Types for Role-Based Access Control of Dynamic Web Data

Types for Role-Based Access Control of Dynamic Web Data

Mariangiola Dezani-Ciancaglini[1], Silvia Ghilezan[2],
Svetlana Jakšić[2], Jovanka Pantović[2]

[1] Dipartimento di Informatica, Università di Torino, Italy
dezani@di.unito.it
[2] Faculty of Technical Sciences, University of Novi Sad, Serbia
{gsilvia,sjaksic,pantovic}@uns.ac.rs

Abstract. We introduce a role-based access control calculus for modelling dynamic web data and a corresponding type system. It is an extension of the $\mathrm{X}d\pi$ calculus proposed by Gardner and Maffeis.

In our framework, a network is a parallel composition of locations, where each location contains processes with roles and a data tree whose edges are associated with roles. Processes can communicate, migrate from a location to another, use the data, change the data and the roles in the local tree. In this way, we obtain a model that controls process access to data.

We propose a type system which ensures that a specified network policy is respected during computations. Finally, we show that our calculus obeys the following security properties: (1) all data trees and processes with roles in a location agree with the location policy; (2) a process can migrate only to a location with whose policy it agrees; (3) a process with roles can read and modify only data which are accessible to it; (4) a process with roles can enable and disable roles in agreement with the location policy.

1 Introduction

One of the essential steps in managing distributed systems with semi-structured data is the security administration, which is one of the main features in prevention of unauthorized access to system resources. Role-based access control (RBAC) [24] is an access control method that relies on the notions of users, roles and permissions. It controls the access of users to the system resources in accordance with the activities they have to perform in the system. In accessing the system resources, a user has those permissions which are assigned to its roles. In a system, roles are statically defined by the organisation structure, hence the security administration is reduced to the management of permissions. This makes RBAC a simplified and desirable access control technology.

With the aim of application of RBAC to peer-to-peer model of semi-structured web data, we introduced the Ⓡ$\mathrm{X}d\pi$ calculus, a formal model for dynamic web applications with RBAC, and proposed a type system to control its safety. The Ⓡ$\mathrm{X}d\pi$ calculus is a role-based extension of the $\mathrm{X}d\pi$ calculus of Gardner and Maffeis [12]. The $\mathrm{X}d\pi$ calculus models process communication, and process migration, as distributed π-calculus, and local interaction between processes and data. A network is a parallel composition of locations, where each location contains a process and a data tree. In our calculus, by assigning roles both to data trees and processes, we obtain a model that allows role administration (i.e. activation and deactivation). Each location has its policy which consists of the *data accessibility policy*, the set of minimal roles that a process needs to have to access the data, and the *administration policy*, the sets of roles which can be enabled or disabled at that location.

As a first contribution of this paper, in Section 2, we design $\mathrm{X}d\pi$ in a RBAC scenario and extend it with commands for role administration, i.e., for enabling and disabling roles. More precisely, the command $\mathtt{enable}_p(r)$ allows the role r to access data identified by the path p and the command $\mathtt{disable}_p(r)$ forbids the role r to access identified data.

As a second contribution of this paper, in Section 3, we propose a type system for Ⓡ$\mathrm{X}d\pi$ and prove that a specified network policy is respected during computations. In our framework, a well-typed network is a parallel composition of well-typed locations with different names and if a location is well typed, then both the enclosed tree and process with roles do not contain occurrences of free variables.

Our calculus is enriched with a type system assuring that:

- if a process can access an edge in a well-typed tree, then the edge is connected to the root of the tree by a path whose edges are all accessible to that process;
- only processes agreeing with the location policy can be activated at a location and can migrate to it;
- a process can modify a subtree only if it can access all the edges of the subtree;
- agreeing with the location policy, a process can enable a role at an edge or disable a role from a subtree if it can access the path which identifies it.

Finally, in Section 4, we prove that the type system obeys the following security properties:

- all data trees and processes with roles in a location agree with the location policy. This holds even for processes with roles generated by reading data or activating scripts and for trees obtained by changing data or activating/deactivating roles;
- a process can migrate only to a location with whose policy it agrees;
- a process with roles can read and modify only data which are accessible to it;
- a process with roles can enable and disable roles in agreement with the location policy.

The main contribution of the present paper is the explicit dynamic association of roles to data in agreement with a distributed policy which enables resource access control.

Outline of the paper. In Section 2 we introduce the syntax and the reduction rules for the Ⓡ$\mathrm{X}d\pi$ calculus. Section 3 is devoted to its type system. Security properties of the system are stated in Section 4. The proofs are the content of the Appendix.

1.1 An example

As an example, we use a university campus (network) containing locations such as a faculty, a classroom and a public space, represented by:

$$\mathtt{FACULTY}[\![\, T_{\mathtt{F}} \parallel R_{\mathtt{F}} \,]\!] \;|||\; \mathtt{CLASSROOM}[\![\, T_{\mathtt{C}} \parallel R_{\mathtt{C}} \,]\!] \;|||\; \mathtt{PUBLIC}[\![\, T_{\mathtt{P}} \parallel R_{\mathtt{P}} \,]\!]$$

where:

- $\mathtt{FACULTY}$, $\mathtt{CLASSROOM}$ and $\mathtt{PUBLIC}$ are names of locations;
- $|||$ is the operator of parallel composition of networks;
- $T_{\mathtt{F}}$, $T_{\mathtt{C}}$ and $T_{\mathtt{P}}$ are trees whose labelled edges are decorated by sets of roles and whose leaves contain data;
- $R_{\mathtt{F}}$, $R_{\mathtt{C}}$ and $R_{\mathtt{P}}$ are processes with roles;
- $\|$ is the separator between data trees and processes.

Each edge in a tree is assigned a set of roles that a process is required to have in order to access it. For example, let $\mathtt{st}$ and $\mathtt{prof}$ be the roles of students and professors, respectively. Let the tree $T_{\mathtt{C}}$ contain the edge $\mathtt{connection}$ with role $\mathtt{prof}$ and let the tree $T_{\mathtt{P}}$ contain the edge $\mathtt{connection}$ with roles $\mathtt{prof}$ and $\mathtt{st}$. This represents a situation in which students can access the $\mathtt{connection}$ in the $\mathtt{PUBLIC}$ location but not in the $\mathtt{CLASSROOM}$, while professors can access the $\mathtt{connection}$ in both locations.

The set of roles is partially ordered, implying that data accessible to processes with lower roles are also accessible to processes with higher roles. For example, by stating

$$\mathtt{st} \sqsubset \mathtt{prof}$$

we get that all data accessible to students are also accessible to professors. With this order we can get the previous situation if we decorate the edge $\mathtt{connection}$ in the $\mathtt{PUBLIC}$ location only with the role $\mathtt{st}$ instead with both roles $\mathtt{st}$ and $\mathtt{prof}$.

The location policy of $\mathtt{CLASSROOM}$ can prescribe that:

- the minimal role to access data is $\mathtt{st}$;
- a process with role $\mathtt{prof}$ can enable and disable the role $\mathtt{st}$ to access data.

We would represent such a policy by $(\{\mathtt{st}\}, (\{\mathtt{prof}\}, \mathtt{st}), (\{\mathtt{prof}\}, \mathtt{st}))$.

A process with role $\mathtt{prof}$ can enable and disable students to access the $\mathtt{connection}$ in the $\mathtt{CLASSROOM}$. A command for enabling is $\mathtt{enable}_{p_{\mathtt{C}}}(\mathtt{st})$, if $p_{\mathtt{C}}$ is the path from the root of the tree $T_{\mathtt{C}}$ to the edge $\mathtt{connection}$ with role $\mathtt{prof}$. After the execution of this enabling command the tree $T_{\mathtt{C}}$ will contain the edge $\mathtt{connection}$ with both roles $\mathtt{prof}$ and $\mathtt{st}$. In this way the classroom policy enables professors to allow students to do exercises using internet during the class.

