UNIVERSITY
OF TRENTO

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

38050 Povo – Trento (Italy), Via Sommarive 14
http://www.dit.unitn.it

A PROPOSITIONAL BRANCHING TEMPORAL LOGIC FOR THE AMBIENT CALCULUS

Radu Mardare and Corrado Priami

October 2003

Technical Report # DIT-03-053

A Propositional Branching Temporal Logic for Ambient Calculus[†]

Radu Mardare and Corrado Priami
University of Trento

Abstract

We advocate the use of a CTL* logic, built upon the Ambient Calculus in dealing with mobile computing phenomena. 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. In applications, the possibility of expressing path properties of computation together with state properties opens new perspectives. Moreover, having a temporal logic to express properties of computation, we can reuse the algorithms for model checking temporal logics in analyzing the phenomena described using Ambient Calculus.

We resort to syntax trees of Ambient Calculus and enrich them with labelling functions that generates *the state processes*. These constitute a sound model for Ambient Calculus and are 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.

We provide the algorithms able to compute, giving the initial state of a system, any possible next state. These algorithms could be used together with the algorithms for model checking temporal logic in order to develop model checking analysis for Ambient Calculus.

1 Introduction

Ambient Calculus [6] is a useful tool to study mobility when processes may reside within a hierarchy of locations. Strongly based on Ambient Calculus was constructed Ambient Logic [5, 4], a logic that can describe properties of mobile computations as well as the hierarchy of locations and the modifications of this hierarchy in time. The main idea of Ambient Logic is treating processes as spatio-temporal entities thus were used two kinds of modalities - one for assertions about space and the other for assertions about time.

Our approach preserves the same spatio-temporal paradigm, but uses only the temporal modality. In [3] is proved that any structure of "*boxes inside boxes*" type can be uniquely described by a modal logic, but also by a flat system of equations in Set Theory. We choused to describe the spatiality of ambient processes by a flat system of equations that is, then, treated as a set of atomical propositions in a propositional logic[1]. To our logic are attached the temporal operators obtaining a branching propositional temporal logic. This logic, as argued bellow, is more expressive than Ambient Logic (being a temporal one, expresses path properties together with state properties), more comprehensive with respect to the intentional model (it distinguish between different processes which cannot be distinguished using Ambient Logic[2]), more simple (being one-dimensional and propositional) and more easy to use for complex analysis as model

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

[1]In order to do this we had to encode all the information in an ambient process in Set Theory and to prove that this encoding generates a sound model for Ambient Calculus.

[2]Our logic can make the difference between $P|Q|R$ and $P|c.(Q|R)$, but Ambient Logic cannot

checking (there are software already developed for this purpose, as SMV, NuSMV, SiMpLer, VIS).

The paper is organized as follows. We begin with a critical analysis of the expressivity of Ambient Logic pointing to the aspects that our approach improves. The next two sections introduce the main concepts of our approach, the labeled syntax tree and the state processes constructed for an ambient process. Section 5 introduce some basic notions of Set Theory, as the system ZFA^-, the flat systems of equations, and some classical results concerning the power of such systems to express hierarchies. Section 6 introduces a congruence relation over state processes that preserve the structural congruence over processes and it proves that the state processes constitute a sound model for Ambient Calculus. In section 7 we introduce our logic and in last section we implement it in order to ...

2 A short presentation of Ambient Calculus

We briefly recall the Ambient Calculus [6] starting with the syntax of ambient processes.

$P, Q, R ::=$	**processes**	$M ::=$	**capabilities**
$(\nu n)P$	restriction	n	name
0	void	$in\ M$	can enter into M
$P\|Q$	composition	$out\ M$	can exit out of M
$!P$	replication	$open\ M$	can open M
$M[P]$	ambient	$M.M'$	path
$M.P$	capability action	ϵ	null
$(n).P$	input action		
$\langle M \rangle$	output action		

Hereafter we assume that ambient programs can include unspecified processes denoted by capital letters P, Q, R, hereafter *atomical processes*[3]. Let Π be the class of atomical process names, and Λ the class of ambient names.

The structural congruence is defined as follows:

1. $P \equiv P$
2. $P \equiv Q \Rightarrow Q \equiv P$
3. $P \equiv Q, Q \equiv R \Rightarrow P \equiv R$
4. $P \equiv Q \Rightarrow (\nu n)P \equiv (\nu n)Q$
5. $P \equiv Q \Rightarrow P|R \equiv Q|R$
6. $P \equiv Q \Rightarrow !P \equiv !Q$
7. $P \equiv Q \Rightarrow n[P] \equiv n[Q]$
8. $(\nu n)(m[P]) \equiv m[(\nu n)P], \quad n \neq m$
9. $!(P|Q) \equiv !P|!Q$
10. $!0 \equiv 0, !P \equiv P|!P$
11. $!!P \equiv !P$
12. $P \equiv Q \Rightarrow M.P \equiv M.Q$
13. $P \equiv Q \Rightarrow (n).P \equiv (n).Q$
14. $P \equiv \varepsilon.P$
15. $(M.M').P \equiv M.M'.P$
16. $(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P$
17. $(\nu n)0 \equiv 0$
18. $(\nu n)P|Q \equiv P|(\nu n)Q, n \notin fn(P)$
19. $P|0 \equiv P$
20. $(P|Q)|R \equiv P|(Q|R)$
21. $P|Q \equiv Q|P$
22. $(x).P \equiv (y).P(x \leftarrow y)$ if $y \notin fn(P)$

and the reduction rules of the ambient calculus by:

$n[in\ m.P|Q]|m[R] \rightarrow m[n[P|Q]|R]$
$m[n[out\ m.P|Q]|R] \rightarrow n[P|Q]|m[R]$
$open\ n.P|n[Q] \rightarrow P|Q$
$(n).P|\langle M \rangle \rightarrow P\{n \leftarrow M\}$
$P \rightarrow Q \Rightarrow (\nu n)P \rightarrow (\nu n)Q$
$P \rightarrow Q \Rightarrow P|R \rightarrow Q|R$
$P \rightarrow Q \Rightarrow n[P] \rightarrow n[Q]$
$P' \equiv P, P \rightarrow Q, Q \equiv Q' \Rightarrow P' \rightarrow Q'$

[3]This is a necessary requirement in developing complex analysis, as 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 $n[in\ m.P]$

3 Ambient Logic

Ambient Logic is a modal logic constructed for Ambient Calculus and able to express properties of mobile computations as well as the hierarchy of location and the evolution of this hierarchy in time. The main idea of Ambient Logic is to treat ambient processes as spatio-temporal entities. Two modal operators were choused to express this universe: one for assertions about space, and other for assertions about time. As in the case of Spatial Logic, the satisfaction relation $P \models A$ means that the process P satisfy the closed formula A.

A logical constant, 0, was chosen to be satisfied by the null process 0, $0 \models 0$.

Naturally, we have $P \models \mathbf{T}$, ($P \models \neg A$ iff $\neg P \models A$), and ($P \models A \vee B$ iff $P \models A$ or $P \models B$).

We have logical propositions of the form $n[A]$, meaning that A holds at location n. These are satisfied by processes of the form $n[P']$ provided that P' satisfy A:

$$P \models n[A] \text{ iff } \exists P' \text{ such that } P \equiv_\alpha n[P'] \text{ and } P' \models A.$$

We have logical propositions of the form $A|B$, meaning that A and B hold contiguously, which are satisfied by contiguous processes of the form $P'|Q$ if P' satisfy A and Q satisfy B, or vice versa:

$$P \models A|B \text{ iff } \exists P', Q \text{ such that } P \equiv_\alpha P'|Q \text{ and } (P' \models A \text{ and } Q \models B) \text{ or } (P' \models B \text{ and } Q \models A).$$

Further, consider $P \downarrow P'$ meaning that $\exists n, P''$ such that $P \equiv_\alpha n[P']|P''$, and $\downarrow^*$ being the reflexive and transitive closure of the relation $\downarrow$. In the same way $\rightarrow^*$ means the transitive and reflexive closure of the reduction relation $\rightarrow$ of Ambient Calculus.

The spatial modality (*somewhere modality*) is introduced as $\Diamond_s A$ meaning that, inside of the process P that satisfies it, there is a process P' that satisfy A:

$$P \models \Diamond_s A \text{ iff } \exists P' \text{ such that } P \downarrow^* P' \text{ and } P' \models A$$

A temporal modality is added as well (*sometime modality*), $\Diamond_t A$, in order to express that in the future of the process satisfying it, a state will be reached that will satisfy the formula A:

$$P \models \Diamond_t A \text{ iff } \exists P' \text{ such that } P \rightarrow^* P' \text{ and } P' \models A$$

A location adjunct operator, $A@n$ proves its utility: $P \models A@n$ iff $n[P] \models A$. Other operators can be added as well, especially for express properties of computation involving the new names operator.

