$$

  Consider $O \stackrel{def}{=} sp_{\mathfrak{O}}(\mathfrak{O}') \subset V(\mathfrak{V})$
- *Let* $id_{\mathfrak{O}} : \mathfrak{O} \to V(\mathfrak{V}) \cup \mathfrak{V}$ *defined by*

$$id_{\mathfrak{O}}(s) = \emptyset$$

- *Let* $sp_{\mathfrak{C}} : \mathfrak{C} \to \mathbb{N}$ *such that*

$$sp_{\mathfrak{C}}(c) = \begin{cases} 1 & \textit{iff } | \to \bullet \to c \textit{ or } n \to \bullet \to c \textit{ with } n \in \mathfrak{P} \\ k+1 & \textit{iff } \bullet_1 \to \bullet_2 \to c \textit{ and } \bullet_1 \to c' \in \mathfrak{C} \textit{ with } sp_{\mathfrak{C}}(c') = k \end{cases}$$

- *Let* $id_{\mathfrak{C}} : \mathfrak{C} \to V(\mathfrak{V}) \cup \mathfrak{V}$ *defined for* $c \in \mathfrak{C}$ *such that* $\bullet_c \to c$ *by*

$$id_{\mathfrak{C}}(c) = \begin{cases} id_{\mathfrak{P}}(n) & \textit{iff } \bullet_c \to n \textit{ with } n \in \mathfrak{P} \\ id_{\mathfrak{C}}(c') & \textit{iff } \bullet_c \to \bullet' \textit{ with } \bullet' \to c' \\ sp_{\mathfrak{O}}(\bullet_c) & \textit{iff } \bullet_c \in \mathfrak{O}' \end{cases}$$

Summarizing we can define the identity function $id : \mathfrak{P} \cup \mathfrak{C} \cup \mathfrak{O} \to \mathfrak{V} \cup V(\mathfrak{V})$ and the spatial function $sp : \mathfrak{P} \cup \mathfrak{C} \cup \mathfrak{O} \to \mathfrak{V} \cup V(\mathfrak{V}) \cup \mathbb{N}$ by:

$$id(s) = \begin{cases} id_{\mathfrak{P}}(s) \text{ iff } s \in \mathfrak{P} \\ id_{\mathfrak{C}}(s) \text{ iff } s \in \mathfrak{C} \\ id_{\mathfrak{O}}(s) \text{ iff } s \in \mathfrak{O} \end{cases} \qquad sp(s) = \begin{cases} sp_{\mathfrak{P}}(s) \text{ iff } s \in \mathfrak{P} \\ sp_{\mathfrak{C}}(s) \text{ iff } s \in \mathfrak{C} \\ sp_{\mathfrak{O}}(s) \text{ iff } s \in \mathfrak{O} \end{cases}$$

Observe that while the range of id is $\mathfrak{V} \cup V(\mathfrak{V})$, the range of sp is $\mathfrak{V} \cup V(\mathfrak{V}) \cup \mathbb{N}$ (we consider here natural numbers as cardinals [7] so that no structure anomaly emerges as long as $\mathbb{N} \subset \mathfrak{V} \cup V(\mathfrak{V})$). Hereafter, for the sake of the presentation, we will still consider natural numbers and not cardinals.

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

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 $\mathfrak{O}'$.

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

Definition 2.7 *Let $S_P = (S, \rightarrow)$ be the syntax tree of the ambient process P. We call* the labeled syntax tree *of it the triplet $Sl_P = (S, \rightarrow, \phi)$ 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 2.8 *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).*

Definition 2.9 *For a given labeled syntax tree $Sl = (S, \rightarrow, \phi)$ we define the functions:*

- $ur : \mathfrak{P} \cup \mathfrak{O}' \rightarrow U_P \cup U_A \cup O$ *by:*

$$ur(s) = \begin{cases} id(s) & \text{if } s \in \mathfrak{P} \\ sp(s) & \text{if } s \in \mathfrak{O}' \end{cases}$$

  This function associates to each node of the structure graph the set-theoretical identity defined by the labeled syntax tree

- *Let $e : U_P \cup U_A \cup O \rightarrow \mathfrak{V} \cup V(\mathfrak{V})$ 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 $\nu \in U_P \cup U_A \subset \mathfrak{V}$, $f(\nu)$ is the name of the process with which ν is associated by id*[8]*, 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*

[8] 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.

atomical process that it is pointing to (the sets receive the name $\langle 0,0\rangle$*).*

- $F : U_P \cup U_A \cup O \to \mathfrak{C}^*$ *for each* $\nu \in U_P \cup U_A \cup O$, $F(\nu) = \langle c_1, c_2, ...c_k\rangle$ *where* $c_i \in \mathfrak{C}$ *such that* $\forall i \in \mathbb{N}$, $id(c_i) = \nu$, $sp(c_i) = i$ *and* $\nexists c_{k+1} \in \mathfrak{C}$ *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* $=_{\mathfrak{C}}$ *defined by the next rules* [9] *:*
  - $\langle c_1, c_2, c_3, ...c_n\rangle - \langle c_1\rangle =_{\mathfrak{C}} \langle c_2, c_3, ...c_n\rangle$
  - $\langle \varepsilon, c_1, ..., c_k\rangle =_{\mathfrak{C}} \langle c_1, c_2, ..., c_k, \varepsilon\rangle =_{\mathfrak{C}} \langle c_1, ..., c_t, \varepsilon, c_{t+1}, ...c_k\rangle =_{\mathfrak{C}} \langle c_1, c_2, ..., c_k\rangle$,
  - $\langle \varepsilon, \varepsilon, ...\varepsilon\rangle =_{\mathfrak{C}} \emptyset$.

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

Definition 2.10 *Let* $S = (S, \to, \phi)$ *be a labeled syntax tree of the ambient process* P. *We will call the* canonical labeled syntax tree *associated with* P, *denoted by* $S^+ = (S^+, \to_+, \phi_+)$, *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.*

Further we will discuss only about 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.

3 Handling the binding operators

Consider the interaction between the firewall and the agent in parallel with other two processes $n[R]$ and *open* $n.t[S]$:

$k'[open\ k.k''[Q]]|(\nu n)n[k[out\ n.in\ k'.in\ n.0]|open\ k'.open\ k''.P]|n[R]|open\ n.t[S]$

Here, (νn) means that the name n inside the scope of (νn) is different of all the other names in the program. In the example, we want to be sure that *out* n and *in* n, which are capabilities prefixing the process 0, will never act over $n[R]$ but only over the ambient that was chosen to name the firewall. Vice versa, *open* n, the capability of t, will never act over the firewall ambient, but only over $n[R]$.