2 ®X$d\pi$ calculus: role based access control for dynamic web data

2.1 Syntax of ®X$d\pi$

We design a model of dynamic web data in a RBAC scenario. Our starting point is the X$d\pi$ of [12] which we equip with roles. We model a peer-to-peer network as a set of connected locations, where each location has a policy and consists of a data tree labelled with roles and a process with roles. Processes with roles can, as in pure X$d\pi$, communicate with other processes, migrate to other location and update the local data. A novelty is that all these actions are controlled by roles. Moreover, processes can administrate roles by enabling and disabling them.

Roles. We assume a countable set of roles $\mathcal{R}$, and use r, s, t to range over elements of $\mathcal{R}$. Let $(\mathcal{R}, \sqsubseteq)$ be a lattice and let $\bot, \top \in \mathcal{R}$ be its bottom and top element, respectively. The operation of join is denoted by $\sqcup$. By α, ρ, σ we denote non-empty sets of roles and by τ, ζ sets of roles containing the $\top$ element. Each process is assigned a set of roles and each edge in trees is assigned a set of roles containing the $\top$ element. For the sake of simplicity we refer to them as to processes and trees with roles, respectively.

Different processes can have different sets of roles and the same role can be assigned to different edges and different processes.

2.1.1 Trees

T	$::= \emptyset_T$	empty rooted tree
	$\mid\ x$	tree variable
	$\mid\ T \vert T$	composition of trees, joining the roots
	$\mid\ \mathtt{a}^{\tau}[T]$	edge labelled $\mathtt{a}^{\tau}$ with subtree T
	$\mid\ \mathtt{a}^{\tau}[\Box\mathit{\Pi}]$	edge labelled $\mathtt{a}^{\tau}$ with script $\Box\mathit{\Pi}$
	$\mid\ \mathtt{a}^{\tau}[p@\lambda]$	edge labelled $\mathtt{a}^{\tau}$ with pointer $p@\lambda$
p	$::= \mathtt{a}^{\alpha} \mid x \mid p/p$	
$V,\ U$	$::= \Box\mathit{\Pi} \mid p@\lambda \mid T$	

Table 1. Syntax of trees T, paths p and data terms V, U

The data model is an unordered edge-labelled rooted tree T with leaves containing empty trees ($\emptyset_T$), scripts ($\Box\mathit{\Pi}$) and pointers ($p@\lambda$). The syntax of trees is presented in Table 1, using $\mathtt{a}^{\tau}$ to denote a tree edge with label $\mathtt{a}$ and with set of roles τ. We use τ to denote sets of roles that contain the role $\top$. In the examples we also allow standard data in the tree leaves.

A *script* $\Box\mathit{\Pi}$ is a static process embedded in a tree that can be activated by a process from the same location. The symbol $\mathit{\Pi}$ ranges over processes with roles and variables.

A *path* p identifies data in a tree. The syntax of paths is given in Table 1, using p to range over paths. In this table

- $\mathtt{a}^{\alpha}$ is a path edge with label $\mathtt{a}$ and with roles α,
- x is a variable and
- $/$ is the path composition.

A *pointer*, $p@\lambda$, refers to the set of data identified by the path p in the tree at the location λ. The symbol λ ranges over location names and variables.

Scripts, pointers and trees are referred to as *data terms* or simply *data*. Using U, V to range over data terms, their syntax is given in Table 1.

In $\circledR$X$d\pi$ roles are employed for defining the way how a path identifies data terms in a tree. A path edge $\mathtt{a}^{\alpha}$ complies with a tree edge $\mathtt{a}^{\tau}$ if for each role of α there is at least one smaller or equal role in τ. More formally, let us define

$$\tau \leqq \alpha \ \text{ iff } \ (\forall s \in \alpha)(\exists t \in \tau)\ t \sqsubseteq s$$

Then we have that a path edge $\mathtt{a}^{\alpha}$ complies with a tree edge $\mathtt{a}^{\tau}$ if $\tau \leqq \alpha$. In general:

A path $\mathtt{a}_1^{\alpha_1}/\ldots/\mathtt{a}_n^{\alpha_n}$ *complies with* a tree path $\mathtt{a}_1^{\tau_1}/\ldots/\mathtt{a}_n^{\tau_n}$ if $\tau_i \leqq \alpha_i$ for $1 \leq i \leq n$.

Then we can define:

Definition 1. *A path* identifies *a data term V in a tree T if it complies with the tree path from the root of T to V.*

Example 1. Let $\bot \sqsubset \mathtt{st} \sqsubset \mathtt{st}_i \sqsubset \mathtt{prof} \sqsubset \mathtt{dean} \sqsubset \top$, where $i \in I$ for some finite I, and let the data tree $T_{\mathtt{F}}$ of the location FACULTY be parallel composition of trees T_i, $i \in I$, where

$$T_i \equiv \mathtt{service}^{\{\mathtt{st}_i,\top\}}[\mathtt{records}^{\{\mathtt{prof},\top\}}[\ mark_i\ ] \mid \mathtt{booklet}^{\{\mathtt{st}_i,\top\}}[\ mark_i\ ]\ ].$$

The path $\mathtt{service}^{\{\mathtt{prof}\}}/\mathtt{records}^{\{\mathtt{prof}\}}$ complies with the tree path $\mathtt{service}^{\{\mathtt{st}_i,\top\}}/\mathtt{records}^{\{\mathtt{prof},\top\}}$, identifying $mark_i$ in the records. The marks in the booklet can be identified with the path $\mathtt{service}^{\{\mathtt{st}_i,\top\}}/\mathtt{booklet}^{\{\mathtt{st}_i\}}$.

2.1.2 Processes

At one location we can have a finite number of processes, with different roles, running in parallel and possibly sharing some communication channels. We define processes with roles by means of pure processes, by decorating pure processes with sets of roles. The definitions of pure processes and processes with roles are mutually recursive. We use P, Q to denote pure processes and R, S to denote processes with roles.

Pure processes. Table 2 gives the formation rules of pure processes. We use γ to range over channel names (decorated by value types) and variables. The processes that we are concerned with are essentially X$d\pi$ processes [12] to which we add commands for administration of access rights. More precisely we consider four kinds of pure processes:

- π-calculus processes [22], for modelling local communication;
- `go` command, for modelling process migration between locations, as in $D\pi$ calculus [17];
- `run`, `read` and `change` commands, for modelling interaction of processes with local data. Activating the execution of scripts embedded in local tree is done with `run`. For reading (copy) the data from a tree we use `read` and for changing (cut, paste,...) a tree we use `change`. The last two commands are in place of the `update` command of [12];
- new commands `enable` and `disable` for changing permissions to access data.

Free variables are defined as usual, taking into account that input, `read` and `change` commands are binding variables.

$P ::= 0$	the nil process
$\mid P \mid P$	parallel composition of pure processes
$\mid \bar{\gamma}\langle v\rangle$	output of value v on channel γ
$\mid \gamma(x).P$	input parametrised by a variable x
$\mid !\gamma(x).P$	replication of an input process
$\mid \mathtt{go}\ \lambda.R$	migrates to location λ, continues as the process with roles R
$\mid \mathtt{run}_p$	runs the scripts identified by path p
$\mid \mathtt{read}_p(\chi).P$	reads data identified by path p and matching with χ
$\mid \mathtt{change}_p(\chi, V).P$	changes data identified by path p and matching with χ using V
$\mid \mathtt{enable}_p(r).P$	allows role r to access data identified by path p
$\mid \mathtt{disable}_p(r).P$	forbids role r to access data identified by path p
$R ::= P^{\urcorner\rho}$	single pure process P with roles ρ assigned to it
$\mid R \mid R$	parallel composition of processes with roles
$\mid (\nu c^{Tv})R$	restriction of channel name
$v ::= c^{Tv} \mid \Box R \mid l \mid p \mid T$	
$\chi ::= \Box x^{(\sigma,\mathcal{E},\mathcal{D})} \mid y^{(\alpha)}@x^{(\sigma,\mathcal{E},\mathcal{D})} \mid x^{(\sigma,\mathcal{E},\mathcal{D},\tau,\zeta)}$	

Table 2. Syntax of pure processes P, processes with roles R, values v and patterns χ

A *value* is either a channel name decorated by a value type, a script, a location name, a path or a tree. Using v to range over values, the syntax of values is given in Table 2.