An important result is that the satisfaction is up to structural equivalence, i.e. if a process satisfies a formula, then any process structural equivalent with it will satisfy this formula as well:

$$(P \models A \wedge P \equiv_\alpha P') \Rightarrow P' \models A$$

3.1 Expressivity of Ambient Logic

Suppose we want to construct a model for the interaction between a firewall and an agent knowing the access passwords, using Ambient Calculus. We have the following definitions, [6]:

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

The interactions will be:

$$Agent|Firewall \equiv$$
$$(\nu n)(k'[open\ k.k''[Q]]|n[k[out\ n.in\ k'.in\ n.0]|open\ k'.open\ k''.P])$$
$$\rightarrow^* (\nu n)(k'[open\ k.k''[Q]]|k[in\ k'.in\ n.0]|n[open\ k'.open\ k''.P])$$

$$\rightarrow^* (\nu n)(k'[open\ k.k''[Q]|k[in\ n.0]]|n[open\ k'.open\ k''.P])$$
$$\rightarrow^* (\nu n)(k'[k''[Q]|in\ n.0]|n[open\ k'.open\ k''.P])$$
$$\rightarrow^* (\nu n)(n[k'[k''[Q]]|open\ k'.open\ k''.P])$$
$$\rightarrow^* (\nu n)(n[k''[Q]|open\ k''.P])$$
$$\rightarrow^* (\nu n)n[Q|P]$$

The problem that arises once we developed a mathematical model for a phenomenon concerns the success of this construction. In this particular case, it concerns the success of the agent to cross the firewall such that, formally, its process Q to be in parallel with P inside the ambient n. Such a property, being about Ambient Calculus computations, cannot be expressed using Ambient Calculus, but only using a logic in top of it. If we try to express such a property in Ambient Logic, we could only say that the expected relation is possible sometime in the future, using the temporal modality. But this do not exclude the possibility that for some possible temporal paths this situation to not be reached ever. In temporal logic, having quantifiers over paths together with quantifiers over moments, we can say that, for all possible paths there exists a moment when our property will be reached. And, only in this way, we can prove that, indeed, our model express correctly the desired property.

Consider now an example from Biology known as *the trimetric GTP binding proteins (G-proteins)* that plays an important role in the signal transduction pathway for numerous hormones and neurotransmitters [2]. It consists in five processes: a regulatory molecule RM, a receptor R, and three processes that are bounded together composing the protein α, β and γ. An information sent by RM to R determines a communication between the receptor R and the protein that generates the brake of the boundary of α, β and γ. We ca express this in Ambient Calculus by:

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

where we denoted by $(\alpha)(GDP \leftharpoondown GTP)$ the process obtained from α by substituting GDP with GTP inside α. If we try, using Ambient Logic, to express the spatial architecture of the ambient $Rm|R|Protein$ we will obtain the proposition $A|n[B|C]|D|E|F$, where $RM \models A$, $\langle GTP\rangle \models B$, $R \models C$, $\alpha \models D$, $\beta \models E$, and $\gamma \models F$. The spatial paradigm of this logic do not allows us to assert about the fact that α, β and γ are bound together and this boundary is blocking any reaction of them. The first possible interaction have to be between RM and R and have as result unlocking the protein complex.
More generally, Ambient Logic cannot distinguish between $P|Q|R$ and $P|c.(Q|R)$. If we have $P \models A$, $Q \models B$ and $R \models C$, then we have $P|Q|R \models A|B|C$ and $P|c.(Q|R) \models A|B|C$ without having $P|Q|R \equiv_\alpha P|c.(Q|R)$.

It is still obvious that there is a big spatial difference between the two processes. While in the case of the process $P|Q|R$ any interaction between these three is possible, in the case of the process $P|c.(Q|R)$, P cannot interact with Q or R, and any interaction between Q and R is forbidden until the capability c will be consumed. It seems that Q and R are bounded into a *theoretical box*, and even if this box is not an ambient, its action over the things inside it, is very similar with the ambient case. In the biological example presented before, we cannot express the fact that α, β and γ cannot interact and cannot move before RM and R interact each other. Only this interaction opens the boundary of the protein such that the three parts of it to be able to interact with the environment. This proves that the Ambient Logic is not

comprehensive enough with respect to the complexity of the phenomena described by Ambient Calculus.

For solving such a problem, a reconsideration of the spatial paradigm is required. As far as the Ambient Logic is not able, in some situations, to distinguish between processes that are not even alpha-congruent (but bisimilar), it seems that it is not an appropriate logic to be used for assumptions about the future of our processes, as we need if we want to perform complex analysis, as model checking, for Ambient Calculus.

The logic we want to propose here, comes to solve these problems and to open the perspective of model checking Ambient Calculus.

4 A Set Theoretical model for Ambient Calculus

4.1 Labeled syntax trees

In this section we define the *labeled syntax trees* for the ambient calculus processes starting from the syntax trees. Then, some abstractions of the labeled syntax trees will be used as the states in the logic we are going to construct.

We first consider only processes without new name operator (handled in the subsection 3.3).

A syntax tree $S = (S, \rightarrow_S)$ for an ambient 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 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 labels by two functions: *id* that gives to each node an identity, and *sp* that registers the spatial position of the node with respect to the underpinned structure of the process.

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

• 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

• to each capability, the identity of the process in front of which this capability is placed

• $\emptyset$, to each syntactical operator node

The spatial function *sp* associates:

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

• 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

[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 $n[in\ m.P]$

[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.

• to each syntactical operator 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 further 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 with the Anti-Foundation Axiom, ZFA, as being a fertile field that offers many tools for analyzing structures, as argued in [3]. Hereafter, we assume a class $\mathfrak{U}$ of urelements, set-theoretical entities which are not sets (they do not have elements) but can be elements of sets[7].

Definition 4.1. 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[8] including a.

Definition 4.2. The *support of a set* a, denoted by $supp(a)$ is $TC(a) \cap \mathfrak{U}$. The elements of $supp(a)$ are the urelements that are *somehow involved* in a.

Definition 4.3. If $a \subseteq \mathfrak{U}$ then $V(a) \stackrel{def}{=} \{b \mid b \text{ is a set and } supp(b) \subseteq a\}$. $V(a)$ is the class of all sets in which the only urelements that are somehow involved are the urelements of a.

Definition 4.4. A *decoration* of a graph $G = (G, \rightarrow_G)$ is an injective function
$e : G \rightarrow V(\mathfrak{U}) \cup \mathfrak{U}$ such that for all $a \in G$ we have:

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

Definition 4.5. 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 $n \rightarrow_S^* p \rightarrow_S^* m$

We now introduce a set of auxiliary functions that are the building blocks for *id* and *sp* (we present in Appendix the following constructions for an example).

Definition 4.6. Let the next functions be defined on the subsets of nodes of the syntax tree $(S, \rightarrow)$ as follows:

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

$$sp_{\mathfrak{O}}(s) = \begin{cases} sp_{\mathfrak{P}}(s) & \text{iff } s \in \mathfrak{O}' \\ 0 & \text{iff } s \in \mathfrak{O} \setminus \mathfrak{O}' \end{cases}, \text{ Consider } O \stackrel{def}{=} sp_{\mathfrak{O}}(\mathfrak{O}') \subset V(\mathfrak{U})$$

- Let $id_{\mathfrak{O}} : \mathfrak{O} \rightarrow V(\mathfrak{U}) \cup \mathfrak{U}$ defined by $\quad id_{\mathfrak{O}}(s) = \emptyset$
- Let $sp_{\mathfrak{C}} : \mathfrak{C} \rightarrow \mathbb{N}$ such that

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

[7] The urelements together with the empty set $\emptyset$ generates all the sets we work with (sometimes sets of sets)
[8] The existence of $TC(a)$ could be justified as follows: $TC(a) = \cup\{a, \cup a, \cup \cup a, ...\}$

- 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

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

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 can consider here natural numbers as cardinals[9] 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 still use natural numbers.

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 *the labeled syntax tree* for a given syntax tree of an ambient process.

Definition 4.7. 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 4.1. 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).

4.2 State processes

In this section we organize the information in the labeled syntax tree, as *the state process* associated with our ambient process. The state process will be a set-theoretical construct that will rearrange the information contained in the labeled syntax tree (implicitly in the associated ambient process) in a form easy to interpret and implement as state in our logic.

Definition 4.8. For a given labeled syntax tree $Sl = (S, \to, \phi)$ we define the functions:

- $ur : \mathfrak{P} \cup \mathfrak{O}' \to U_P \cup U_A \cup O$ 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 $e : U_P \cup U_A \cup O \to \mathfrak{V} \cup V(\mathfrak{V})$ be the function defined by

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

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

It associates to each urelement chosen for an ambient, atomical process or complex process, the set of addresses of its children.

• $f : U_P \cup U_A \cup O \to \Lambda \cup \Pi$, where Λ is the set of names of ambients of Ambient Calculus, and Π is the set of atomical process names[10]. 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[11], 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 (or the name $\langle 0, 0\rangle$ if it points to the complex processes in $\mathfrak{O}'$).

• $F : U_P \cup U_A \cup O \to \mathfrak{C}^*$ 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 equality relation, $=_{\mathfrak{C}}$, over capability chains[12]:

- $\langle c_1, c_2, c_3, ...c_n\rangle - \langle c_1\rangle =_{\mathfrak{C}} \langle c_2, c_3, ...c_n\rangle$,
- $\langle \varepsilon, c_1, ..., c_k\rangle =_{\mathfrak{C}} \langle c_1, c_2, ..., c_k, \varepsilon\rangle =_{\mathfrak{C}} \langle c_1, ..., c_t, \varepsilon, c_{t+1}, ...c_k\rangle =_{\mathfrak{C}} \langle c_1, c_2, ..., c_k\rangle$,
- $\langle \varepsilon, \varepsilon, ...\varepsilon\rangle =_{\mathfrak{C}} \emptyset$.