The intuition is that the name of the ambient chosen to name the firewall should be one unused before. A possible solution could be just to choose a new name $r \in \Lambda$ and to replace n with it in all its occurrences inside the scope of (νn). This solution is, locally, good, but it will not prevent the name r

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

to be ever used in other processes that we could combine with ours, so that a name conflict would arise. In other words the renaming solution is not a compositional one. We guarantee compositionality by a trick that resembles de Bruijn indexes for name-free λ-calculus: we accept ordered pairs of natural numbers as possible names of ambients and we use them to completely remove any (νn) occurrence from processes. So, we replace the k^{th} new name (νn) in a process with the pair $\langle k, 1\rangle$ [10]. This approach allows us to combine our process with others for which we already constructed the labeled syntax trees. In this way all the names in the second process will receive names as $\langle k, 2\rangle$ meaning that is the k^{th} new name of the second process, and so on, the k^{th} new name of the l^{th} process will receive the name $\langle k, l\rangle$.

This construction is supported by the assumption that inside of an ambient process can only occur a finite number of new name operators and that we will combine only a finite number of processes.

According with the above, our example becomes:

$$k'[open\ k.k''[Q]]|\langle 1,1\rangle[k[out\ \langle 1,1\rangle.in\ k'.in\ \langle 1,1\rangle.0]|open\ k'.open\ k''.P] \\ |n[R]|open\ n.t[S]$$

The analysis of the reductions of our process as in the introduction, shows that the expected result is still possible without using the new name operator. Indeed:

$$Firewall \stackrel{def}{=} \langle 1,1\rangle[k[out\ \langle 1,1\rangle.in\ k'.in\ \langle 1,1\rangle.0]|open\ k'.open\ k''.P]$$
$$Agent \stackrel{def}{=} k'[open\ k.k''[Q]]$$
$$Agent|Firewall|n[R]|open\ n.t[S] \equiv$$
$$k'[open\ k.k''[Q]]|\langle 1,1\rangle[k[out\ \langle 1,1\rangle.in\ k'.in\ \langle 1,1\rangle.0]|open\ k'.open\ k''.P]|n[R]|open\ n.t[S]$$
$$\rightarrow^* k'[open\ k.k''[Q]|k[in\ \langle 1,1\rangle.0]]|\langle 1,1\rangle[open\ k'.open\ k''.P]|n[R]|open\ n.t[S]$$
$$\rightarrow^* k'[k''[Q]|in\ \langle 1,1\rangle.0]|\langle 1,1\rangle[open\ k'.open\ k''.P]|n[R]|open\ n.t[S]$$
$$\rightarrow^* \langle 1,1\rangle[k'[k''[Q]]|open\ k'.open\ k''.P]|n[R]|open\ n.t[S]$$
$$\rightarrow^* \langle 1,1\rangle[Q|P]|n[R]|open\ n.t[S]$$
$$\rightarrow^* \langle 1,1\rangle[Q|P]|R|t[S]$$

We handle $\langle l, k\rangle$ as any other ambient name, whenever it appears in our processes. This means that the set Λ contains, as a subset, a subset of $\mathbb{N} \times \mathbb{N}$. This modification does not affect the four rules of structural congruence ((Struct Res Res), (Struct Res Par), (Struct Res Amb) and (Struct Zero Res), see [5]). Only it modifies the intentional interpretation of (νn). It will not mean *this name is new inside the scope of our quantifier*, but *replace this name in all its occurrences inside the scope of our quantifier by an unused pair of*

[10] we will replace in the ambient calculus process, all the occurrences of n inside the scope of (νn), being ambients or capabilities, with $\langle k, 1\rangle$

natural numbers.

In this way we reduce all the syntax trees of ambient calculus to syntax trees without new name operators.

4 Compositional analysis of Labeled Syntax Trees

In this section we define an Algebra of labeled syntax trees extending the compositional operations of Ambient Calculus from syntax trees to the labeled syntax trees. This mean that, starting from the labeled syntax tree of P_1 and P_2 we will define those for $P_1|P_2$, $c_1.c_2...c_n.m[P]$ or $!P$.

Such a construction can be reduced to the construction of the function id for each case [11] .

4.1 Parallel composition

Assume that $(S_1, \rightarrow_1, \phi_1)$ and $(S_2, \rightarrow_2, \phi_2)$ are the labeled syntax trees for the processes P_1 respective P_2. According to the Remark2.8, we can suppose that the sets of urelements chosen for the two labeled trees are disjunct (if this is not the case, we can choose other urelements for P_1, because the labeled tree is unique up to the choice of the urelements). We can construct the syntax tree for $P_1|P_2$ using the rules of Ambient Calculus. All we have to do further is to define the function ϕ for the new syntax tree.

Suppose that $\alpha_1, \alpha_2 \in \mathfrak{V}$ are the identities of the master ambients in the two cases (i.e. $(S_1, \rightarrow_1, \phi_1)$ is the tree for $u_1[P_1]$ and $(S_2, \rightarrow_2, \phi_2)$ is the tree for $u_2[P_2]$). Consider $\alpha \in \mathfrak{V}$ a new urelement (unused in the two labeled trees). Now we define the function id for $u[P_1|P_2]$ by:

$$id(u) = \alpha,$$
$$id(n) = id_1(n) \text{ for each } n \in \mathfrak{P}_1 \setminus \{\alpha_1\},\ id \text{ is not defined in } \alpha_1,$$
$$id(n) = id_2(n) \text{ for each } n \in \mathfrak{P}_2 \setminus \{\alpha_2\},\ id \text{ is not defined in } \alpha_2$$

Of course, from the way of composing two processes by parallel operator, we have $e_\alpha = e_{\alpha_1} \cup e_{\alpha_2}$.

4.2 Ambient composition

Assume that the labeled syntax tree for P_1 is $(S_1, \rightarrow_1, \phi_1)$. We want to construct the labeled syntax tree $(S, \rightarrow, \phi)$ for $c_1.c_2...c_k.m[P_1]$. Consider that the labeled syntax tree of P_1 has u_1 as its master ambient the ambient with the identity α_1. Let $\alpha, \beta \in \mathfrak{V}$ be two urelements unused in the labeled syntax tree of P_1. Let u be the master ambient of $c_1.c_2...c_k.m[P_1]$. We can define, in the

[11] see the Remark2.8

standard way, the syntax tree of $u[c_1.c_2...c_k.m[P_1]]$ using the rules of Ambient Calculus. Further we define id for it by:

$$id(u) = \alpha,\ id(m) = \beta,$$
$$id(n) = id_1(n) \text{ for all } n \in \mathfrak{P}_1 \setminus \{u\},\ id \text{ is not defined in } u$$

Of course, from the way ambient composition is defined, we have $e_\alpha = \{e_\beta\}$, and $e_\beta = e_u$.