As arguments in commands for reading and changing the tree we have *patterns* (ranged over by χ), whose syntax is given in Table 2. A pattern is:

- a script whose variable is decorated with a location policy (defined below) or
- a pointer whose path variable is decorated with a set of roles and whose location variable is decorated with its policy or
- a tree variable decorated with a location policy and two sets of roles.

By $|\chi|$ we denote the data term obtained from χ by erasing all decorations.

Processes with Roles. The syntax of processes with roles is given in Table 2. A process with roles is obtained from a pure process by assigning a set of roles ρ to it or as a parallel composition of such processes. Processes with (possibly different) roles can share private communication channels (restriction operator ν).

Another novelty of ®X$d\pi$ is the flexibility of data access which is due to introduction of roles. Namely, each process with at least one role bigger than or equal to one role in α has the permission to access the path edge $\mathtt{a}^{\alpha}$. For this reason we introduce the relation $\leq$ between sets of roles defined by:

$$\alpha \leq \rho \text{ if } (\exists s \in \alpha)(\exists r \in \rho)\ s \sqsubseteq r.$$

Then a process with roles ρ can access the edge $\mathtt{a}^{\alpha}$ if $\alpha \leq \rho$. More generally:

A process with roles ρ can *access* a path $\mathtt{a}_1^{\alpha_1}/\ldots/\mathtt{a}_n^{\alpha_n}$ if $\alpha_i \leq \rho$ for $1 \leq i \leq n$.

Similarly a process with roles ρ can *access* a tree path $\mathtt{a}_1^{\tau_1}/\ldots/\mathtt{a}_n^{\tau_n}$ if $\tau_i \leq \rho$ for $1 \leq i \leq n$. Lastly we define:

Definition 2. *A process with roles can* access *a data term V in a tree T if it can access the tree path from the root of T to V.*

Note that all processes with role $\top$ can access all data, since the sets of roles associated to tree edges are never empty.

Example 2. A process with role $\{\mathtt{prof}\}$ can access the tree paths

$$\mathtt{service}^{\{\mathtt{st}_i,\top\}}/\mathtt{records}^{\{\mathtt{prof},\top\}} \text{ and } \mathtt{service}^{\{\mathtt{st}_i,\top\}}/\mathtt{booklet}^{\{\mathtt{st}_i,\top\}}$$

and then it can access marks both in the records and in the student's booklet, while a process with role $\{\mathtt{st}_i\}$ can access only marks in the booklet since it can access the tree path $\mathtt{service}^{\{\mathtt{st}_i,\top\}}/\ \mathtt{booklet}^{\{\mathtt{st}_i,\top\}}$. If we suppose that for $i \neq j$ the roles $\mathtt{st}_i$ and $\mathtt{st}_j$ are unrelated, a process with the role $\{\mathtt{st}_j\}$ cannot access the tree path $\mathtt{service}^{\{\mathtt{st}_i,\top\}}/\ \mathtt{booklet}^{\{\mathtt{st}_i,\top\}}$.

2.1.3 Networks

Networks are the main syntactic features of the untyped ®X$d\pi$ calculus. A network $\mathbf{N}$ is a parallel composition (|||) of *locations* l consisting of a data tree T and a process R, where roles are associated both to data trees and to processes. Processes at different locations can share communication channels c^{Tv}. The syntax of networks is given in Table 3. We use l, m to range over location names, c to range over channel names and Tv to denote a value type as defined in Table 9.

$$\mathbf{N} ::= \mathbf{0} \ \mid\ \mathbf{N} \parallel\!| \mathbf{N} \ \mid\ l[\![\, T \parallel R \,]\!] \ \mid\ (\nu c^{Tv})\mathbf{N}$$

Table 3. Syntax of networks

2.1.4 Location Policies

$$(\rho, r) \in^{+} \mathcal{E} \Longleftrightarrow \exists (\rho', r') \in \mathcal{E} \text{ such that } \rho' \leq \rho \text{ and } r' \sqsubseteq r$$
$$(\rho, r) \in^{-} \mathcal{D} \Longleftrightarrow \exists (\rho', r') \in \mathcal{D} \text{ such that } \rho' \leq \rho \text{ and } r \sqsubseteq r'$$

Table 4. Definitions of $\in^{+}$ and $\in^{-}$.

A location policy is the triple $(\sigma, \mathcal{E}, \mathcal{D})$, where σ is a set of roles, whereas $\mathcal{E}$ and $\mathcal{D}$ are subsets of $\{(\rho, r) : \rho \subseteq \mathcal{R},\ r \in \mathcal{R}\}$. The data accessibility policy is given by the set σ, the set of minimal roles a process is required to have to access the data at that location. The administration policy is given by the other sets which prescribe changes of data access rights as follows ($\in^{+}$ and $\in^{-}$ are defined in Table 4):

- if $(\rho, r) \in^{+} \mathcal{E}$, a process with roles ρ can give the permission to (enable) the role r to access the data;
- if $(\rho, r) \in^{-} \mathcal{D}$, a process with roles ρ can take the permission from (disable) the role r to access the data.

We introduce $\in^{+}$ and $\in^{-}$ in order to:

- allow processes with higher roles to modify access rights which processes with lower roles can already modify (condition $\rho' \leq \rho$);
- allow to enable higher roles when lower roles can be enabled (condition $r' \sqsubseteq r$);
- allow to disable lower roles when higher roles can be disabled (condition $r \sqsubseteq r'$).

Definition 3. *A location policy $(\sigma, \mathcal{E}, \mathcal{D})$ is well-formed if $(\rho, r) \in \mathcal{E} \cup \mathcal{D}$ implies $r \neq \top$ and $\sigma \leqq \rho \cup \{r\}$.*

The conditions for a location to be well formed require that:

- the role $\top$ is neither enabled nor disabled: this agrees with the assumption that all sets of roles decorating tree edges contain $\top$;
- the roles involved in changing access rights can be roles of some edges in the local tree, i.e. they are not less than some roles in σ.

Example 3. The policy $(\sigma_{\mathtt{F}}, \mathcal{E}_{\mathtt{F}}, \mathcal{D}_{\mathtt{F}})$ of the FACULTY, with $\sigma_{\mathtt{F}} = \{\mathtt{st}_i : i \in I\}$, $\mathcal{E}_{\mathtt{F}} = \{(\{\mathtt{prof}\}, \mathtt{as})\}$, and $\mathcal{D}_{\mathtt{F}} = \{(\{\mathtt{dean}\}, \mathtt{st}_i) : i \in I\}$, is well formed. According to this policy:

- a student with role st is not allowed to access any data at the location;
- a professor can give the permission to a teaching assistant to edit the records;
- the dean can forbid the student i to access some data;
- it holds that $(\{\mathtt{dean}\}, \mathtt{prof}) \in^{+} \mathcal{E}$.