The function F associates with each urelement used to point to a process, the list of capabilities that exists in front of it.

Remark 4.2. Once constructed the function ϕ for our ambient process, the construction of the functions f, F and e is unique. Moreover, it is easy to verify that, giving an ambient process and the functions f, F and e of it, we can uniquely define the function ϕ of it.

Definition 4.9. Let $Sl_P = (S, \to, \phi)$ be the syntax tree associated with the ambient process P, and let $T_P = (\mathfrak{P} \cup \mathfrak{O}', \to_T)$ be the structure graph associated with it. We call *the labeled structure graph* of it the triplet $Tl_P = (\mathfrak{P} \cup \mathfrak{O}', \to_T, \psi)$ where ψ is the function defined on the nodes of the structure tree by:

$$\psi(s) = \langle ur_s, e(ur_s), f(ur_s), F(ur_s)\rangle.$$

Further we call, for short, by $\psi_P \stackrel{def}{=} \psi(\mathfrak{P} \cup \mathfrak{O}')$ the image by ψ of S, where, as before, $Sl_P = (S, \to, \phi)$. Moreover, because $ur(\mathfrak{P}_P) = U_P,\ ur(\mathfrak{P}_A) = U_A$ and $ur(\mathfrak{O}') = O$ we can describe ψ_P as

$$\psi_P = \langle U_A \cup O, U_P, e, f, F\rangle = \langle\langle U_A \cup O, U_P, e\rangle, f, F\rangle.$$

Definition 4.10. With respect to the previous definitions, we call *the state process* associated with P the ordered set $ST_P \stackrel{def}{=} \psi_P = \langle\langle U_A \cup O, U_P, e\rangle, f, F\rangle$. By extension, we can consider ψ as associating to each ambient process its state process.

Remark 4.3. In addition to remarks 4.1 and 4.2, we can observe that, being a process P, the construction of its state process is unique up to the choice of urelements, i.e. up to the choice of U_P and U_A, and it is made by assigning an urelement to each ambient process and to each atomical process (the rest of the construction being done, uniquely, by induction on the structure of the process).

[10] By accepting unspecified processes, we have to accept that the set Π contain more than the null name

[11] 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.

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

4.3 Handling the new name operators

It is now the moment to clarify the action of the new name operator. We propose an alternative to express, inside Ambient Calculus, the action of the new name operator without using it.

Consider the interaction between the firewall and the agent presented before, 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 $n[R]$.

Being the fact that the number of occurrences of new name operator in a process is finite[13], the action of it can be supplied by predefining a wellordered set of ambient names, subset of Λ. These names should never be used to name ambients in other situation but for new names. Each time the new name operator appears, we choose the next unused name from this set and replace it in all its occurrences inside the scope of the quantifier. In this way, we can supply the action of the new name operator without supplementary syntactical rules. This choice will be very useful for applications, as argued bellow.

Our proposal is to accept $\mathbb{N}\times\mathbb{N} \subset \Lambda$. In this way, by a trick that resembles de Bruijn indexes for name-free λ-calculus, we guarantee compositionality with respect to future constructions. 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 $\langle k, 1\rangle$ [14]. This approach allows us to combine our process with others. All the new 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 $\langle k, l\rangle$.

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 shows that the expected result is still possible without using the new name operator. Indeed:

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

Further we will treat $\langle l, k\rangle$ as any other ambient name, whenever it appears in our processes. This mean that we consider that the set Λ contains, as a subset, a subset of $\mathbb{N}\times\mathbb{N}$. This modification[15] does not affect the four rules of structural congruence ((Struct Res Res), (Struct

[13] In extenso, we can accept the denumerable possibility

[14] We replace n by $\langle k, 1\rangle$ in all its occurrences inside the scope of (νn), being ambients or capabilities

[15] It is not necessary to accept natural numbers as ambient names, but to define a function from $\mathbb{N}\times\mathbb{N}$ to Λ that will generate the ordered set of names required

Res Par), (Struct Res Amb) and (Struct Zero Res), see [6]). It modify only 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 natural numbers.*

In this way we reduce all the syntax trees of ambient calculus to syntax trees free of new name operators.

4.4 Hierarchical Structures as Hypersets

In this section we present the set theoretical tools used to analyze hierarchies. We define the notion of *flat system of equations*. These systems will be identified behind the spatial structure of any ambient process, and will be used to understand the relevance of structural congruence between processes from spatial point of view.

All our work is based on some results of Set Theory, first developed by F. Honsell and M. Forti (1983) [8], P. Aczel (1988) [1], and by J.Barwise and L.Moss (1996) [3]. The advantages of using the system ZFA is that it allows non-wellfounded sets (hypersets), that can describe recursive hierarchies in a finite manner, as we will show further. We use these properties, but concerning the economy of the paper, we will not present additional features of the system here. For more can be consulted the cited literature. Hereafter we speak about sets meaning also wellfounded (classical) sets and non-wellfounded sets (hypersets).

Definition 4.11. A *flat system of equations* is a triple $\mathcal{E} = \langle X, A, e\rangle$ with X and A disjoint sets (not necessarily sets of urelements) such that $X \cap A = \emptyset$, and a function $e : X \to \mathcal{P}(X \cup A)$. X is called the set of *indeterminates* of $\mathcal{E}$, A is called the set of *atoms* of $\mathcal{E}$. For each $v \in X$, the set $b_v \stackrel{def}{=} e_v \cap X$ is called the set of indeterminates on which v immediately depends. Similarly, the set $c_v \stackrel{def}{=} e_v \cap A$ is called the set of atoms on which v immediately depends (we wrote e_v for $e(v)$).

Example 4.1. *Consider the set $x = \{\alpha, \{\beta, \{\gamma\}\}\}$. This hierarchical structure can be described by a few one-level hierarchies $x = \{\alpha, y\}$, $y = \{\beta, z\}$, $z = \{\gamma\}$ or using a flat system of equations $\mathcal{E} = \langle X, A, e\rangle$ with $X = \{x, y, z\}$, $A = \{\alpha, \beta, \gamma\}$ and $e_x = \{\alpha, e_y\}$, $e_y = \{\beta, e_z\}$, $e_z = \{\gamma\}$. The solution of this system of equations describes our set.*

If our set have the definition $x = \{\alpha, \{\beta, \{x\}\}\}$, i.e. is a hyperset (having a recursive definition), it still can be expressed using a finite flat system of equations defined by $e_x = \{\alpha, e_y\}$, $e_y = \{\beta, e_z\}$, $e_z = \{e_x\}$. And the solution of this system describes our hyperset x.

Definition 4.12. A *solution* to $\mathcal{E}$ is a function s with domain X satisfying $s_x = \{s_y | y \in b_x\} \cup c_x$ for each $x \in X$.
The *solution-set* of a flat system $\mathcal{E}$ of equations is the set
$ss(\mathcal{E}) = \{s_v | v \in X\} = s[X]$ (we wrote $s[X]$ for the image of X by s).

The following result[16] is the one that allows our further construction. We will not present the technical set theoretical details here, they can be found in the cited literature.

Theorem 4.2. *In ZFA each set $a \in V[A]$ is a solution-set of a flat system of equations which has A as the set of atoms (or a subset of A) and any flat system of equations with the atoms from A have as unique solution-set, a set $a \in V[A]$.*

Further we introduce the relation of bisimulation over systems of equations, extending the one proposed in [3], which will help us to define the relation of congruence over state processes.

Definition 4.13. Let $A \subseteq \mho$, and let $\mathcal{E} = \langle X, A, e\rangle$ and $\mathcal{E}' = \langle X', A, e'\rangle$ be two flat systems of equations which use A as their set of atoms. An *A-bisimulation* relation between $\mathcal{E}$ and $\mathcal{E}'$is a relation $\mathfrak{R} \subseteq X \times X'$ such that whenever $x \mathfrak{R} x'$ the following conditions hold:

[16] This is proved and discussed in [3], part III, section 6

1. For every $y \in e_x \cap X$ there is an $y' \in e'_{x'} \cap X'$ such that $y \mathfrak{R} y'$
2. For every $y' \in e'_{x'} \cap X'$ there is an $y \in e_x \cap X$ such that $y \mathfrak{R} y'$
3. e_x and $e'_{x'}$ contain the same atoms, i.e. $e_x \cap A = e'_{x'} \cap A$.

The systems are *A-bisimilar*, written $\mathcal{E} \equiv_A \mathcal{E}'$, if there is an A-bisimulation relation between them such that for every $x \in X$ there is an $x' \in X'$ with $x \mathfrak{R} x'$ and vice verse.

Theorem 4.3. *The relation "$\equiv_A$" is an equivalence relation over flat systems of equations with the atoms in A.*

For the proof, see [3], p.81.

Theorem 4.4. *Let $\mathcal{E}$ and $\mathcal{E}'$ be flat systems of equations over the same set $A \in \mathfrak{V}$. $\mathcal{E}$ and $\mathcal{E}'$ have the same solution-sets if and only if they are bisimilar.*

For the proof, see [3], p.79.

For our purpose the bisimulation defined before is not enough. The previous bisimulation works on systems that describe the same sets. We need a relation of bisimulation able to identify a structure (hierarchy) up to the urelements involved in it. Thus we define a relation which considers bisimilar systems up to the choice of atoms as far as the systems have cardinal equivalent sets of atoms.