4.3 Algebra of labeled trees

If we call by $\mathfrak{LST}$ the class of all labeled syntax tree (with respect to the identity up to the choice of urelements), the two operations defined before could be introduced as:

$$\|: \mathfrak{LST} \times \mathfrak{LST} \to \mathfrak{LST} \text{ for the parallel composition, and}$$
$$c_1.c_2...c_k.m@ : \mathfrak{LST} \to \mathfrak{LST} \text{ for the ambient composition}$$

If $(S_1, \to_1, \phi_1)$ and $(S_2, \to_2, \phi_2)$ are the labeled syntax trees for the processes P_1 respective P_2 then the labeled syntax tree for $P_1|P_2$ constructed before is $(S_1, \to_1, \phi_1) \parallel (S_2, \to_2, \phi_2)$ and $c_1.c_2...c_k.m@(S_1, \to_1, \phi_1)$ is the one constructed before for $c_1.c_2...c_k.m[P_1]$.

These organize an interesting algebraic structure over $\mathfrak{LST}$.

4.4 The Replication

Assume that the labeled syntax tree for P is $(S_P, \to_P, \phi_P)$. Then the one for $P|P$ will be $(S_P, \to_P, \phi_P) \parallel (S_P, \to_P, \phi_P)$. Reconsidering the way of constructing the parallel composition of labeled trees we observe that for constructing $(S_P, \to_P, \phi_P) \parallel (S_P, \to_P, \phi_P)$ we have to choose a duplicate of $(S_P, \to_P, \phi_P)$ that have the set of urelements disjunct of the initial one. Moreover, for each $\alpha_1 \in \mathfrak{V}$ used as identity for $n \in \mathfrak{P}$ in $(S_P, \to_P, \phi_P)$ we have to choose an $\alpha_2 \in \mathfrak{V}$ used as identity for the same ambient or atomical process $n \in \mathfrak{P}$, but in the duplicated $(S_P, \to_P, \phi_P)$.

We can define $(S_P, \to_P, \phi_P) \parallel (S_P, \to_P, \phi_P) \stackrel{def}{=} (S_P, \to_P, \phi_P) \parallel^2$

In the same way we can define $(S_P, \to_P, \phi_P) \parallel^k$ for all $k \in \mathbb{N}$. Generally speaking, this construction is supposing to choose for each ambient or atomical process found, not an urelement as identity, but a finite chain of k urelements. For each $i \in 1, 2, ...k$, the i^{th} elements of each chain satisfying the requirements of the identity labels for the elements of $\mathfrak{P}$. I.e., if $\{\alpha, \beta, \gamma, ...\zeta\}$ are the urelements chosen for $(S_P, \to_P, \phi_P)$, then we will have the chains:
for α: $\alpha_1, \alpha_2, ..., \alpha_k$, for β: $\beta_1, \beta_2, ..., \beta_k$, for γ: $\gamma_1, \gamma_2, ..., \gamma_k$ .. for ζ: $\zeta_1, \zeta_2, ..., \zeta_k$.
And for each $i \in 1, 2, ...k$ the urelements $\{\alpha_i, \beta_i, \gamma_i, ...\zeta_i\}$ are identities for our

ambients and atomical processes of P.

In the same way we can choose a denumerable chain of urelements for each node of the process graph of P. We denote the labeled syntax tree constructed in this way by $(S_P, \rightarrow_P, \phi_P)\ \|^\infty$. More concrete, if the master ambient of $(S_P, \rightarrow_P, \phi_P)$ is α, and if we consider that $\alpha' \in \mathfrak{V}$ is the master ambient of $(S_P, \rightarrow_P, \phi_P)\ \|^\infty$ then $e^\infty_{\alpha'} \stackrel{def}{=} \cup\{e_{\alpha_i} \mid i \in 1, 2, ...k\}$ and $e^\infty_{\nu_j} \stackrel{def}{=} e^j_{\nu_j}$, where $\nu_j \in \{\beta_j, \gamma_j, ...\zeta_j\}$, e^∞ being the function e of $(S_P, \rightarrow_P, \phi_P)\ \|^\infty$ and e^j the one for the labeled tree j.

As a consequence of the previous construction we can state that the labeled tree $(S_P, \rightarrow_P, \phi_P)\ \|^\infty$ is the labeled syntax tree of $!P$.

5 The Logic

The main goal of this paper is to construct a temporal logic strongly based on Ambient Calculus and able to describe properties of mobile computations as well as the hierarchy of locations and their modifications over time.

The logic we intend to construct is a branching propositional temporal logic, CTL^* [12]. The requirements of such a construction [6] are to organize a structure $\mathfrak{M} = (S_0, \mathfrak{S}, \mathfrak{R}, \mathfrak{L})$ where S_0 is the initial state of our model, $\mathfrak{S}$ is the class of all possible states in our model, $\mathfrak{R}$ is the accessibility relation between states, $\mathfrak{R} \subseteq \mathfrak{S} \times \mathfrak{S}$, and $\mathfrak{L} : \mathfrak{S} \longrightarrow \mathcal{P}(\mathfrak{AP})$ is a function which associates to each state $S \in \mathfrak{S}$ a set of atomical propositions $\mathfrak{L}(S) \subseteq \mathcal{P}(\mathfrak{AP})$ - the set of the atomical propositions true in the state S ($\mathfrak{AP}$ will be the class of atomical propositions).

We developed the labeled syntax trees to use them as states in our logic. The choice of the initial state depends on the purpose of our analysis. If we are interested in the future of an ambient calculus process P by himself, then the labeled syntax tree of P 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 will be the labeled syntax tree of $m[P|Q]$ (we can use, for defining this, the computation operations developed for labeled trees, i.e. $\|$ and $m@$).

The intuition in constructing $\mathfrak{S}$ for a given initial state $S_0 = (S_0, \rightarrow_0, \phi_0)$ (consisting in a labeled syntax tree of a given process P) is to be done in such a way that to contain all the syntax trees of all processes that have the same ambients and atomical processes as P, with the same identities, but in possible different spatial structure [13]. Between these states we will eliminate

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

[13] we include here also the situations where some ambients were dissolved by consuming,

those S_i for which $\mathfrak{O}'_i \not\subseteq \mathfrak{O}'_0$ for the reason that such a spatial configuration is not possible for a state obtained from S_0 [14]. These motivate the following definition:

Definition 5.1 *Assume that $S_0 = (S_0, \rightarrow_0, \phi_0)$ is our initial state. Then* $\mathfrak{S} \stackrel{def}{=} \{(S_i, \rightarrow_i, \phi_i) |\ \mathfrak{P}_i \subseteq \mathfrak{P}_0,\ \mathfrak{O}'_i \subseteq \mathfrak{O}'_0,$ *and for all* $n \in \mathfrak{P}_i,\ id_i(n) = id_0(n)\}$.