In what follows we will consider only well-formed location policies. We assume a fixed function $\mathcal{T}$ which associates locations with their policies.

$$\mathcal{T}(l) = (\sigma, \mathcal{E}, \mathcal{D}) \quad \text{if } (\sigma, \mathcal{E}, \mathcal{D}) \text{ is the policy of } l.$$

2.2 Reduction Rules

The reduction relation is the least relation on networks which is closed with respect to the structural equivalence, reduction rules given in Table 8 and reduction contexts given by:

$$C ::= - \;\mathbf{|}\; C \;|||\; \mathbf{N} \;\mathbf{|}\; (\nu c^{Tv})C$$

(trees)	$V \equiv V' \Rightarrow \mathtt{a}^{\rho}[V] \equiv \mathtt{a}^{\rho}[V']$
(scripts)	$R \equiv R' \Rightarrow \Box R \equiv \Box R'$
(processes with roles)	$(\nu c^{Tv})0^{\neg\rho} \equiv 0^{\neg\rho}$
	$(P \mid Q)^{\neg\rho} \equiv P^{\neg\rho} \mid Q^{\neg\rho}$
	$v \equiv v' \Rightarrow \bar{c}^{Tv}\langle v\rangle^{\neg\rho} \equiv \bar{c}^{Tv}\langle v'\rangle^{\neg\rho}$
	$(\nu c^{Tv})(\nu d^{Tv'})R \equiv (\nu d^{Tv'})(\nu c^{Tv})R$
	$c^{Tv} \notin \mathit{fn}(R) \Rightarrow R \mid (\nu c^{Tv})S \equiv (\nu c^{Tv})(R \mid S)$
	$V \equiv V' \Rightarrow \mathtt{change}_p(\chi, V).P^{\neg\rho} \equiv \mathtt{change}_p(\chi, V').P^{\neg\rho}$
(networks)	$(\nu c^{Tv})\mathbf{0} \equiv \mathbf{0}$
	$(\nu c^{Tv})(\nu d^{Tv'})\mathbf{N} \equiv (\nu d^{Tv'})(\nu c^{Tv})\mathbf{N}$
	$c^{Tv} \notin \mathit{fn}(\mathbf{N}) \Rightarrow \mathbf{N} \;|||\; (\nu c^{Tv})\mathbf{N}' \equiv (\nu c^{Tv})(\mathbf{N} \;|||\; \mathbf{N}')$
	$T \equiv T' \wedge R \equiv R' \Rightarrow l[\![\; T \parallel R \;]\!] \equiv l[\![\; T' \parallel R' \;]\!]$
	$c^{Tv} \notin \mathit{fn}(T) \Rightarrow l[\![\; T \parallel (\nu c^{Tv})R \;]\!] \equiv (\nu c^{Tv})l[\![\; T \parallel R \;]\!]$

Table 5. Structural equivalence

The structural equivalence is the least equivalence relation on networks that satisfies alpha-conversion, the commutative monoid properties for $(\emptyset_T, \mid)$ on trees, for $(0^{\neg\rho}, \mid)$ on processes with roles and for $(\mathbf{0}, \;|||\;)$ on networks, and the axioms of Table 5. As usual *fn* is the set of free channel names occurring in a process with roles or in a tree or in a network.

The reduction relation describes four forms of interactions:

- local interaction between processes with roles (rules (`com`) and (`com!`));
- interaction between locations, describing migration of processes with roles (rules (`go`) and (`stay`));
- local interaction between processes with roles and trees, describing execution of scripted processes and data manipulation (rules (`run`), (`read`) and (`change`));
- local interaction between processes with roles and trees, describing role administration (rules (`enable`) and (`disable`)).

The communication rules (`com`) and (`com!`) are from the π-calculus [22]. Processes can communicate only if they are in the same location. There are two rules for migration. Rule (`go`) describes migration to a distinct location. The other rule, (`stay`), describes staying at the current location.

$$\begin{array}{lcl}
\mathsf{map}(p,\emptyset_T,\mathsf{f},\mathsf{op}) & = & \begin{cases}\emptyset_T & \text{if } \mathsf{op} = \mid,\\ \emptyset & \text{if } \mathsf{op} = \cup\end{cases}\\
\mathsf{map}(p,T_1 \mid T_2,\mathsf{f},\mathsf{op}) & = & (\mathsf{map}(p,T_1,\mathsf{f},\mathsf{op}))\ \mathsf{op}\ (\mathsf{map}(p,T_2,\mathsf{f},\mathsf{op}))\\
\mathsf{map}(p,\mathsf{b}^{\tau}[V],\mathsf{f},\mid) & = & \begin{cases}\mathsf{b}^{\tau}[\mathsf{f}(V)] & \text{if } p = \mathsf{b}^{\alpha}\text{, and } \tau \leqq \alpha\ ,\\ \mathsf{b}^{\tau}[\mathsf{map}(q,V,\mathsf{f},\mid)] & \text{if } p = \mathsf{b}^{\alpha}/q\text{, and } \tau \leqq \alpha\ ,\\ \mathsf{b}^{\tau}[V] & \text{otherwise}\end{cases}\\
\mathsf{map}(p,\mathsf{b}^{\tau}[V],\mathsf{f},\cup) & = & \begin{cases}\mathsf{f}(V) & \text{if } p = \mathsf{b}^{\alpha}\text{, and } \tau \leqq \alpha\ ,\\ \mathsf{map}(q,V,\mathsf{f},\cup) & \text{if } p = \mathsf{b}^{\alpha}/q\text{, and } \tau \leqq \alpha\ ,\\ \emptyset & \text{otherwise}\end{cases}
\end{array}$$

Table 6. Definition of the function `map`$(p, T, \mathsf{f}, \mathsf{op})$

The reduction rules (`run`), (`read`), (`change`), (`enable`) and (`disable`) use the functions defined in Table 7 by means of the function map given in Table 6. The function map takes as arguments a path p, a tree T, a function f and an operator op $\in \{\mid, \cup\}$. It is defined by cases:

- if the tree is empty, then the result is the neutral element for op;
- if the tree is a parallel composition of sub-trees, then the result is the application of op to the values returned by applying map to the sub-trees;
- otherwise, map checks if the top path edge complies with the top tree edge and:
  - in case of compliance
    - if the path is only one edge, then map applies the function f to the so identified data term, and it returns a tree if op = $\mid$ and the function value if op = $\cup$;
    - if the path is longer, then map is applied recursively to the so identified data term;
  - in case of noncompliance the result is the current tree if op = $\mid$ and the empty set if op = $\cup$.

The functions `rn`, `sub`, `ch`, `en` and `di` are obtained from the function map by specialising f and op. For the first two functions op = $\cup$ and the returned values are sets, for the remaining functions op = $\mid$ and the returned values are trees.

For `rn` the function f checks if the identified data is a script. If it is the case, it returns the set which contains the found process, and the empty set otherwise.

For the functions `sub` and `ch` we need the typed matching function `match` which, in order to check if a data term agrees with a pattern, requires not only the data term to be of the form of the three pattern shapes, respectively, a script, a pointer, or a tree, but

$$
\begin{array}{ll}
\mathtt{rn}(p,T) & = \mathsf{map}(p,T,\lambda v.\begin{cases}\{R\} & \text{if } v=\Box R\\ \emptyset & \text{otherwise}\end{cases},\cup)\\
\mathtt{sub}(p,\chi,T) & = \mathsf{map}(p,T,\lambda v.\begin{cases}\{\!\{v/|\chi|\}\!\} & \text{if } \mathtt{match}(\chi,v),\\ \emptyset & \text{otherwise}\end{cases},\cup)\\
\mathtt{ch}(p,\chi,V,T) & = \mathsf{map}(p,T,\lambda v.\begin{cases}V\{\!\{v/|\chi|\}\!\} & \text{if } \mathtt{match}(\chi,v),\\ v & \text{otherwise}\end{cases},|)\\
\mathtt{en}(p,r,T) & = \mathsf{map}(p,T,\lambda v.(v)^{+r},|)\\
\mathtt{di}(p,r,T) & = \mathsf{map}(p,T,\lambda v.(v)^{-r},|)
\end{array}
$$

where

$$
\begin{array}{ll}
(\emptyset_T)^{+r} = \emptyset_T & (\emptyset_T)^{-r} = \emptyset_T\\
(T_1|T_2)^{+r} = (T_1)^{+r}|(T_2)^{+r} & (T_1|T_2)^{-r} = (T_1)^{-r}|(T_2)^{-r}\\
(\mathsf{a}^{\tau}[U])^{+r} = \mathsf{a}^{\tau\cup\{r\}}[U] & (\mathsf{a}^{\tau}[U])^{-r} = \mathsf{a}^{\tau\setminus\setminus\{r\}}[(U)^{-r}]\\
(p@l)^{+r} = p@l & (p@l)^{-r} = p@l\\
(\Box R)^{+r} = \Box R & (\Box R)^{-r} = \Box R
\end{array}
$$