Definition 4.14. Let $A, A' \subseteq \mathfrak{V}$, and X, X' with $A \cap X = \emptyset$, $A' \cap X' = \emptyset$. Let $\mathcal{E} = \langle X, A, e \rangle$ and $\mathcal{E}' = \langle X', A', e' \rangle$ be two flat systems of equations. A *weak-bisimulation relation* between $\mathcal{E}$ and $\mathcal{E}'$ is a relation $\mathfrak{R} \subseteq X \times X'$ such that there is a bijective function $\zeta : B \to B'$ where $B, B' \subset \mathfrak{V}$, $A \subseteq B, A' \subseteq B'$, and whenever $x \mathfrak{R} x'$ the following conditions hold:

1. For every $y \in e_x \cap X$ there is an $y' \in e'_{x'} \cap X'$ such that $y \mathfrak{R} y'$
2. For every $y' \in e'_{x'} \cap X'$ there is an $y \in e_x \cap X$ such that $y \mathfrak{R} y'$
3. $\zeta(e_x \cap A) = e'_{x'} \cap A'$

We say that the systems are *weak-bisimilar*, and we write $\mathcal{E} \equiv_w \mathcal{E}'$ if there is a weak-bisimulation relation between them such that for every $x \in X$ there is an $x' \in X'$ with $x \mathfrak{R} x'$ and vice versa.

The meaning of this definition is that two weak-bisimilar systems describe the same structure up to the choice of the atoms.

Theorem 4.5. *An A-bisimulation is a weak-bisimulation.*

Proof. Indeed if we take ζ to be the identity function on A, the conditions of definition are easy to verify. □

Hereafter we assume $\zeta : \mathfrak{V} \to \mathfrak{V}$, by extending it with $\zeta(x) = x$ for all $x \in \mathfrak{V} \setminus B$. This allows us to prove the following theorem:

Theorem 4.6. *The relation $\equiv_w$ is an equivalence relation over flat systems of equations.*

Proof. The proof is immediate

- $\equiv_w$ is reflexive: the identity relation over X is a bisimulation if we take the identity over A as ζ.
- $\equiv_w$ is symmetric: suppose $\mathcal{E} \equiv_w \mathcal{E}'$. For proving that $\mathcal{E}' \equiv_w \mathcal{E}$ is enough to consider the inverse of the relation $\mathfrak{R}$ together with the inverse of ζ.

- $\equiv_w$ is transitive: suppose that $\mathcal{E} \equiv_w \mathcal{E}'$ by $\mathfrak{R}$ with ζ and $\mathcal{E}' \equiv_w \mathcal{E}''$ by $\mathfrak{R}'$ with ζ'. Then we have $\mathcal{E} \equiv_w \mathcal{E}''$ if we consider $\zeta'' = \zeta' \circ \zeta$ and $\mathfrak{R}'' = \mathfrak{R}' \circ \mathfrak{R}$.

□

We can extend the definition of ζ to the whole universe of set theory $\mho \cup V(\mho)$ by letting $\zeta(x) = \{\zeta(y) \mid y \in x\}$ for all $x \in V(\mho)$. This extension gives the possibility to prove:

Theorem 4.7. *Let $A, A' \subseteq \mho$ and $\mathcal{E} = \langle X, A, e\rangle$, $\mathcal{E}' = \langle X', A', e'\rangle$ be two flat systems of equations. Then $\mathcal{E} \equiv_w \mathcal{E}'$ iff $\zeta(ss(\mathcal{E})) = ss(\mathcal{E}')$, where ζ is the function used by the weak-bisimulation relation.*

Proof. We will prove this by induction on the complexity of the solution sets. The complexity of a set is used as the number of levels of set-nesting inside a given set. The urelements (i.e the atoms of our systems) have complexity 0.

- ($\Longrightarrow$) Suppose $\mathcal{E} \equiv_w \mathcal{E}'$, we will prove that $\zeta(ss(\mathcal{E})) = ss(\mathcal{E}')$, i.e. that $\{\zeta(s_x)|x \in X\} = \{s'_{x'}|x' \in X'\}$. These sets contain elements of different complexity. We will prove this equality by induction on the complexity. For urelements, for each $x \in A$ $\exists! x' \in A'$ *s.t.* $\zeta(x) = x'$ and vice versa. For complexity 1: $s_x = c_x \subseteq A$, so $\zeta(s_x) = c'_{x'}$ for sets of complexity 1. Suppose the same for the complexity k. Let $x \in X$ be an element of complexity $k+1$. Then $s_x = \{s_y|y \in b_x\} \cup c_x$, by definition, where y has the complexity k and the elements of c_x have the complexity 0. Now we can use the inductive hypothesis and we obtain the desired proof.

- ($\Longleftarrow$) Suppose $\zeta(ss(\mathcal{E})) = ss(\mathcal{E}')$, we will prove that $\mathcal{E} \equiv_w \mathcal{E}'$ by defining $\mathfrak{R} \subseteq X \times X'$ as $\{x, x'\} \in \mathfrak{R}$ iff $\zeta(s_x) = s_{x'}$. The requirements of the definition can be easily verified.

□

This theorem says that, as far as we can extend a bijection ζ between the set of atoms of the two systems to the whole universe of set theory in the way presented before, then the systems are bisimilar iff their solutions correspond each other by ζ.

4.5 The Congruence relation over state processes

Consider an ambient process P and its state process $ST_P = \langle\langle U_A \cup O, U_P, e\rangle, f, F\rangle$. In this context we can prove that:

Theorem 4.8. *$\mathcal{E} = \langle U_A \cup O, U_P, e\rangle$ is a flat system of equations with $U_A \cup O$ the set of indeterminates and U_P the set of atoms.*

This result allows us to identify the state processes by $ST_P = \langle \mathcal{E}, f, F\rangle$ where $\mathcal{E} = \langle U_A \cup O, U_P, e\rangle$ is the flat system of equations which describes the hierarchical structure of the ambient process.

Definition 4.15. Let P be an ambient process and ST_P be a state process for it. If $U_P^+ = f^{-1}(0) \cap F^{-1}(\varepsilon) \neq \emptyset$, we will denote by $ST_P^+ = \langle \mathcal{E}^+, f|_{(U_P \setminus U_P^+) \cup U_A \cup O}, F|_{(U_P \setminus U_P^+) \cup U_A \cup O}\rangle$, the state process obtained from ST_P by restricting the functions f respective F to $(U_P \setminus U_P^+) \cup U_A \cup O$ and by defining $\mathcal{E}^+ = \langle X, U_P \setminus U_P^+, e^+\rangle$, where $e_v^+ = e_v \setminus U_P^+$. We call ST_P^+ the *canonical state process of P*.

The introduction of canonical state processes it was necessary in order to simulate the null action of the process 0 in ambient calculus, when it appears without capabilities.

Definition 4.16. Let ST_1, ST_2 be two state processes and $ST_1^+ = \langle \mathcal{E}_1, f_1, F_1 \rangle$ and $ST_2^+ = \langle \mathcal{E}_2, f_2, F_2 \rangle$ be the canonical state processes for them, with $\mathcal{E}_1 = \langle U_A^1 \cup O^1, U_P^1, e_1 \rangle$, $\mathcal{E}_2 = \langle U_A^2 \cup O^2, U_P^2, e_2 \rangle$. We call the two state-processes *congruent*, and write $ST_1 \cong ST_2$, if the following conditions hold:

1. There exists two bijective functions
   $\zeta : U_P^1 \cup U_A^1 \cup O_1 \to U_P^2 \cup U_A^2 \cup O_2$ and $\mathfrak{Pr} : \Lambda \cup \Pi \to \Lambda \cup \Pi$ with the properties[17]:
   (a) $\zeta(U_P^1) = U_P^2$, $\zeta(U_A^1) = U_A^2$ and $\zeta(y) = \{\zeta(x) |\ for\ all\ x \in y\}$;
   (b) $\mathfrak{Pr}(n) = n$ for all $n \in \Lambda \cup \Pi \setminus (\mathbb{N} \times \mathbb{N})$
   (c) $\mathfrak{Pr}(f_1(\alpha)) = f_2(\zeta(\alpha))$ for all $\alpha \in U_P^1 \cup U_A^1 \cup O_1$
   (d) $\mathfrak{Pr}(F_1(\alpha)) =_{\mathfrak{C}} F_2(\zeta(\alpha))$ for all $\alpha \in U_P^1 \cup U_A^1 \cup O_1$
2. $\mathcal{E}_1 \equiv_w \mathcal{E}_2$ where the function used in the weak-bisimulation relation is ζ

The function ζ associates to each urelement of the first state process an urelement of the second one. This correspondence is meant to associate urelements that were chosen for ambients or atomical processes having the same name (in the case that this name is in $\mathbb{N} \times \mathbb{N}$ we take into consideration the possibility of renaming it by the function $\mathfrak{Pr}$). Moreover, these processes have to have the same chain of capabilities in front. The condition 2 ensures that the two ambient processes corresponding to the state processes have the same hierarchical structure.