In conclusion we can consider, by extension, that all these processes have $U_A^0 = U_A^i$, $U_P^0 = U_P^i$ and $O_0 = O_i$. For this reason we discuss further about U_P, U_A and O without other indexes.

Definition 5.2 *We define the set of atomical propositions as:*
$\mathfrak{AP} = \{x\mathfrak{in}y | x \in U_P \cup U_A \cup O$ *and* $y \in U_A \cup O\}$.

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

Definition 5.3 *We define the interpretation function $\mathfrak{L} : \mathfrak{S} \rightarrow \mathcal{P}(\mathfrak{AP})$ by:*

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

Definition 5.4 *We define the accessibility relation $\mathfrak{R} \subseteq \mathfrak{S} \times \mathfrak{S}$ as it follows: if $(S_0, \rightarrow_0, \phi_0)$ and $(S_1, \rightarrow_1, \phi_1)$ are the labeled syntax trees for the processes P_0 and P_1, then*
$\langle (S_0, \rightarrow_0, \phi_0), (S_1, \rightarrow_1, \phi_1) \rangle \in \mathfrak{R}$ *iff* $P_0 \rightarrow P_1$
(i.e. P_1 can be reached from P_0 in one step of ambient calculus reduction).

The accessibility relation can be described using some simple algorithms, one for each reduction rule of Ambient Calculus. We developed these algorithms in a companion paper [8]. They determine, giving a state S, which are the possible states S' such that $(S, S') \in \mathfrak{R}$.

Following the classic way of introducing CTL^* we define:

Definition 5.5 *A* fullpath *is an infinite sequence $S_0, S_1, ...$ of states such that $(S_i, S_{i+1}) \in \mathfrak{R}$ for all i. We use the convention that if $x = (S_0, S_1, ...)$ denotes a fullpath, then x^i denotes the suffix path $(S_i, S_{i+1}, S_{i+2}, ...)$.*

---

for example, *open* capability; we consider, in this case, that these ambients still exist in our process but they have an "empty position".

[14] the reduction rules of Ambient Calculus allow the destruction of some complex processes by consuming capabilities, but does not allow construction of some complex processes.

5.1 Syntax

We inductively define a class of state formulae (formulae which will be true or false of states) and a class of path formulae (true or false of paths), starting from $\mathfrak{AP}$. 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*).

Syntactical rules:

(i) Each atomical proposition $\alpha \mathfrak{in} \beta \in AP$ is a state formula

(ii) If p, q are state formulae then so are $p \wedge q$, $\neg p$

(iii) If p is a path formula then $E\ p$, $A\ p$ are state formulae

1'. Each state formula is a path formula

2'. If p, q are path formulae then so are $p \wedge q$, $\neg p$

3'. If p, q are path formulae then so are Xp, $p \cup q$

Syntactical conventions:

(i) Ap abbreviates $\neg E \neg p$.

(ii) EFp abbreviates $E(true \cup p)$.

(iii) AGp abbreviates $\neg EF \neg p$.

(iv) AFp abbreviates $A(true \cup p)$.

(v) EGp abbreviates $\neg AF \neg p$.

5.2 Semantics

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

$\mathfrak{M}, S_0 \models P$ iff $P \in \mathfrak{L}(S_0)$, where $P \in \mathfrak{AP}$

$\mathfrak{M}, S_0 \models p \wedge q$ iff $\mathfrak{M}, S_0 \models p$ and $\mathfrak{M}, S_0 \models q$

$\mathfrak{M}, S_0 \models \neg p$ iff it is not the case that $\mathfrak{M}, S_0 \models p$

$\mathfrak{M}, S_0 \models Ep$ iff $\exists$ fullpath $x = (S_0, S_1, ...)$ in $\mathfrak{M}$ with $\mathfrak{M}, x \models p$

$\mathfrak{M}, S_0 \models Ap$ iff $\forall$ fullpath $x = (S_0, S_1, ...)$ in $\mathfrak{M}$ with $\mathfrak{M}, x \models p$

$\mathfrak{M}, x \models p$ iff $\mathfrak{M}, S_0 \models p$

$\mathfrak{M}, x \models p \wedge q$ iff $\mathfrak{M}, x \models p$ and $\mathfrak{M}, x \models q$

$\mathfrak{M}, x \models \neg p$ iff it is not the case that $\mathfrak{M}, x \models p$

$\mathfrak{M}, x \models p \cup q$ iff $\exists i\,(\mathfrak{M}, x^i \models q$ and $\forall j\,(j < i$ implies $\mathfrak{M}, x^j \models p))$

$\mathfrak{M}, x \models Xp$ iff $\mathfrak{M}, x^1 \models p$

Definition 5.6 *A state formula p (resp. path formula p) is* valid *provided that for every structure $\mathfrak{M}$ and every state S (resp. fullpath x) in $\mathfrak{M}$ we have $\mathfrak{M}, s \models p$ (resp. $\mathfrak{M}, x \models p$). A state formula (resp. path formula) p is* satisfiable *provided that for some structure $\mathfrak{M}$ and some states S (resp. fullpath x) in $\mathfrak{M}$ we have $\mathfrak{M}, S \models p$ (resp. $\mathfrak{M}, x \models p$).*

The following theorem provides a logical characterization of the structural congruence.

Theorem 5.7 *Let P_1, P_2 be two ambient processes. Then the next assertions are equivalent:*

(i) $P_1 \equiv_\alpha P_2$

(ii) *There are two models $\mathfrak{M}_1$, $\mathfrak{M}_2$ for the two processes such that the next conditions are satisfied:*

(a) *There exists two bijective functions $\psi : U_P^1 \cup U_A^1 \cup O_1 \to U_P^2 \cup U_A^2 \cup O_2$ and $\mathfrak{Pr} : \Lambda \cup \Pi \to \Lambda \cup \Pi$ with the properties*[15] *:*
$\psi(U_P^1) = U_P^2$, $\psi(U_A^1) = U_A^2$ *and* $\psi(y) = \{\psi(x) |$ *for all* $x \in y\}$*;*
$\mathfrak{Pr}(\langle 0, 0\rangle) = \langle 0, 0\rangle$ *and* $\mathfrak{Pr}(n) = n$ *for all* $n \in \Lambda \cup \Pi \setminus (\mathbb{N} \times \mathbb{N})$
$\mathfrak{Pr}(f_1(\alpha)) = f_2(\psi(\alpha))$ *for all* $\alpha \in U_P^1 \cup U_A^1 \cup O_1$
$\mathfrak{Pr}(F_1(\alpha)) =_{\mathfrak{C}} F_2(\psi(\alpha))$ *for all* $\alpha \in U_P^1 \cup U_A^1 \cup O_1$

(b) *The two logics fulfill the conditions:*
$\mathfrak{M}_1, S_1 \models \alpha \mathbf{in} \beta$ *iff* $\mathfrak{M}_2, S_2 \models \psi(\alpha) \mathbf{in} \psi(\beta)$

The meaning of this theorem is that we can identify the structural equivalent processes[16] by the possibility of defining a one-to-one function between the urelements chosen for ambients[17] and between the urelements chosen for atomical processes in the two models that to allow the corresponding atomical propositions in the two logics to be true/false, in the same time, in the corresponding model.