Table 7. Definitions of functions `rn`, `sub`, `ch`, `en` and `di`

it requires also the data term to satisfy the type information given by the pattern. This means that:

(1) if the pattern is $\Box x^{(\sigma,\mathcal{E},\mathcal{D})}$, then the data term must be a script which can run at locations with policy $(\sigma,\mathcal{E},\mathcal{D})$,

(2) if the pattern is $y^{(\alpha)}@x^{(\sigma,\mathcal{E},\mathcal{D})}$, then the data term must be a pointer in which
  (i) the last edge of the path has the set α of roles and
  (ii) the policy of the location is $(\sigma,\mathcal{E},\mathcal{D})$,

(3) if the pattern is $x^{(\sigma,\mathcal{E},\mathcal{D},\tau,\zeta)}$, then the data term must be a tree
  (i) which can stay at locations with policy $(\sigma,\mathcal{E},\mathcal{D})$ and
  (ii) such that the union of the sets of roles associated to the top edges is τ and
  (iii) such that a process with roles $\rho \geq \zeta$ can access the whole tree.

These conditions are enforced by using the type assignment system of Section 3. More precisely the definition of the match function is:

(1) $\mathtt{match}(\Box x^{(\sigma,\mathcal{E},\mathcal{D})},\Box R)$ if $\vdash R : ProcRole(\sigma,\mathcal{E},\mathcal{D})$;
(2) $\mathtt{match}(y^{(\alpha)}@x^{(\sigma,\mathcal{E},\mathcal{D})},p@l)$ if $\vdash p : Path(\alpha)$ and $\vdash l : Loc(\sigma,\mathcal{E},\mathcal{D})$;
(3) $\mathtt{match}(x^{(\sigma,\mathcal{E},\mathcal{D},\tau,\zeta)},T)$ if $\vdash T : Tree(\sigma,\mathcal{E},\mathcal{D},\tau,\zeta)$.

In sub the function f returns a substitution (denoted by $\{\!\{v/|\chi|\}\!\}$) if the identified data matches the given pattern. If the data and the pattern are pointers, then the substitution replaces the location variable with the location and the path variable with the path. In the other case the substitution simply replaces the pattern variable with the data. More precisely we define:

- $\{\!\{\Box R/\Box x\}\!\} = \{R/x\}$;

$(\texttt{com})$	$l\llbracket\, T \parallel \bar{c}^{Tv}\langle v\rangle^{\neg\rho'} \mid c^{Tv}(z).P^{\neg\rho} \mid R\,\rrbracket \rightarrow l\llbracket\, T \parallel P\{v/z\}^{\neg\rho} \mid R\,\rrbracket$
$(\texttt{com!})$	$l\llbracket\, T \parallel \bar{c}^{Tv}\langle v\rangle^{\neg\rho'} \mid !c^{Tv}(z).P^{\neg\rho} \mid R\,\rrbracket \rightarrow l\llbracket\, T \parallel !c^{Tv}(z).P^{\neg\rho} \mid P\{v/z\}^{\neg\rho} \mid R\,\rrbracket$
$(\texttt{go})$	$l\llbracket\, T_1 \parallel \texttt{go}\ m.R \mid R_1\,\rrbracket \;\Vert\!\mid\; m\llbracket\, T_2 \parallel R_2\,\rrbracket \rightarrow l\llbracket\, T_1 \parallel R_1\,\rrbracket \;\Vert\!\mid\; m\llbracket\, T_2 \parallel R \mid R_2\,\rrbracket$
$(\texttt{stay})$	$l\llbracket\, T \parallel \texttt{go}\ l.R \mid R'] \rightarrow l\llbracket\, T \parallel R \mid R'\,\rrbracket$
$(\texttt{run})$	$\dfrac{\texttt{rn}(p,T) = \{R_1,\ldots,R_n\}}{l\llbracket\, T \parallel \texttt{run}_p{}^{\neg\rho} \mid R\,\rrbracket \rightarrow l\llbracket\, T \parallel R_1 \mid \ldots \mid R_n \mid R\,\rrbracket}$
$(\texttt{read})$	$\dfrac{\texttt{sub}(p,\chi,T) = \{\texttt{s}_1,\ldots,\texttt{s}_n\}}{l\llbracket\, T \parallel \texttt{read}_p(\chi).P^{\neg\rho} \mid R\,\rrbracket \rightarrow l\llbracket\, T \parallel P\texttt{s}_1{}^{\neg\rho} \mid \ldots \mid P\texttt{s}_n{}^{\neg\rho} \mid R\,\rrbracket}$
$(\texttt{change})$	$\dfrac{\texttt{ch}(p,\chi,V,T) = T'}{l\llbracket\, T \parallel \texttt{change}_p(\chi,V).P^{\neg\rho} \mid R\,\rrbracket \rightarrow l\llbracket\, T' \parallel P^{\neg\rho} \mid R\,\rrbracket}$
$(\texttt{enable})$	$\dfrac{\texttt{en}(p,r,T) = T'}{l\llbracket\, T \parallel \texttt{enable}_p(r).P^{\neg\rho} \mid R\,\rrbracket \rightarrow l\llbracket\, T' \parallel P^{\neg\rho} \mid R\,\rrbracket}$
$(\texttt{disable})$	$\dfrac{\texttt{di}(p,r,T) = T'}{l\llbracket\, T \parallel \texttt{disable}_p(r).P^{\neg\rho} \mid R\,\rrbracket \rightarrow l\llbracket\, T' \parallel P^{\neg\rho} \mid R\,\rrbracket}$

Table 8. Reduction rules

- $\{\!\{p@l/y@x\}\!\} = \{p/y, l/x\}$;
- $\{\!\{T/x\}\!\} = \{T/x\}$.

In case of mismatch the result is the empty set.

The function $\texttt{ch}$ has an extra argument V, the data term in which the given pattern is replaced with the identified data in case of matching between the identified data and the given pattern. More precisely, the function f in $\texttt{ch}$ returns the data V in which the given pattern is replaced with the identified data in case of matching, or it returns the identified data otherwise.

The $\texttt{en}$ and $\texttt{di}$ functions use the auxiliary mappings $(\ )^{+r}$ and $(\ )^{-r}$, respectively. The mapping $(\ )^{+r}$ adds the role r to the first tree edges starting from the roots of the subtrees identified by path p, if any, or does nothing otherwise. The mapping $(\ )^{-r}$ removes all the roles less than or equal to the role r from all the edges in the subtrees identified by path p, if any, or does nothing otherwise. In fact we define:

$$\tau \setminus\!\setminus \{r\} = \{s \in \tau \mid s \not\sqsubseteq r\}.$$

The typing rules assure that $r \neq \top$, so we never get a set of roles without $\top$ as a result.