Theorem 4.9. *The congruence relation over state processes is an equivalence relation.*

Proof. Let ST_1, ST_2, ST_3 be state-processes
Symmetry: if $ST_1 \cong ST_2$ (using ζ and $\mathfrak{Pr}$) then $ST_2 \cong ST_1$ (using ζ' and $\mathfrak{Pr}'$). We will define $\zeta' = \zeta^{-1}$, $\mathfrak{Pr}' = \mathfrak{Pr}^{-1}$.
Reflexivity: $ST \cong ST$. We define ζ as the identity over $U_P \cup U_A \cup O$ and $\mathfrak{Pr}$ as the identity over $(\mathcal{N} \times \mathcal{N}) \cap (f(U_P))$.
Transitivity: if $ST_1 \cong ST_2$ (using ζ_1 and $\mathfrak{Pr}_1$) and $ST_2 \cong ST_3$ (using ζ_2 and $\mathfrak{Pr}_2$) then $ST_1 \cong ST_3$ (using ζ' and $\mathfrak{Pr}'$). We define $\zeta' = \zeta_2 \circ \zeta_1$ and $\mathfrak{Pr}' = \mathfrak{Pr}_2 \circ \mathfrak{Pr}_1$ □

4.6 Algebra of State Processes

In this section we define an Algebra of state processes extending the compositional operations of Ambient Calculus from syntax trees to the class $\mathfrak{ST}$ of state processes. We start from the state processes of P_1 and P_2, and we will define those for $P_1|P_2$, $c_1.c_2...c_n.m[P_1]$ and $!P_1$. Such a construction, following the remark 4.1, can be reduced to the construction of the function *id* for each case, the rest being only an inductive construction.

Remark 4.4 *(Technical trick).* In order to treat the hierarchy of the ambient process as a set theoretical one, we have to consider the top-level ambients arranged in a top box - hereafter master ambient[18]. This mean that if we have to analyze a process as $P|Q$, we will analyze the process $u[P|Q]$, where u is the master ambient.

4.6.1 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 remark 4.1, we can suppose that the sets of urelements chosen for the two labeled trees are disjunct[19]. We can construct the syntax tree for $P_1|P_2$ using the

[17] Further we write $\mathfrak{Pr}(\langle c_1, c_2, ... \rangle)$ 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 $\mathfrak{Pr}(n)$

[18] This choice is sustained by the reduction rule, (Red Amb): If $P \to Q$ then $n[P] \to n[Q]$, of Ambient Calculus

[19] If this is not the case, because the labeled tree is unique up to the choice of the urelements, we can choose other urelements for P_1

rules of Ambient Calculus. All we have to do further is to define the function ϕ for the new syntax tree.

Suppose that $\alpha_1, \alpha_2 \in \mathfrak{U}$ 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 $\alpha \in \mathfrak{U}$ 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 $e_\alpha = e_{\alpha_1} \cup e_{\alpha_2}$. If $\langle \mathcal{E}_1, f_1, F_1 \rangle$ is a state process for P_1 and $\langle \mathcal{E}_2, f_2, F_2 \rangle$ is a state process for P_2, then the state process constructed in top of the labeled syntax tree defined before will be denoted by $\langle \mathcal{E}_1, f_1, F_1 \rangle \parallel \langle \mathcal{E}_2, f_2, F_2 \rangle$. So, we have $\parallel : \mathfrak{ST} \times \mathfrak{ST} \rightarrow \mathfrak{ST}$.

Theorem 4.10. *A state-process for $P_1|P_2$ is congruent with $\langle \mathcal{E}_1, f_1, F_1 \rangle \parallel \langle \mathcal{E}_2, f_2, F_2 \rangle$.*

Proof. Suppose that ST is a state-process for $P_1|P_2$. We will prove that $ST \cong \langle \mathcal{E}_1, f_1, F_1 \rangle \parallel \langle \mathcal{E}_2, f_2, F_2 \rangle$. We define ζ such that to associate the master ambient of $P_1|P_2$ with u, and the rest of urelements to the corresponding ones of $\langle \mathcal{E}_1, f_1, F_1 \rangle \parallel \langle \mathcal{E}_2, f_2, F_2 \rangle$ that point to the same ambients or atomical processes. In the same way we define the projection $\mathfrak{Pr}$ to associate the pairs of natural numbers that rename the same new names or the same name variables. With these definitions, the congruence can be easily prove. □

Theorem 4.11. *If $ST_{P_1} \cong ST_{P_2}$ and $ST_{Q_1} \cong ST_{Q_2}$ then $ST_{P_1} \parallel ST_{Q_1} \cong ST_{P_2} \parallel ST_{Q_2}$*

Proof. We define ζ as being ζ_P for urelements chosen for P_1, ζ_Q for urelements chosen for Q_1 and that associates the master ambient of $ST_{P_1} \parallel ST_{Q_1}$ with the one of $ST_{P_2} \parallel ST_{Q_2}$. $\mathfrak{Pr}$ will be the concatenation of $\mathfrak{Pr}_P$ and $\mathfrak{Pr}_Q$. With these definitions the congruence is immediate. □

Theorem 4.12. *If $ST_P \cong ST_Q$ then $ST_{P|R} \cong ST_{Q|R}$.*

Proof. Because $ST_P \cong ST_Q$ and $ST_R \cong ST_R$, using the previous theorem, we have $ST_P \parallel ST_R \cong ST_Q \parallel ST_R$. We proved that $ST_P \parallel ST_R \cong ST_{P|R}$ and $ST_Q \parallel ST_R \cong ST_{Q|R}$. Now, by transitivity, we obtain the desired relation. □

4.6.2 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, with the identity α_1. Let $\alpha, \beta \in \mathfrak{U}$ be two urelements unused in the labeled syntax tree of P. Let u be the master ambient of $c_1.c_2...c_k.m[P_1]$. We can define, in the 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:

$$id(u) = \alpha,\ id(m) = \beta,$$
$$id(n) = id_1(n) \text{ for all } n \in \mathfrak{P}_1 \setminus \{u\},\ id \text{ is not defined in } u_1$$

Of course, from the way ambient composition is defined, we have $e_\alpha = \{e_\beta\}$, and $e_\beta = e_{u_1}$. If $\langle \mathcal{E}_1, f_1, F_1 \rangle$ is a state-process for P_1, then a state-process constructed in top of the labeled syntax tree defined before is denoted by $c_1.c_2....c_n.m@\langle \mathcal{E}, f, F \rangle$. So, we have $c_1.c_2...c_k.m@ : \mathfrak{ST} \rightarrow \mathfrak{ST}$.

Theorem 4.13. *A state-process for $c_1.c_2....c_n.m[P]$ is congruent with $c_1.c_2....c_n.m@\langle \mathcal{E}, f, F \rangle$.*

Proof. Suppose that ST is a state-process for $c_1.c_2....c_n.m[P]$. We prove that $ST \cong c_1.c_2....c_n.m@\langle \mathcal{E}, f, F \rangle$. We define ζ as associating the master ambient of ST with the master ambient of $c_1.c_2....c_n.m@\langle \mathcal{E}, f, F \rangle$, and the urelements chosen for the same occurrence of the same atomical process or ambient in the both state processes. With these definitions the congruence is immediate. □

Theorem 4.14. *If $ST_P \cong ST_Q$ then $c_1...c_k.n@ST_P \cong c_1...c_k.n@ST_Q$.*

Proof. We define ζ' as being ζ for the urelements of P and that associates the urelement chosen for n in $c_1...c_k.n@ST_P$ with the one chosen for n in $c_1...c_k.n@ST_Q$. This definition verify the requirements of congruence relation. □

Theorem 4.15. *If $ST_P \cong ST_Q$ then $ST_{n[P]} \cong ST_{n[Q]}$*

Proof. Because $ST_P \cong ST_Q$, using the previous theorem, we have $n@ST_P \cong n@ST_Q$. We proved that $ST_{n[P]} \cong n@ST_P$ and $ST_{n[Q]} \cong n@ST_Q$. Now, by transitivity, we obtain the desired relation. □

4.6.3 Replication

Starting from the definition of $\|$: $\mathfrak{ST} \times \mathfrak{ST} \to \mathfrak{ST}$, we can define $\|^2$: $\mathfrak{ST} \to \mathfrak{ST}$ by $(\|^2\ P) \stackrel{def}{=} P \parallel P$.

Then, inductively, we can define for each $k \in \mathbb{N}$, $(\|^k\ P) \stackrel{def}{=} (\|^{k-1}\ P) \parallel P$. As a limit it is possible to define $\|^\infty\ P$, and this state process corresponds to $!P$.

Proposition 4.16. *The following statements are true:*

1. $ST_P \|^\infty \cong ST_P \parallel (ST_P \|^\infty)$, $ST_0 \|^\infty \cong ST_0$.
2. $(ST_P \|^\infty) \|^\infty \cong ST_P \|^\infty$.
3. *If $ST_P \cong ST_Q$ then $ST_P \|^\infty \cong ST_Q \|^\infty$.*
4. $ST_{P|Q} \|^\infty \cong (ST_P \|^\infty) \parallel (ST_Q \|^\infty)$

Proof. 1. If for ST_P we have $U_P \cup U_A = \{\alpha, \beta, \gamma, ...\zeta\}$ then for $ST_P \|^\infty$ we have $U_P^\infty \cup U_A^\infty = \{\alpha_1, \alpha_2, ...\beta_1, \beta_2, \beta_3, ...\gamma_1, \gamma_2, ...\zeta_1, \zeta_2, \zeta_3, ...\}$. We will consider a state process $ST_0 \cong ST_P$ with $U_P^0 \cup U_A^0 = \{\alpha_0, \beta_0, \gamma_0, ...\zeta_0\}$. So, $ST_P \parallel (ST_P \|^\infty) \cong ST_0 \parallel (ST_P \|^\infty)$. We can define ζ such that for all $\nu \in \{\alpha, \beta, \gamma, ...\zeta\}$ and all $i \in \{0, 1, 2, ...\}$, $\zeta(\nu_i) = \nu_{i+1}$. It is easy to verify that this is a congruence relation.