6 Implementing the labeled syntax trees

In this section we sketch the way we implemented the labeled syntax trees in order to make model checking for Ambient Calculus. The main purpose

[15] Further we wrote $\mathfrak{Pr}(\langle c_1, c_2, ...\rangle)$ for all $c_i \in \mathfrak{C}$ in order to denote the result of substituting all names $n \in \Lambda \cup \Pi$ that appear in capabilities by $\mathfrak{Pr}(n)$

[16] do not forget that we discuss exclusively the canonical labeled trees

[17] up to renaming of the new names by the projection $\mathfrak{Pr}$

of this paper is to show the advantages that could be obtained in analyzing the security problems expressed in Ambient Calculus by using our logic. For this reason we will not present here the algorithms that can compute the accessibility relation. These can be found in [8]. We present only the idea behind our implementation.

Moreover, our implementation is adapted to the requirements of NuSMV, but our logic could work as well with over model checkers.

The accessibility relation is defined, inductively, on the structure of the initial state, by analyzing all of its possible derivatives. We have to analyze how the use of the existing prefixes of the ambient process will influence the architecture of our labeled tree.

Consider the ambient process that describes the interaction of a firewall with an agent knowing the passwords, already handled earlier. We have

$$(\nu n)(k'[open\ k.k''[Q]]|n[k[out\ n.in\ k'.in\ n.0]|open\ k'.open\ k''.P]) \qquad (1)$$

We construct the labeled syntax tree for it. As before, we wrap the process into a master ambient u:

$$u[(\nu n)(k'[open\ k.k''[Q]]|n[k[out\ n.in\ k'.in\ n.0]|open\ k'.open\ k''.P])] \qquad (2)$$

and we replace the new name by $\langle 1,1\rangle$.

$$u[k'[open\ k.k''[Q]]|\langle 1,1\rangle[k[out\ \langle 1,1\rangle.in\ k'.in\ \langle 1,1\rangle.0]|open\ k'.open\ k''.P]] \qquad (3)$$

For 3 we choose the urelements: α for u, β for $\langle 1,1\rangle$, o for 0, κ for k, κ' for k', κ'' for k'', p for P and q for Q with $\alpha,\beta,\kappa,\kappa',\kappa'',p,q,o \in \mho$. So, $U_A = \{\alpha,\beta,\kappa,\kappa',\kappa''\}$, $U_P = \{q,p,o\}$, and $O = \emptyset$ and $f : U_P \cup U_A \cup O \longrightarrow \Lambda \cup \Pi$ by: $f(\alpha) = u$, $f(\beta) = \langle 1,1\rangle$, $f(o) = 0$, $f(\kappa) = k$, $f(\kappa') = k'$, $f(\kappa'') = k''$, $f(q) = Q$, $f(p) = P$ and $e : U_A \cup U_P \cup O \longrightarrow \mho \cup V(\mho)$ is defined by:

$$e_\alpha = \{e_{\kappa'}, e_\beta\} \Longrightarrow \begin{cases} e_{\kappa'} \in e_\alpha & \Longrightarrow \begin{cases} \kappa' \mathbf{in} \alpha \text{ is true} \end{cases} \\ e_\beta \in e_\alpha & \Longrightarrow \begin{cases} \beta \mathbf{in} \alpha \text{ is true} \end{cases} \end{cases}$$

$$e_{\kappa'} = \{e_{\kappa''}\} \Longrightarrow \{\ e_{\kappa''} \in e_{\kappa'} \Longrightarrow \{\ \kappa'' \mathbf{in} \kappa' \quad \text{is true}$$

$$e_\beta = \{e_\kappa, p\} \Longrightarrow \begin{cases} e_\kappa \in e_\beta & \Longrightarrow \begin{cases} \kappa \mathbf{in} \beta \text{ is true} \end{cases} \\ p \in e_\beta & \Longrightarrow \begin{cases} p \mathbf{in} \beta \text{ is true} \end{cases} \end{cases}$$

$$e_{\kappa''} = \{q\} \Longrightarrow \{\ q \in e_{\kappa''} \Longrightarrow \{\ q \mathbf{in} \kappa'' \quad \text{is true}$$

$$e_\kappa = \{o\} \Longrightarrow \{\ o \in e_\kappa \Longrightarrow \{\ o \mathbf{in} \kappa \quad \text{is true}$$

Hence our initial state is described by the list of true atomical propositions. We construct two matrices to encode this information.

The first matrix, $\mathbb{T}_1$, 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 $x \mathbf{in} y$ is true. All the empty entries are set to 0. See *Example table1*.

The second matrix, $\mathbb{T}_2$, has as rows the elements of $U_P \cup U_A \cup O$, and as many columns as the number of prefixes forming the largest sequential chain of capabilities in the process plus two. Actually, we have the first column, indexed by f, reporting the value of f applied to the row index (if f is defined in it). The remaining columns define F. The last column is filled by ε. In our example we have
$f(\alpha) = u$, $F(\alpha) = \langle \varepsilon, \varepsilon, ... \rangle$, $f(\beta) = \langle 1, 1 \rangle$, $F(\beta) = \langle \varepsilon, \varepsilon, ... \rangle$,
$f(o) = 0$, $F(o) = \langle out\langle 1, 1 \rangle, in\ k', in\langle 1, 1 \rangle, \varepsilon \rangle$, $f(\kappa) = k$, $F(\kappa) = \langle \varepsilon, \varepsilon, ... \rangle$,
$f(\kappa') = k'$, $F(\kappa') = \langle \varepsilon, \varepsilon, ... \rangle$, $f(\kappa'') = k''$, $F(\kappa'') = \langle open\ k, \varepsilon, ... \rangle$, $f(q) = Q$,
$F(q) = \langle \varepsilon, \varepsilon, ... \rangle$, $f(p) = P$, $F(p) = \langle open\ k', open\ k'', \varepsilon, ... \rangle$.
See *Example table2* for the construction of the matrix $\mathbb{T}_2$.