Example 4. If we are given the tree

$$T \equiv \texttt{records}^{\{\texttt{prof},\top\}}[\ mark_i] \mid \texttt{booklet}^{\{\texttt{st}_i,\top\}}[\ mark_i],$$

then

$$(T)^{+\mathtt{as}} \equiv \mathtt{records}^{\{\mathtt{as},\mathtt{prof},\top\}}[\ mark_i] \mid \mathtt{booklet}^{\{\mathtt{st}_i,\mathtt{as},\top\}}[\ mark_i] \text{ and}$$
$$(T)^{-\mathtt{st}_i} \equiv \mathtt{records}^{\{\mathtt{prof},\top\}}[\ mark_i] \mid \mathtt{booklet}^{\{\top\}}[\ mark_i].$$

Example 5. The value of $(T)^{+r}$ where

$$T \equiv \mathtt{a}^{\{s_1,s_2,\top\}}[\Box R] \mid \mathtt{a}^{\{s_1,s_3,\top\}}[\mathtt{b}^{\{s_3,\top\}}[p@\lambda]]$$

will be the tree with the role r added to the top edges:

$$T' \equiv \mathtt{a}^{\{s_1,s_2,r,\top\}}[\Box R] \mid \mathtt{a}^{\{s_1,s_3,r,\top\}}[\mathtt{b}^{\{s_3,\top\}}[p@\lambda]].$$

If we apply the mapping $(\ )^{-s_2}$ to the obtained tree T' and if $s_3 \sqsubseteq s_2$, while the other roles are unrelated, we will get

$$(T')^{-s_2} \equiv \mathtt{a}^{\{s_1,r,\top\}}[\Box R] \mid \mathtt{a}^{\{s_1,r,\top\}}[\mathtt{b}^{\{\top\}}[p@\lambda]].$$

The process $\mathtt{run}_p{}^{\neg\rho}$ in rule (run) activates in parallel all the scripts identified by the path p in the local tree using the function $\mathtt{rn}$.

In rule ($\mathtt{read}$) the process $\mathtt{read}_p(\chi).P^{\neg\rho}$ finds all the data terms identified by the path p in the local tree and matching χ to obtain a set of substitutions using the function $\mathtt{sub}$. For all the substitutions $\mathtt{s}$ in this set it activates $P\mathtt{s}^{\neg\rho}$.

The change process $\mathtt{change}_p(\chi, V).P^{\neg\rho}$ modifies the local tree using the function $\mathtt{ch}$. It finds the set of data terms identified by the path p and matching with χ. For each data term U in this set the change process replaces it by $V\{\!\{U/|\chi|\}\!\}$.

Rules ($\mathtt{enable}$) and ($\mathtt{disable}$) add and remove the role r starting from the roots of the subtrees identified by the path p. The difference is that rule ($\mathtt{enable}$) only adds the role to the edges starting from these roots, while rule ($\mathtt{disable}$) removes the role from all the edges in the subtrees.

The type system of next section will assure that all paths which occur in a process with roles are accessible to that process.

Example 6. The net

$$\mathbf{N} = m\llbracket\, \mathtt{a}^{\{t,\top\}}[\mathtt{b}^{\{s\}}@l] \ \|\ P^{\neg\rho'}\,\rrbracket \ \ |||\ \ l\llbracket\, \mathtt{b}^{\{s,\top\}}[\mathtt{c}^{\{\top\}}[\emptyset_T]] \ \|\ 0^{\neg\rho}\,\rrbracket,$$

where
$P = \mathtt{read}_{\mathrm{a}\{t\}}(y^{(\{s\})}@x^{(\sigma,\mathcal{E},\mathcal{D})}).\mathtt{change}_{\mathrm{a}\{t\}}(y_1^{(\{s\})}@x_1^{(\sigma,\mathcal{E},\mathcal{D})}, \Box R).\mathtt{go}\ x.\mathtt{enable}_y(r)^{\neg\rho}$
and $\mathcal{T}(l) = (\sigma, \mathcal{E}, \mathcal{D})$, reduces as follows:

$$\begin{aligned}
\mathbf{N} &\rightarrow m\llbracket\, \mathtt{a}^{\{t,\top\}}[\mathtt{b}^{\{s\}}@l] \ \|\ P'^{\neg\rho'}\,\rrbracket \ \ |||\ \ l\llbracket\, \mathtt{b}^{\{s,\top\}}[\mathtt{c}^{\{\top\}}[\emptyset_T]] \ \|\ 0^{\neg\rho}\,\rrbracket \\
&\rightarrow m\llbracket\, \mathtt{a}^{\{t,\top\}}[\Box R] \ \|\ \mathtt{go}\ l.\mathtt{enable}_{\mathrm{b}\{s\}}(r)^{\neg\rho\neg\rho'}\,\rrbracket \ \ |||\ \ l\llbracket\, \mathtt{b}^{\{s,\top\}}[\mathtt{c}^{\{\top\}}[\emptyset_T]] \ \|\ 0^{\neg\rho}\,\rrbracket \\
&\rightarrow m\llbracket\, \mathtt{a}^{\{t,\top\}}[\Box R] \ \|\ 0^{\neg\rho}\,\rrbracket \ \ |||\ \ l\llbracket\, \mathtt{b}^{\{s,\top\}}[\mathtt{c}^{\{\top\}}[\emptyset_T]] \ \|\ \mathtt{enable}_{\mathrm{b}\{s\}}(r)^{\neg\rho}\,\rrbracket \\
&\rightarrow m\llbracket\, \mathtt{a}^{\{t,\top\}}[\Box R] \ \|\ 0^{\neg\rho}\,\rrbracket \ \ |||\ \ l\llbracket\, \mathtt{b}^{\{s,\top\}}[\mathtt{c}^{\{r,\top\}}[\emptyset_T]] \ \|\ 0^{\neg\rho}\,\rrbracket
\end{aligned}$$

where $P' = \mathtt{change}_{\mathrm{a}\{t\}}(y_1^{(\{s\})}@x_1^{(\sigma,\mathcal{E},\mathcal{D})}, \Box R).\mathtt{go}\ l.\mathtt{enable}_{\mathrm{b}\{s\}}(r)^{\neg\rho}$. This example shows how by means of $\mathtt{read}$ and $\mathtt{change}$ commands we can use the data in trees for modifying both the processes and the trees.

3 Type Assignment for $\circledR$X$d\pi$

In this section we introduce a type system for $\circledR$X$d\pi$ in order to control the communication of values, the migration of processes and the access of processes to data. In Table 9 we introduce the syntax of types corresponding to the syntactic categories from Section 2.1.

A tree type $Tree(\sigma, \mathcal{E}, \mathcal{D}, \tau, \zeta)$ is well formed if $\sigma \leqq \tau$, meaning that each role appearing at initial branches of the tree has to be bigger than or equal to a role from the set σ of minimal roles which is given by location policy. This condition implies that each edge in a well-typed tree has a set of roles which respects the location policy.

A process type $Proc(\sigma, \mathcal{E}, \mathcal{D}, \rho)$ is well formed if $\sigma \leq \rho$. This requirement guaranties that the process has at least one role bigger than or equal to one role belonging to the set of minimal roles prescribed by the location policy.

In the following we will consider only well-formed types.

$Loc(\sigma, \mathcal{E}, \mathcal{D})$	type of locations with policy $(\sigma, \mathcal{E}, \mathcal{D})$
$Script(\sigma, \mathcal{E}, \mathcal{D})$	type of scripts which can be activated at locations with policy $(\sigma, \mathcal{E}, \mathcal{D})$
$Path(\alpha)$	type of paths having the last edge with set of roles α
$Pointer(\alpha)$	type of pointers whose path is typed by $Path(\alpha)$
$Tree(\sigma, \mathcal{E}, \mathcal{D}, \tau, \zeta)$	type of trees, which can stay at locations with policy $(\sigma, \mathcal{E}, \mathcal{D})$, with initial branches asking τ and which can be completely accessed by processes with at least one role of ζ
$Proc(\sigma, \mathcal{E}, \mathcal{D}, \rho)$	type of pure processes, which can stay at locations with policy $(\sigma, \mathcal{E}, \mathcal{D})$ and which can be assigned roles ρ
$ProcRole(\sigma, \mathcal{E}, \mathcal{D})$	type of processes with roles which can stay at locations with policy $(\sigma, \mathcal{E}, \mathcal{D})$
$Ch(Tv)$	type of channels communicating values of type Tv
Net	type of networks