2. This can be reduced to the definition of a bijective function between $\mathbb{N} \times \mathbb{N}$ and $\mathbb{N}$. This can be done with Cantor's diagonal method.

3. Suppose that for ST_P we have $U_P \cup U_A = \{\alpha_P, \beta_P, ...\zeta_P\}$ and for ST_Q we have $U_P \cup U_A = \{\alpha_Q, \beta_Q, ...\zeta_Q\}$ with $\zeta(\nu_P) = \nu_Q$ for all ν. Suppose that for $ST_P \|^\infty$ we have $U_P^\infty \cup U_A^\infty = \{\alpha_P^1, \alpha_P^2, ...\beta_P^1, \beta_P^2...\delta_P^1, \delta_P^2, ...\}$ and for $ST_Q \|^\infty$ we have $U_P^\infty \cup U_A^\infty = \{\alpha_Q^1, \alpha_Q^2, ...\beta_Q^1, \beta_Q^2...\delta_Q^1, \delta_Q^2, ...\}$ Then we can establish the congruence between $ST_P \|^\infty$ and $ST_Q \|^\infty$ by defining a function $\zeta(\nu_P^i) = \nu_Q^i$. This function is defining a congruence.

4. The proof goes in the same way. □

As a consequence of the previous results we can state the following:

Theorem 4.17. *$ST_P \|^\infty$ is the state-process for $!P$.*

4.7 The soundness theorem

The next results prove that the class of ambient calculus processes with the structural equivalence relation is isomorph with the class of state processes with the congruence relation, by the function that associates to each ambient calculus program the state process constructed for it.

Proposition 4.18. *If we use the notation ST_P for the state-process of P, then the next assertions are true:*

1. $ST_P \cong ST_P$
2. $ST_P \cong ST_Q \Rightarrow ST_Q \cong ST_P$
3. $ST_P \cong ST_Q, ST_Q \cong ST_R \Rightarrow ST_P \cong ST_R$
4. $ST_P \cong ST_Q \Rightarrow ST_{(\nu n)P} \cong ST_{(\nu n)Q}$
5. $ST_{(\nu n)(m[P])} \cong ST_{m[(\nu n)P]}$ *when* $n \neq m$
6. $ST_P \cong ST_Q \Rightarrow ST_{M.P} \cong ST_{M.Q}$
7. $ST_P \cong ST_Q \Rightarrow ST_{(n).P} \cong ST_{(n).Q}$
8. $ST_P \cong ST_{\varepsilon.P}$
9. $ST_{(M.M').P} \cong ST_{M.M'.P}$
10. $ST_{(\nu n)(\nu m)P} \cong ST_{(\nu m)(\nu n)P}$
11. $ST_{(\nu n)0} \cong ST_0$
12. $ST_{(\nu n)P|Q} \cong ST_{P|(\nu n)Q}$ *if* $n \notin fn(P)$
13. $ST_{P|0} \cong ST_P$
14. $ST_{(P|Q)|R} \cong ST_{P|(Q|R)}$
15. $ST_{P|Q} \cong ST_{Q|P}$
16. $ST_{(x).P} \cong ST_{(y).P(x \leftarrow y)}$ *if* $y \notin fn(P)$

Proof. 1, 2, 3 these three rules are consequences of the theorem 5.1.

4. Suppose that in the formula $(\nu n)P$, n will be replaced by $\langle k, l\rangle$ (unused before), and in the formula $(\nu n)Q$, n will be replaced by $\langle s, t\rangle$ (unused before). Because $ST_P \cong ST_Q$, the function $\mathfrak{Pr}$ is associating all the new names and name variables of P with the new names and name variables of Q. We will extend this function by $\mathfrak{Pr}(\langle k, l\rangle) = \langle s, t\rangle$. With this extension we have $ST_{(\nu n)P} \cong ST_{(\nu n)Q}$

5. Suppose that we have the state-process for P already constructed. The state-process for $(\nu n)(m[P])$ will associate an urelement, α, to m and will replace all the occurrences of n by $\langle k, l\rangle$ (unused in ST_P). The state-process for $m[(\nu n)P]$ is associating to each occurrence of n the pair $\langle s, t\rangle$ (unused in ST_P) and to m an urelement β. Defining ζ as the identity for all urelements belonging to the state-process of P and $\zeta(\alpha) = \beta$, and defining $\mathfrak{Pr}(\langle k, l\rangle) = \langle s, t\rangle$ and as identity in rest, we obtain the desired relation.

6. Suppose that $\{\alpha_1, \alpha_2, ...\alpha_k\}$ is the top-level structure of the process P and that $\{\beta_1, \beta_2, ...\beta_k\}$ is the same for Q. If $F_P(\{\alpha_1, \alpha_2, ...\alpha_k\}) = \langle c_1, c_2, ...c_n\rangle$ (and, by the definition of bisimulation, $F_Q(\{\beta_1, \beta_2, ...\beta_k\}) = \langle c_1, c_2, ...c_n\rangle$), we define $F_{M.P}(\{\alpha_1, \alpha_2, ...\alpha_k\}) = \langle M, c_1, c_2, ...c_n\rangle$. If F_P is not defined in $\{\alpha_1, \alpha_2, ...\alpha_k\}$ then we define $F_{M.P}(\{\alpha_1, \alpha_2, ...\alpha_k\}) = \langle M, \varepsilon\rangle$ (and, by definition, $F_{M.Q}(\{\beta_1, \beta_2, ...\beta_k\}) = \langle M, \varepsilon\rangle$). Now the requirements of the definition are fulfilled.

7. This proof goes in the same way as the one for the previous rule.

8. This goes immediately from the fact that $\langle c_1, ..., c_k\rangle = \langle \varepsilon, c_1, ..., c_k\rangle$

9. This goes as 6.

10. Suppose that in $(\nu n)(\nu m)P$, n is replaced by $\langle k, l\rangle$ and m is replaced by $\langle k+1, l\rangle$, then in $(\nu m)(\nu n)P$, m will be replaced by $\langle k, l\rangle$ and n by $\langle k+1, l\rangle$. We define $\mathfrak{Pr}(\langle k, l\rangle) = \langle k+1, l\rangle$ and $\mathfrak{Pr}(\langle k+1, l\rangle) = \langle k, l\rangle$ and identity in rest. The conditions of the definition of congruence can be easy verify.

11. Indeed there is no occurrence of n in 0, i.e we will cut (νn) and no modification will be done more.

12. When the new name will be replaced by $\langle k, l\rangle$ in the first process the replacement will be done only inside Q because there is no free occurrence of n in P. We will obtain the same formula as for the second process. The congruence will be a consequence of the property of reflexivity

13. The canonical processes for the two programs are identical.

14. We define ζ such that the urelements corresponding to the same occurrences of the same ambients or atomical processes in P, Q respective R, in the two state-processes, to correspond each other.

15. idem.

16. If x and y are name variables, then they will be replaced by $\langle k, l\rangle$ respective $\langle s, t\rangle$, and all we have to do is to define $\mathfrak{Pr}$ such that these two to correspond each other. If x and y are variables over capabilities, then the property is true by the convention we made before: $\langle c_1, ...c_t, x, c_{t+1}, ..., c_k\rangle = \langle c_1, ...c_t, y, c_{t+1}, ..., c_k\rangle$ □

Theorem 4.19. *If we use the notation ST_P for the state-process of P, then*

$$P \equiv_\alpha Q \text{ iff } ST_P \cong ST_Q.$$

Proof. ($\Longrightarrow$) If $P \equiv_\alpha Q$ then $ST_P \cong ST_Q$.
For proving this, it is enough to prove that the dual rules (for state-processes with the congruence operator) of the 22 rules that define the structural congruence over ambient processes, are true. But all these are already proved. Indeed the dual expressions of the rules 1-4 were proved in proposition 5.3, 1-4, the dual of the rule 5 was proved in theorem 6.3. The dual of the rules 6,9-11 were proved in the proposition 6.7, the dual of 7 in the theorem 6.6 and the dual of the rules 8, 12-22 were proved in the proposition 5.3, 5-16.