Example table1

$\mathbb{T}_1$	α	β	κ	κ'	κ''	o	p	q
α	0	1	0	1	0	0	0	0
β	0	0	1	0	0	0	1	0
κ	0	0	0	0	0	1	0	0
κ'	0	0	0	0	1	0	0	0
κ''	0	0	0	0	0	0	0	1

Example table2

$\mathbb{T}_2$	f	F
α	u	$\varepsilon \quad \varepsilon \quad ...$
β	$\langle 1,1\rangle$	$\varepsilon \quad \varepsilon \quad ...$
κ	k	$\varepsilon \quad \varepsilon \quad ...$
κ'	k'	$\varepsilon \quad \varepsilon \quad ...$
κ''	k''	$open\ k \quad \varepsilon \quad ...$
o	0	$out\langle 1,1\rangle \quad in\ k' \quad in\langle 1,1\rangle \quad \varepsilon$
p	P	$open\ k' \quad open\ k'' \quad \varepsilon$
q	Q	$\varepsilon \quad \varepsilon \quad ...$

The two matrices $\mathbb{T}_1$ and $\mathbb{T}_2$ suffice to describe each state S. Actually we proved in [9] that the function that associates to each process the quadruple $(U_P \cup U_A \cup O, e, f, F)$, named *the labeled structure tree associated with our process*, give us a sound model for Ambient Calculus. We will not present this result here, being less important for our purpose.

If we consider the more complex example (A.1) discussed in the Appendix, where $O \neq \emptyset$, for it the two matrices have the form:

$\mathbb{T}_1$	α	β	γ	δ	μ	q	p	p'	r	k	$\{p,r\}$	$\{\{p,r\},k\}$
α	0	1	1	0	0	0	0	0	0	0	0	0
β	0	0	0	1	0	1	0	0	0	0	0	0
γ	0	0	0	0	0	0	0	1	0	0	0	0
δ	0	0	0	0	1	0	0	0	0	0	0	0
μ	0	0	0	0	0	0	0	0	0	0	0	1
$\{p,r\}$	0	0	0	0	0	0	1	0	1	0	0	0
$\{\{p,r\},k\}$	0	0	0	0	0	0	0	0	0	1	1	0

$\mathbb{T}_2$	f	F
α	u	$\varepsilon \quad \varepsilon \quad ...$
β	m	$\varepsilon \quad \varepsilon \quad ...$
γ	n	$\varepsilon \quad \varepsilon \quad ...$
δ	s	$\varepsilon \quad \varepsilon \quad ...$
μ	n	$out\ m \quad in\ m \quad \varepsilon$
q	Q	$open\ n \quad \varepsilon \quad ...$
p	P	$\varepsilon \quad ...$
p'	P	$open\ s \quad \varepsilon$
r	R	$\varepsilon \quad ...$
k	K	$\varepsilon \quad ...$
$\{p, r\}$	$\langle 0, 0\rangle$	$out\ s \quad \varepsilon \quad ...$
$\{\{p, r\}, k\}$	$\langle 0, 0\rangle$	$open\ t \quad \varepsilon \quad ..$

In this case, the elements of O also appear in the matrices. For the construction of the matrices in more complex cases, see also [8].

Putting the information of a state in the form of the two matrices, gives us the possibility to define an algorithm, *the reduction algorithm*, to compute the evolution of states under reductions.

Assume that the initial state S_1 is described by the tables $\mathbb{T}_1$ and $\mathbb{T}_2$. The reduction algorithm will start by choosing randomly a row from the table $\mathbb{T}_2$. It will check the first capability form the chain of capabilities, i.e. the first position of the F-part of the matrix. If the capability found here, c, is the null capability, ε, then the algorithm will chouse an other row; else will proceed to verify the conditions of consuming this capability, hereafter c-condition algorithm. The c-condition algorithm is specific for each type of capability, and its role is to analyze the structure of the ambient process in order to see if such a reduction is possible [18]. If the capability cannot be consumed, the algorithm will chose an other row, different of those unaccepted before, and will restart the reduction algorithm; else it will proceed with c-reduction

[18] it will analyze if the spatial arrangement of the process to see if it allows this reduction. Moreover, this algorithm will check more then the conditions generated by the reduction rules; it will stop the consuming of a forbidden capability, as in the case $c_1.(c_2.P|Q)$ where the consuming of c_2 is forbidden as time as c_1 exists

algorithm which will update the two tables, $\mathbb{T}_1$ and $\mathbb{T}_2$, for the new state obtained by consuming the c capability. We will have a c-reduction algorithm for each type of capability as well.

The reduction algorithm, together with the algorithms $c - condition$ and $c - reduction$ for all the types of capabilities, can be found in [8].

Denoting by $S_1 \models_{alg} S_2$ that S_2 is obtained from S_1, in one step, using the reduction algorithm instantiated with suitable c-condition and c-reduction, we can define the accessibility relation between states as:

$$S_1 \mathfrak{R} S_2 \text{ iff } S_1 \models_{alg} S_2$$

7 Applying the Logic

We will show in this section the advantages of using a temporal logic to describe the computations of Ambient Calculus. Consider the previous example. We defined the firewall and the agent knowing the passwords as

$$Firewall \stackrel{def}{=} (\nu n) n[k[out\ n.in\ k'.in\ n.0] | open\ k'.open\ k''.P]$$

$$Agent \stackrel{def}{=} k'[open\ k.k''[Q]]$$

If the mathematical model chosen to describe the firewall-agent interaction is appropriate, then our system shoud have the property that, independently of the path of time that it will choose, always we will meet, in the future, the situation $(\nu n) n[Q|P]$ (or, equivalently, $\langle 1,1\rangle[Q|P]$).

Our logic allows us to formulate all these as a logical statement. Indeed, if we recall the construction of the labeled syntax tree for the process $Firewall|Agent$ made in the previous section, then the property we are interested in could be expressed as

$$Firewall|Agent \models AF(\beta \mathbf{in} \alpha \bigwedge q \mathbf{in} \beta \bigwedge p \mathbf{in} \beta)$$

It says that in all time paths exists at least a reachable state for which the new name $\langle 1,1\rangle = f(\beta)$ (versus n) is a child of the master ambient $u = f(\alpha)$, $Q = f(q)$ is a child of $\langle 1,1\rangle$ and $P = f(p)$ is a child of $\langle 1,1\rangle$. Further, for checking the truth value of this statement, a model checker could be used together with our algorithms for implementing the state-processes [8]. Proving that our logical formula is true it finally means that our mathematical model for describing the interaction between a $Firewall$ and an $Agent$ knowing the passwords 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.