Tv ranges over *value types* defined by:

$Tv ::= Ch(Tv) \mid Loc(\sigma, \mathcal{E}, \mathcal{D}) \mid Script(\sigma, \mathcal{E}, \mathcal{D}) \mid Path(\alpha) \mid Tree(\sigma, \mathcal{E}, \mathcal{D}, \tau, \zeta)$

Table 9. Syntax of types and definition of value type

An *environment* Γ associates variables with value types and with types of processes with roles, i.e. we define:

$$\Gamma ::= \emptyset \mid \Gamma, x : Tv \mid \Gamma, x : ProcRole(\sigma, \mathcal{E}, \mathcal{D})$$

We use the environments by the standard axioms:

$$\frac{}{\Gamma, x : Tv \vdash x : Tv}\ (\mathrm{Ax}_1) \qquad \frac{}{\Gamma, x : ProcRole(\sigma, \mathcal{E}, \mathcal{D}) \vdash x : ProcRole(\sigma, \mathcal{E}, \mathcal{D})}\ (\mathrm{Ax}_2)$$

The typing rule for locations uses the function $\mathcal{T}$:

$$\frac{\mathcal{T}(l) = (\sigma, \mathcal{E}, \mathcal{D})}{\varGamma \vdash l : \mathit{Loc}(\sigma, \mathcal{E}, \mathcal{D})} \ \text{(Loc)}$$

3.1 Typing rules for trees

$$\frac{\varGamma \vdash \varPi : \mathit{ProcRole}(\sigma, \mathcal{E}, \mathcal{D})}{\varGamma \vdash \Box\varPi : \mathit{Script}(\sigma, \mathcal{E}, \mathcal{D})} \ \text{(Script)}$$

$$\frac{}{\varGamma \vdash \mathtt{a}^{\alpha} : \mathit{Path}(\alpha)} \ \text{(PathEdge)} \qquad \frac{\varGamma \vdash p_1 : \mathit{Path}(\beta) \quad \varGamma \vdash p_2 : \mathit{Path}(\alpha)}{\varGamma \vdash p_1/p_2 : \mathit{Path}(\alpha)} \ \text{(PathComp)}$$

$$\frac{\varGamma \vdash p : \mathit{Path}(\alpha) \quad \varGamma \vdash \lambda : \mathit{Loc}(\sigma, \mathcal{E}, \mathcal{D})}{\varGamma \vdash p@\lambda : \mathit{Pointer}(\alpha)} \ \text{(Pointer)}$$

$$\frac{}{\varGamma \vdash \emptyset_T : \mathit{Tree}(\sigma, \mathcal{E}, \mathcal{D}, \{\top\}, \{\bot, \top\})} \ \text{(EmptyTree)}$$

$$\frac{\varGamma \vdash p@\lambda' : \mathit{Pointer}(\alpha)}{\varGamma \vdash \mathtt{a}^{\tau}[p@\lambda'] : \mathit{Tree}(\sigma, \mathcal{E}, \mathcal{D}, \tau, \tau)} \ \text{(TreePointer)}$$

$$\frac{\varGamma \vdash \Box\varPi : \mathit{Script}(\sigma, \mathcal{E}, \mathcal{D})}{\varGamma \vdash \mathtt{a}^{\tau}[\Box\varPi] : \mathit{Tree}(\sigma, \mathcal{E}, \mathcal{D}, \tau, \tau)} \ \text{(TreeScript)}$$

$$\frac{\varGamma \vdash T : \mathit{Tree}(\sigma, \mathcal{E}, \mathcal{D}, \tau', \zeta) \quad \tau \leqq \tau'}{\varGamma \vdash \mathtt{a}^{\tau}[T] : \mathit{Tree}(\sigma, \mathcal{E}, \mathcal{D}, \tau, \tau \natural \zeta)} \ \text{(TreeEdge)}$$

$$\frac{\varGamma \vdash T_1 : \mathit{Tree}(\sigma, \mathcal{E}, \mathcal{D}, \tau_1, \zeta_1) \quad \varGamma \vdash T_2 : \mathit{Tree}(\sigma, \mathcal{E}, \mathcal{D}, \tau_2, \zeta_2)}{\varGamma \vdash T_1 \mid T_2 : \mathit{Tree}(\sigma, \mathcal{E}, \mathcal{D}, \tau_1 \cup \tau_2, \zeta_1 \sqcup \zeta_2)} \ \text{(TreeComp)}$$

Table 10. Typing rules for scripts, paths, pointers and trees

Table 10 gives the typing rules for scripts, paths, pointers and trees.

Typing rule for scripts. If a process with roles $\varPi$ respects the location policy $(\sigma, \mathcal{E}, \mathcal{D})$, then the script $\Box\varPi$ respects the location policy $(\sigma, \mathcal{E}, \mathcal{D})$, too.

Typing rules for paths. A path p is of type $\mathit{Path}(\alpha)$ if the last edge in the path has the set of roles α.

Example 7. The type of the path $\mathtt{service}^{\{\mathtt{st}_i\}}/\mathtt{records}^{\{\mathtt{prof}\}}$ is $\mathit{Path}(\{\mathtt{prof}\})$.

Typing rule for pointers. A pointer is of type $\mathit{Pointer}(\alpha)$ if the path has type $\mathit{Path}(\alpha)$ and the location is typable.

Typing rules for trees. By rule (EmptyTree), an empty tree can stay in any location, since the relation $\sigma \leqq \{\top\}$ holds for any σ. Due to rule (TreeScript), if a script $\Box\Pi$ respects a location policy, it can be in a leaf of the trees in locations with that policy.

In a well-typed tree, by rule (TreeEdge), an edge with roles τ can connect a parent node with a child node of type $Tree(\sigma, \mathcal{E}, \mathcal{D}, \tau', \zeta)$ only if $\tau \leqq \tau'$. This assures that a process which can access the tree T can also access the edge. Therefore, since in a well-typed tree the tree path $\mathtt{a}_1^{\tau_1}/\ldots/\mathtt{a}_n^{\tau_n}$ has the property $\tau_1 \leqq \ldots \leqq \tau_n$, we can reformulate accessibility of tree paths by processes as follows:

A process with roles ρ can *access* a tree path $\mathtt{a}_1^{\tau_1}/\ldots/\mathtt{a}_n^{\tau_n}$ if $\tau_n \leq \rho$.

In the type of the conclusion, we define

$$\tau\natural\zeta = \begin{cases} \tau & \text{if } \zeta = \{\bot, \top\}, \\ \zeta & \text{otherwise.} \end{cases}$$

in order to get that the last set of roles guarantees that all edges are accessible. So we get $\tau\natural\zeta = \tau$ if T is the empty tree or $\tau' = \zeta = \{\bot, \top\}$ (note that in this last case $\tau \leqq \tau'$ implies $\bot \in \tau$) and $\tau\natural\zeta = \zeta$ otherwise.

If a process with roles $\rho \geq \zeta_i$ can access the tree T_i for $i = 1, 2$, then a process with roles $\rho \geq \zeta_1 \sqcup \zeta_2 = \{r \sqcup s \mid r \in \zeta_1 \;\&\; s \in \zeta_2\}$ can access the tree $T_1 \mid T_2$. For this reason we use $\zeta_1 \sqcup \zeta_2$ in the conclusion of rule (TreeComp). Note that $T|\emptyset_T$ has the same type as T, because $\tau \cup \{\top\} = \tau$ and $\zeta \sqcup \{\bot, \top\} = \zeta$ for any τ and ζ (recall that τ, ζ range over sets of roles containing $\top$).