($\Longleftarrow$) If $ST_P \cong ST_Q$ then $P \equiv_\alpha Q$.
Let ST_P, ST_Q be two congruent state-processes, $\mathcal{S}_P^+ = \langle \mathcal{E}_P, f_P, F_P \rangle$ and $\mathcal{S}_Q^+ = \langle \mathcal{E}_Q, f_Q, F_Q \rangle$ be the canonical state-processes for them, with $\mathcal{E}_P = \langle X_P, A_P, e_P \rangle$, $\mathcal{E}_Q = \langle X_Q, A_Q, e_Q \rangle$.
This proof have to be done by induction on the complexity of the flat systems, and of the functions F (i.e. by induction on the structure of the ambient processes involved). We will try to make the reverse construction, starting from these state-processes, we will identify the ambient calculus processes that correspond to them, and we will prove that these are structural congruent.
The function ζ is associating to each urelement that points to an ambient of ST_P the urelement of ST_Q that points to an ambient with the same name, and the same is happening for atomical processes. Because $\mathcal{E}_P \cong_w \mathcal{E}_Q$, we have that ST_P and ST_Q have the same *boxes inside boxes* structure. Using the rules 5, 20 and 21 of structural congruence, we obtain that the structural congruence is up of the way of arranging objects inside a box. Going up, with the rule 7, boxes having the same names and containing structural congruent objects are still structural congruent. Now, using the rules 12, 14 and 15 we can be sure that adding the same lists of capabilities in the same places of our structures, even for associations of objects (as in situations when some objects are bounded together by brackets), the processes remain still structural congruent. The rules 4, 8, 16, 17 and 18 preserve the structural congruence when we identify the new names of our processes. We will use the function $\mathfrak{Pr}$ for identifying the new names that correspond each other in the two processes. Now the rules 13 and 22 allow us to do the same for input, still preserving the structural congruence. The rules 9, 10 and 11 allow us to use replication operator still preserving the structural congruence. And last, we can, by adding the process 0, to go from canonical processes up, preserving, still, the structural congruence.
In this way we can be sure that our processes are structural congruent. □

Corollary 4.20. *If ST_1 and ST_2 are two state processes constructed for the same ambient process, then $ST_1 \cong ST_2$.*

So, the construction of a state process for a given ambient process is unique up to congruence.

Theorem 4.21 (Soundness Theorem). *If we denote by $\mathfrak{ST}$ the class of state-processes, then $\langle \mathfrak{ST}, \cong \rangle$ is a sound model for Ambient Calculus.*

Proof. This result is a direct consequence of the previous proposition. □

5 The Logic

The logic we intend to construct is a branching propositional temporal logic, CTL^*[20]. The requirements of such a construction [7] are to organize a structure $\mathfrak{M} = (S_0, \mathfrak{S}, \mathfrak{R}, \mathfrak{L})$ where S_0 is an 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 will associate to each state $S \in \mathfrak{S}$ a set of atomical propositions $\mathfrak{L}(S) \subseteq \mathcal{P}(\mathfrak{Ap})$ - the set of true atomical propositions in the state S ($\mathfrak{Ap}$ will be the class of atomical propositions).

We developed the state processes 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

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

calculus process P, 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 on 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 $m@$).

Definition 5.1. Assume that $ST_0 = \langle \mathcal{E}_0, f_0, F_0 \rangle$ with $\mathcal{E}_0 = \langle U_A^0 \cup O^0, U_P^0, e_0 \rangle$ is our initial state. Then we define $\mathfrak{S}$ for all state processes $ST = \langle \mathcal{E}, f, F \rangle$ with $\mathcal{E} = \langle U_A \cup O, U_P, e \rangle$ by

$\mathfrak{S} \stackrel{def}{=} \{ST \mid U_P \subseteq U_P^0,\ O \subseteq O_0, U_A \subset U_A^0 \text{ and } f = f_0|_{U_A \cup O \cup U_P}\}$.

The set $\mathfrak{S}$ consists in all the state processes of all ambient processes that contain the same ambients and atomical processes as P, with the same identities, but in possible different spatial structure[21]. In extenso we will accept that the possible states have $U_P = U_P^0,\ O = O_0$, and $U_A = U_A^0$[22].

Definition 5.2. Now we consider the set of atomical propositions defined as:

$\mathfrak{Ap} = \{x \mathfrak{in} y | x \in U_P \cup U_A \cup O \text{ and } y \in U_A \cup O\}$.

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 $card(U_P) \times card(U_A) \times card(O)$ which depends (polynomial) on the number of atomical processes and ambients in the initial ambient process.

Definition 5.3. We define the interpretation function $\mathfrak{L} : \mathfrak{S} \to \mathcal{P}(\mathfrak{Ap})$ by:

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

Definition 5.4. We define the accessibility relation $\mathfrak{R} \subseteq \mathfrak{S} \times \mathfrak{S}$ as it follows:
if $ST_P, ST_Q \in \mathfrak{S}$ are the state processes for the ambient processes P and Q, then $\langle ST_P, ST_Q \rangle \in \mathfrak{R}$ iff $P \to Q$. (i.e. Q can be reached from P in one step of ambient calculus reduction - this explains why $ST_P, ST_Q \in \mathfrak{S}$).

5.1 Syntax

Following the classic way of introducing CTL^* we define a *fullpath* as an infinite sequence $S_0, S_1, ...$ of states such that $(S_i, S_{i+1}) \in \mathfrak{R}$ for all i[23].

Further, we introduce the syntax of the CTL* logic in the usual way [7]. 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 have the classical logic operators - $\wedge$ and $\neg$ - together with the temporal operators X (*next time*) and $\cup$ (*until*). We have, as well, the path quantifier E (*for some futures*). From these we can derive the temporal operators G (*always*) and F (*sometimes*), and the path quantifier A (*for all futures*). The propositions of this logic can be satisfied by processes, or by sequences of processes (as a computational path). The syntactical rules are the classical ones for CTL* [7].

Syntactical rules:

1. Each atomical proposition $\alpha \mathfrak{in} \beta \in AP$ is a state formula
2. If p, q are state formulae then so are $p \wedge q$, $\neg p$
3. If p is a path formula then $E\ p$, $A\ p$ are state formulae

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

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

[23] We use the convention that if $x = (S_0, S_1, ...)$ denotes a fullpath, then x^i denotes the suffix path $(S_i, S_{i+1}, S_{i+2}, ...)$

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:

1. Ap abbreviates $\neg E \neg p$.
2. EFp abbreviates $E(true \cup p)$.
3. AGp abbreviates $\neg EF \neg p$.
4. AFp abbreviates $A(true \cup p)$.
5. 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 $\exists i \left(\mathfrak{M}, x^i \models q \text{ and } \forall j \left(j < i \text{ implies } \mathfrak{M}, x^j \models p\right)\right)$

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

Definition 5.5. 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 structural congruence.

Theorem 5.1. *Let P_1, P_2 be two ambient processes. Then the next assertions are equivalent:*

1. $P_1 \equiv_\alpha P_2$
2. *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*
   $\zeta : U_P^1 \cup U_A^1 \cup O_1 \to U_P^2 \cup U_A^2 \cup O_2$ *and* $\mathfrak{Pr} : \Lambda \cup \Pi \to \Lambda \cup \Pi$ *with the properties*[24]*:*

[24] As before we write $\mathfrak{Pr}(\langle c_1, c_2, ...\rangle)$ 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 $\mathfrak{Pr}(n)$

i. $\zeta(U_P^1) = U_P^2$, $\zeta(U_A^1) = U_A^2$ *and* $\zeta(y) = \{\zeta(x) |$ *for all* $x \in y\}$;
ii. $\mathfrak{Pr}(n) = n$ *for all* $n \in \Lambda \cup \Pi \setminus (\mathbb{N} \times \mathbb{N})$
iii. $\mathfrak{Pr}(f_1(\alpha)) = f_2(\zeta(\alpha))$ *for all* $\alpha \in U_P^1 \cup U_A^1 \cup O_1$
iv. $\mathfrak{Pr}(F_1(\alpha)) =_{\mathfrak{C}} F_2(\zeta(\alpha))$ *for all* $\alpha \in U_P^1 \cup U_A^1 \cup O_1$

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

Proof. We already proved that $1 \Leftrightarrow 2$. For proving that $2 \Leftrightarrow 3$ we observe that the clauses of 3 are the same with those from the definition of congruence, excepting the clause b. So, all we have to prove is that the clause b is equivalent with the fact that the two flat systems are weak-bisimilar. But this is immediate from the way we define the semantics for atomical sentences of our logic, starting from the equations of the flat system associated with it. □

5.3 Applying the logic

We reconsider here the examples discussed on the beginning of the paper to see how the logic proposed is solving the discussed problems[25].