We now briefly present the results of model checking our example using NuSMV. The following formula represents our process after the translation

into the model for NuSMV

$$u[k1[open\ k.k2[Q]]|n[k[out\ n.in\ k1.in\ n.0]|open\ k1.open\ k2.P]]$$

with the property specification of $n[P|Q]$ (Q crosses the firewall) and its negation that includes the claim that Q will not get inside the ambient n (it never crosses the firewall).

The above properties have been checked running NuSMV on a linux operating system equipping a dual Intel Xeon CPU 2.4GHz with 2Gb RAM. The table below reports the amount of resources consumed.

	RAM (Mb)	CPU (min)
Building the BDD of the model	61	17.54
Checking $AF(n[Q\|P])$	6	27.09
Checking the negation of the above property	4	0.14
Total	71	44.77

The assertion *crossing the firewall* is true, while its negation is false and NuSMV provides the following trace as a counterexample

$\rightarrow\ State1.1\ \leftarrow$
$u[k1[open\ k.k2[Q]]|n[k[out\ n.in\ k1.in\ n.0]|open\ k1.open\ k2.P]]$
$\rightarrow\ State1.2\ \leftarrow$
$u[k1[open\ k.k2[Q]]|n[open\ k1.open\ k2.P]|k[in\ k1.in\ n.0]]$
$\rightarrow\ State1.3\ \leftarrow$
$u[k1[open\ k.k2[Q]|k[in\ n.0]]|n[open\ k1.open\ k2.P]]$
$\rightarrow\ State1.4\ \leftarrow$
$u[k1[k2[Q]|in\ n.0]|n[open\ k1.open\ k2.P]]$
$\rightarrow\ State1.5\ \leftarrow$
$u[n[open\ k1.open\ k2.P|k1[k2[Q]|0]]]$
$\rightarrow\ State1.6\ \leftarrow$
$u[n[open\ k2.P|k2[Q]|0]]$
$\rightarrow\ State1.7\ \leftarrow$
$u[n[P|Q|0]]$

The property discussed here cannot be expressed in Ambient Logic. All we can say using this logic is that

$$Firewall|Agent \models \diamond n \circledR n[\mathcal{A}|\mathcal{B}] \text{ where } P \models \mathcal{A} \text{ and } Q \models \mathcal{B}$$

But this formula says only that it is possible, in some future, to have the desired situation, but not in all possible time paths. If we replace the possibility quantifier by necessity one, we will express the fact that our situation it will happen in all the future moments, so it is still not what we tried to express.

8 Future work: Extending Ambient Calculus to Non-Wellfounded Structures

This approach to CTL* logic for Ambient Calculus is a refinement of a more abstract one we developed in [9] using a set theoretical description of trees as flat systems of equations, inspired by [2]. This set theoretical tools offers interesting perspectives in analyzing structures and, what is the most important, it gives the possibility of analyzing non-wellfounded structures (such as trees with circular branches) with the same accuracy and flexibility as the classical well-founded ones (classical trees). Unfortunately, because of their "unusual" nature, these ideas are not used in Computer Science almost at all in spite of the fact that those who discovered them (F. Honsell and M. Forti (1983) [7], P. Aczel (1988) [1], J.Barwise (1996) [2]) were developing them especially for being used in Computer Science.

We investigated this possibility in connection with Ambient Calculus. Consider, for example, that the firewall is expected to interact with all the agents that know the passwords. Generally speaking this is the situation in reality. Now the previous definition of the firewall is not satisfactory because it was modelled such that, if the interaction with one agent was done, the firewall cannot interact with any other, being unable to recognize the passwords once again.

A solution in this case could be to use the replication operator in the definition of the firewall.

$Firewall \stackrel{def}{=} !((\nu n)n[k[out\ n.in\ k'.in\ n.0]|open\ k'.open\ k''.P])$ We can consider now two agents $Agent_1 \stackrel{def}{=} k'[open\ k.k''[Q_1]]$ and $Agent_2 \stackrel{def}{=} k'[open\ k.k''[Q_2]]$. We have

$$\begin{gathered} Agent_1|Agent_2|Firewall \equiv \\ (\nu n)(k'[open\ k.k''[Q_1]]|n[k[out\ n.in\ k'.in\ n.0]|open\ k'.open\ k''.P]) \\ |(\nu n)(k'[open\ k.k''[Q_2]]|n[k[out\ n.in\ k'.in\ n.0]|open\ k'.open\ k''.P]) \\ |!((\nu n)n[k[out\ n.in\ k'.in\ n.0]|open\ k'.open\ k''.P]) \\ \rightarrow^* (\nu n)n[Q_1|P]|(\nu n)n[Q_2|P]|Firewall \end{gathered}$$

This problem could be treated from a different point of view than using replication operator. This approach will enlarge the ambient calculus syntax to circular and self referential formulae.

Assume, informally, the formula

$$\Omega_P = (\nu n)(open\ n.P \mid n\ [open\ n.P \mid n\ [open\ n.P \mid n\ [....]\ ]\ ])$$

where $n \notin fn(P)$ and the structure of formula is cyclic. Such a formula is forbidden in ambient calculus syntax, but it can be expressed using non-wellfouded trees, because our ambient n has the structure described by the equation:

$$e_\alpha = \{\beta, e_\alpha\}$$

This describes the non-wellfounded set (hyperset) $\alpha = \{\beta, \alpha\}$. We can recognize the non-wellfounded nature, if we try to apply the definition of α:

$$\alpha = \{\beta, \alpha\} = \{\beta, \{\beta, \alpha\}\} = \{\beta, \{\beta, \{\beta, \alpha\}\}\}$$

and so on. We can go on deeper and deeper to infinite.

It is easy to verify that

$$\Omega_P \to \Omega_P|P \to \Omega_P|P|P \to ... \to \Omega_P|P|P|P|...$$

The action of the process Ω_P looks like the action of the replication operator, the only difference is that Ω_P needs time (one reduction step) to replicate, while $!P$ can do it instantaneously (by structural congruence). From other point of view, if we accept circular syntax trees for our processes, we do not need, necessarily, to use such an operator as the replication one. Moreover, the construction of the labeled syntax tree for this case is a very simple one and it follows exactly the same steps as the classical case. We do not need to choose new urelements for describing the cyclic definitions, we only need to write the appropriate function e (the flat system of equations).

9 Conclusions

Our approach to Ambient Calculus opens the perspective of using model checking algorithms (or software) developed for temporal logics in analyzing mobile computations. This is because we found a way of implementing the information behind the ambient processes, using the two matrices, and we constructed the algorithms to calculate the accessibility relation between states.