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

$$(\nu n)(k'[open\ k.k''[Q]] | n[k[out\ n.in\ k'.in\ n.0] | open\ k'.open\ k''.P]) \tag{5.1}$$

In order to construct the state process for it we will wrap it in a master ambient u and we replace all the occurrences of the new name n by $\langle 1,1\rangle$:

$$u[k'[open\ k.k''[Q]] | \langle 1,1\rangle[k[out\ \langle 1,1\rangle.in\ k'.in\ \langle 1,1\rangle.0] | open\ k'.open\ k''.P]] \tag{5.2}$$

For 5.2 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 $\alpha, \beta, \kappa, \kappa', \kappa'', p, q, o \in \mathfrak{V}$. So, $U_A = \{\alpha, \beta, \kappa, \kappa', \kappa''\}$, $U_P = \{q, p, o\}$, $O = \emptyset$, and f is defined by: $f(\alpha) = u$, $f(\beta) = \langle 1,1\rangle$, $f(o) = 0$, $f(\kappa) = k$, $f(\kappa') = k'$, $f(\kappa'') = k''$, $f(q) = Q$, $f(p) = P$. Following the definition of e we obtain the flat system:

$$\begin{array}{lcl}
e_\alpha = \{e_{\kappa'}, e_\beta\} & \Longrightarrow & \left\{ \begin{array}{l} e_{\kappa'} \in e_\alpha \\ e_\beta \in e_\alpha \end{array} \right. \Longrightarrow \left\{ \begin{array}{ll} \kappa' \mathfrak{in} \alpha & \text{is true} \\ \beta \mathfrak{in} \alpha & \text{is true} \end{array} \right. \\
e_{\kappa'} = \{e_{\kappa''}\} & \Longrightarrow & \{\ e_{\kappa''} \in e_{\kappa'} \Longrightarrow \{\ \kappa'' \mathfrak{in} \kappa' \quad \text{is true} \\
e_\beta = \{e_\kappa, p\} & \Longrightarrow & \left\{ \begin{array}{l} e_\kappa \in e_\beta \\ p \in e_\beta \end{array} \right. \Longrightarrow \left\{ \begin{array}{ll} \kappa \mathfrak{in} \beta & \text{is true} \\ p \mathfrak{in} \beta & \text{is true} \end{array} \right. \\
e_{\kappa''} = \{q\} & \Longrightarrow & \{\ q \in e_{\kappa''} \Longrightarrow \{\ q \mathfrak{in} \kappa'' \quad \text{is true} \\
e_\kappa = \{o\} & \Longrightarrow & \{\ o \in e_\kappa \Longrightarrow \{\ o \mathfrak{in} \kappa \quad \text{is true}
\end{array}$$

In this way we obtained the list of true atomical propositions. In *table1* we can see a map of all set of atomical propositions, $\mathfrak{Ap}$, for this state. This matrix has one line for each element of $U_A \cup O$, one column for each element of $U_P \cup U_A \cup O$, and is constructed by setting the entry of column x and row y to 1, if the proposition $x \mathfrak{in} y$ is true. All the empty entries are set to 0.

The function F have the values:
$F(\alpha) = \langle \varepsilon, \varepsilon, ...\rangle$, $F(\beta) = \langle \varepsilon, \varepsilon, ...\rangle$, $F(o) = \langle out\langle 1,1\rangle, in\ k', in\langle 1,1\rangle, \varepsilon\rangle$, $F(\kappa) = \langle \varepsilon, \varepsilon, ...\rangle$, $F(\kappa') = \langle \varepsilon, \varepsilon, ...\rangle$, $F(\kappa'') = \langle open\ k, \varepsilon, ...\rangle$, $F(q) = \langle \varepsilon, \varepsilon, ...\rangle$, $F(p) = \langle open\ k', open\ k'', \varepsilon, ...\rangle$.

[25] For these examples we will construct directly the state processes. We skip the intermediary steps. For a extended construction can be consulted the Appendix.

Table1	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

Table2	α	p	q	r	$\{q,r\}$
α	0	1	0	0	1
$\{q,r\}$	0	0	1	1	0

Table3	α	p	q	r
α	0	1	1	1

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 n is a child of the master ambient $u = f(\alpha)$, $Q = f(q)$ and $P = f(p)$ are children of $\langle 1, 1\rangle = f(\beta)$. Further, for checking the truth value of this statement, a model checker could be use. Proving that our logical formula is true it finally means that our mathematical model 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[26].

Further we will show how our logic makes the distinction between $u[P|Q|R]$ and $u[P|c.(Q|R)]$. Suppose that in both cases we choose $f(\alpha) = u$, $f(p) = P$, $f(q) = Q$ and $f(r) = R$. The first process will satisfy the logical proposition A, while the second will satisfy B:

$$A := pin\alpha \wedge qin\alpha \wedge rin\alpha, \quad B := pin\alpha \wedge \{q,r\}in\alpha \wedge qin\{q,r\} \wedge rin\{q,r\}.$$

This difference is possible because in this case for both processes $U_P = \{p, q, r\}$ and $U_A = \{\alpha\}$, but for the first one $O = \emptyset$ while for the second $O = \{\ \{q,r\}\ \}$. In the *Table2* can be seen the set of atomical propositions for the process $u[P|c.(Q|R)]$. Observe that in this case, the elements of O are in the propositions as well. In *Table3* the same atomical propositions, but for the process $u[P|Q|R]$.

6 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] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Molecular Biology of the Cell.* Garland Publishing, Inc., fourth edition, 2002.

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

[26] We performed this analysis, for our example, in[9] where can be found all the technical information concerning this implementation in the NuSMV software and the resources consumed to perform it

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

[5] 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.

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

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

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

[9] R. Mardare and C. Priami. A logical approach to security in the context of ambient calculus. *to appear in Electronic Notes in Computer Science, Elsevier*, available at http://dit.unitn.it/ mardare/publications.htm, 2003.

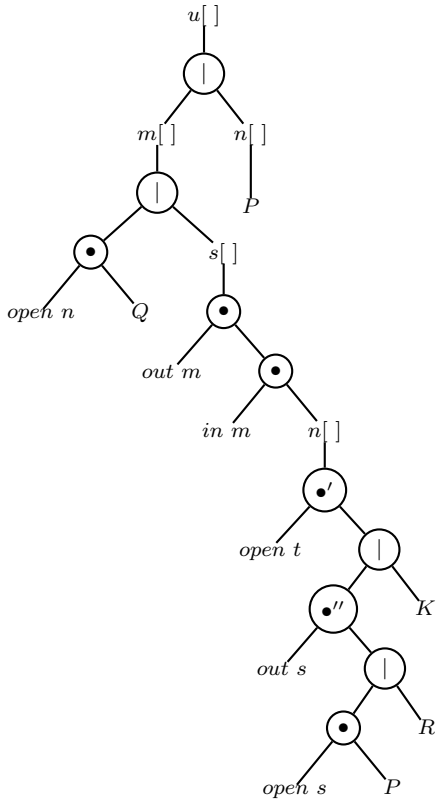

Figure 1: Syntax tree of the process A.2.

A The construction of a state process

We present further the construction of a labeled syntax tree first and then of a state process for a given ambient process.

Consider the ambient calculus process:

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

As a general rule, we embed our program into a *master ambient*[27] (the master ambient will have a fresh name). Our program becomes:

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

The syntax tree of this process is in Figure A.

First step is the definition of 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 $\{\alpha, \beta, \gamma, \delta, \mu, p, q, p', r, k\} \subset \mathfrak{U}$.

Observe that in our situation $\mathfrak{O}' = \{\bullet', \bullet''\}$ (see Figure A). 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:

[27] This is a technical trick that is not disturbing our analysis because of the rule (RedAmb): $P \to Q \Rightarrow n[P] \to n[Q]$, [6], but it helps to treat the processes as a whole from the spatial point of view.

$sp(u) = \{sp(m), sp(n)\}$ (here n is the child of u), $sp(m) = \{sp(s), q\}$, $sp(n) = \{p'\}$ (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 $\bullet \in \mathfrak{O} \setminus \mathfrak{O}'$, $\phi(\bullet) = \langle \emptyset, 0\rangle$, for all $| \in \mathfrak{O}$, $\phi(|) = \langle \emptyset, 0\rangle$,

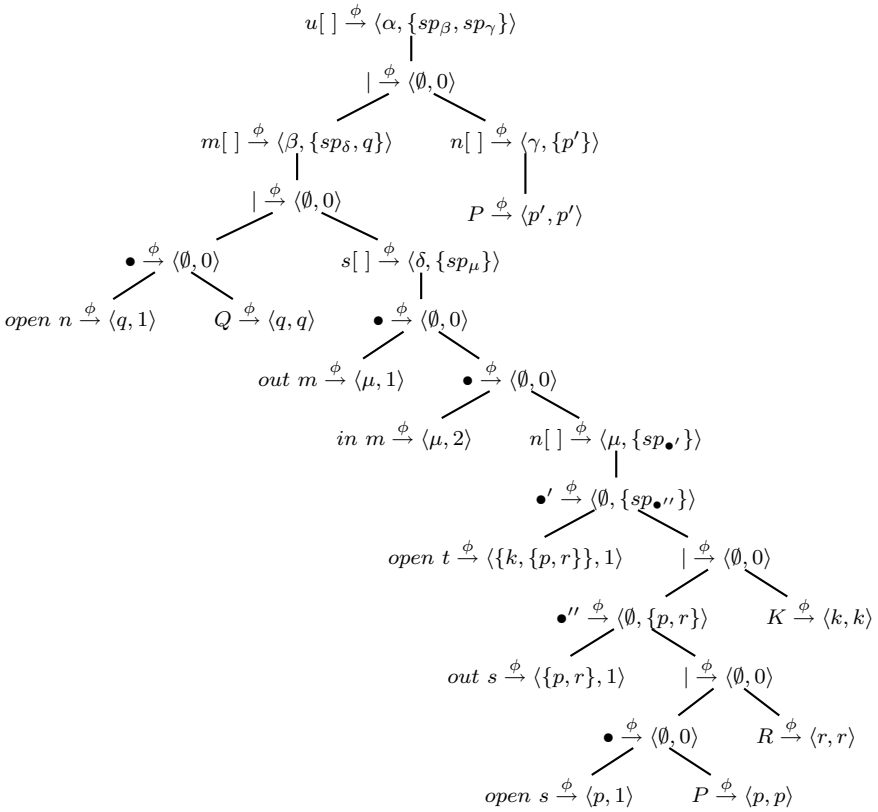

Figure 2: Labeled syntax tree of A.2.

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:

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

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:

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