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* (a CTL is possible also) and we are investigating now this possibility.

Our ongoing researches make us confident in the possibility to use NuSMV, together with an external translator (used to assign to the ambient calculus process its labeled syntax tree) to model check Ambient Calculus.

References

[1] P. Aczel. *Non-Well-Founded Sets.* CLSI Lecture Notes Number 14 Stanford: CSLI Publication, 1988.

[2] J. Barwise and L. Moss. *Vicious Circles. On the Mathematics of Non-Wellfounded Phenomena.* CLSI Lecture Notes Number 60 Stanford: CSLI Publication, 1996.

[3] L. Cardelli and A.D. Gordon. Ambient logic. *http://www.luca.demon.co.uk/.*

[4] 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.

[5] L. Cardelli and A.D. Gordon. Mobile ambients. *Theoretical Computer Science, Special Issue on Coordination, D. Le Mtayer Editor*, pages 177–213, June 2000.

[6] E. A. Emerson. Temporal and modal logic. *Handbook of Theoretical Computer Science*, B: Formal Models and Sematics:995–1072, 1990.

[7] F. Honsell and M. Forti. *Set Theory with Free Construction Principles.* Annali Scuola Normale Supeiore di Pisa, 1983.

[8] 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 www.dit.unitn.it following the link Publications.

[9] 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 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:

$$m[open\ n.Q|s[out\ m.in\ m.n[open\ t.(\ out\ s.(open\ s.P|R)|K)]\ ]\ ]\ |n[P]. \quad (A.1)$$

As a general rule, we embed our program into a *master ambient* [19] (the master ambient will have a fresh name). Our program becomes:

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

The syntax tree of this process is in Figure 19.

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 $\{\alpha, \beta, \gamma, \delta, \mu, p, q, p', r, k\} \subset \mathcal{U}$.

[19] This is a technical trick that is not disturbing our analysis because of the rule (RedAmb): $P \to Q \Rightarrow n[P] \to n[Q]$, [5], but it helps to treat the processes as a whole from the spatial point of view.

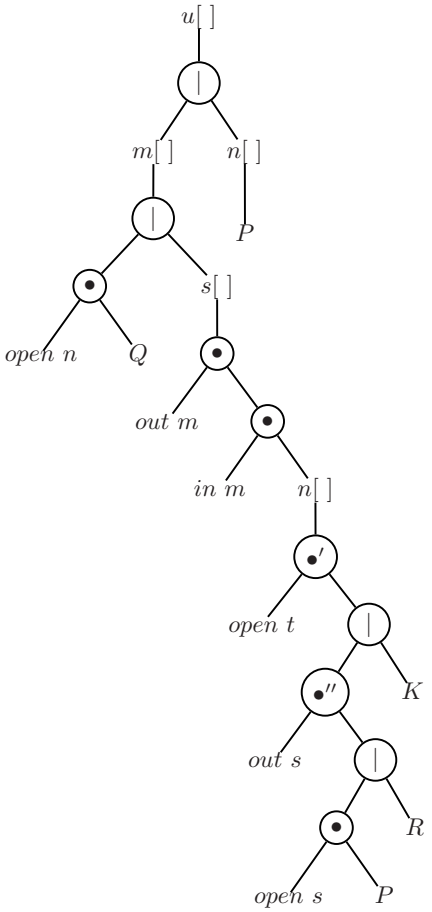

Fig. A.1. Syntax tree of the process A.2.

Observe that in our situation $\mathfrak{O}' = \{\bullet', \bullet''\}$ (see Figure 19). The space function sp for $\mathfrak{P} \cup \mathfrak{O}'$ 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):

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

The labeled syntax tree is in Figure 19.

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:

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

Note that f is not injective because $f(p) = f(p')$ and $f(\gamma) = f(\mu)$.

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:

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

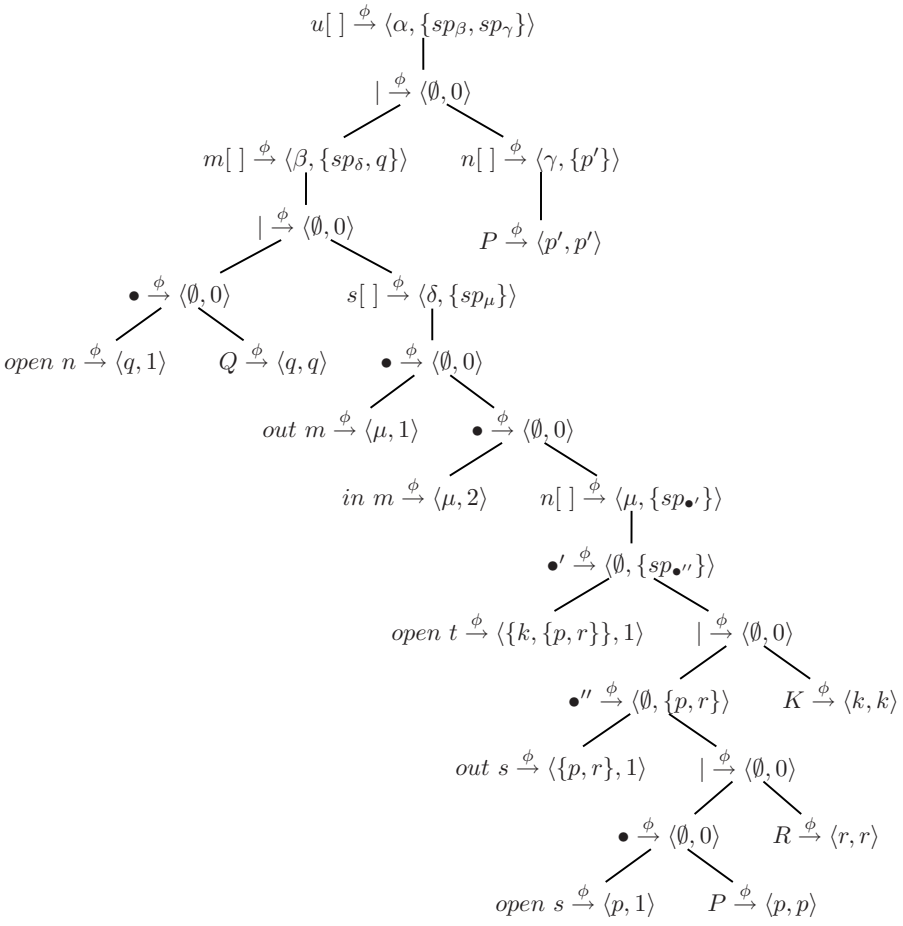

Fig. A.2. Labeled syntax tree of A.2.