Example 8. We assume that trees containing standard data in leaves are typed as expected.
If

$$T_1 \equiv \mathtt{records}^{\{\mathtt{prof},\top\}}[\, mark_i] \text{ and } T_2 \equiv \mathtt{booklet}^{\{\mathtt{st_i},\top\}}[\, mark_i].$$

we can derive:

$$\vdash T_1 : Tree(\sigma_{\mathtt{F}}, \mathcal{E}_{\mathtt{F}}, \mathcal{D}_{\mathtt{F}}, \{\mathtt{prof}, \top\}, \{\mathtt{prof}, \top\}) \text{ and}$$
$$\vdash T_2 : Tree(\sigma_{\mathtt{F}}, \mathcal{E}_{\mathtt{F}}, \mathcal{D}_{\mathtt{F}}, \{\mathtt{st}_i, \top\}, \{\mathtt{st}_i, \top\}).$$

By (TreeComp), since $\{\mathtt{prof}, \top\} \sqcup \{\mathtt{st}_i, \top\} = \{\mathtt{prof}, \top\}$, we derive that

$$\vdash T_1 \mid T_2 : Tree(\sigma_{\mathtt{F}}, \mathcal{E}_{\mathtt{F}}, \mathcal{D}_{\mathtt{F}}, \{\mathtt{prof}, \mathtt{st}_i, \top\}, \{\mathtt{prof}, \top\}).$$

Since $\{\mathtt{st}_i, \top\} \leqq \{\mathtt{prof}, \mathtt{st}_i, \top\}$ and $\{\mathtt{st}_i, \top\}\natural\{\mathtt{prof}, \top\} = \{\mathtt{prof}, \top\}$, we conclude by (TreeEdge) that

$$\vdash \mathtt{service}^{\{\mathtt{st}_i,\top\}}[T_1 \mid T_2] : Tree(\sigma_{\mathtt{F}}, \mathcal{E}_{\mathtt{F}}, \mathcal{D}_{\mathtt{F}}, \{\mathtt{st}_i, \top\}, \{\mathtt{prof}, \top\}).$$

Notice that all the tree types are well formed since the order on current roles is $\mathtt{st}_i \sqsubset \mathtt{prof} \sqsubset \top$.

3.2 Typing rules for processes

The typing rule for channels is as expected

$$\frac{}{\mathit{\Gamma} \vdash c^{Tv} : Ch(Tv)} \ \text{(Chan)}$$

There are two type assignments for processes, one for pure processes and one for processes with roles, given in Table 11.

Typing rules for pure processes. In the typing rules (In), (Out) and (Rep) we need the notion of *characteristic roles* of a value type Tv (notation $\mathcal{C}(Tv)$) as defined below:

$$\begin{array}{ll} \mathcal{C}(Ch(Tv)) = \mathcal{C}(Tv) & \mathcal{C}(Script(\sigma, \mathcal{E}, \mathcal{D})) = \mathcal{C}(Loc(\sigma, \mathcal{E}, \mathcal{D})) = \{\bot\} \\ \mathcal{C}(Tree(\sigma, \mathcal{E}, \mathcal{D}, \tau, \zeta)) = \tau & \mathcal{C}(Path(\alpha)) = \alpha. \end{array}$$

Typing rules (In), (Out) and (Rep) assure that a process with roles ρ can communicate only values with at least one characteristic role that is less than or equal to a role in ρ.

In rules (Read) and (Change) we use the environment $\mathit{\Gamma}_\chi$ defined by:

$$\mathit{\Gamma}_\chi = \begin{cases} x : ProcRole(\sigma, \mathcal{E}, \mathcal{D}) \text{ if } \chi = \Box x^{(\sigma,\mathcal{E},\mathcal{D})}, \\ x : Loc(\sigma, \mathcal{E}, \mathcal{D}), y : Path(\alpha) \text{ if } \chi = y^{(\alpha)} @ x^{(\sigma,\mathcal{E},\mathcal{D})}, \\ x : Tree(\sigma, \mathcal{E}, \mathcal{D}, \tau, \zeta) \text{ if } \chi = x^{(\sigma,\mathcal{E},\mathcal{D},\tau,\zeta)} \end{cases}$$

that assigns types to the variables of the pattern χ.

If α is the set of roles of the last edge in a path and the path complies with a tree path, the condition $\alpha \leq \rho$ assures that a process with roles ρ can access the tree path. This condition is sufficient, since the path complies with the tree path means that if τ is the set of roles of the edge connecting the identified data with his father than $\tau \leqq \alpha$, so we conclude $\tau \leq \rho$ as required.

We replace a data term by a tree only if the tree obtained in that way is well typed. This is checked by the condition $\alpha \leqq \tau'$ in rule (Change). When we replace a subtree (i.e. when $\chi = x^{(\sigma,\mathcal{E},\mathcal{D},\tau,\zeta)}$) the condition $\zeta \leq \rho$ assures that the whole subtree is accessible to the process.

With (Enable) and (Disable) we change roles of edges. Rule (Enable) assures that the roles ρ can add the role r according to the location policy. Similarly, rule (Disable) assures that the roles ρ can erase the role r according to the location policy. In rule (Enable) the condition $\alpha \leqq \{r\}$ gives a well-typed tree as a result. In rule (Disable) this condition is not needed.

Example 9. Let the tree in the `FACULTY` location be as in Example 1. A professor can give the permission to a teaching assistant to edit the records by the process

$$P_{\mathtt{P}} = \mathtt{enable}_{\mathtt{service}^{\{\mathtt{st}_i\}}}(\mathtt{as}).$$

The process $P_{\mathtt{P}}$ can be typed by $Proc(\sigma_{\mathtt{F}}, \mathcal{E}_{\mathtt{F}}, \mathcal{D}_{\mathtt{F}}, \{\mathtt{prof}\})$ because $(\{\mathtt{prof}\}, \mathtt{as}) \in \mathcal{E}_{\mathtt{F}}$, i.e. it respects the location policy, and both `prof` and `as` can access the path. Similarly, the dean can forbid the student i to access his booklet by the process

$$P_{\mathtt{D}} = \mathtt{disable}_{\mathtt{service}^{\{\mathtt{dean}\}}}(\mathtt{st}_i).$$

Because $(\{\mathtt{dean}\}, \mathtt{st}_i) \in \mathcal{D}_{\mathtt{F}}$ the type of $P_{\mathtt{D}}$ can be $Proc(\sigma_{\mathtt{F}}, \mathcal{E}_{\mathtt{F}}, \mathcal{D}_{\mathtt{F}}, \{\mathtt{dean}\})$.

$$\frac{}{\mathit{\Gamma} \vdash 0 : \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)}\ (\text{Nil})$$

$$\frac{\mathit{\Gamma} \vdash P_1 : \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho) \quad \mathit{\Gamma} \vdash P_2 : \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)}{\mathit{\Gamma} \vdash P_1 \mid P_2 : \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)}\ (\text{Par})$$

$$\frac{\mathit{\Gamma} \vdash \gamma : \mathit{Ch}(Tv) \quad \mathit{\Gamma} \vdash v : Tv \quad \mathcal{C}(Tv) \leq \rho}{\mathit{\Gamma} \vdash \bar{\gamma}\langle v \rangle : \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)}\ (\text{Out})$$

$$\frac{\mathit{\Gamma} \vdash \gamma : \mathit{Ch}(Tv) \quad \mathit{\Gamma}, x : Tv \vdash P : \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho) \quad \mathcal{C}(Tv) \leq \rho}{\mathit{\Gamma} \vdash \gamma(x).P : \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)}\ (\text{In})$$

$$\frac{\mathit{\Gamma} \vdash \gamma : \mathit{Ch}(Tv) \quad \mathit{\Gamma}, x : Tv \vdash P : \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho) \quad \mathcal{C}(Tv) \leq \rho}{\mathit{\Gamma} \vdash !\gamma(x).P : \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)}\ (\text{Rep})$$

$$\frac{\mathit{\Gamma} \vdash R : \mathit{ProcRole}(\sigma, \mathcal{E}, \mathcal{D}) \quad \mathit{\Gamma} \vdash \lambda : \mathit{Loc}(\sigma, \mathcal{E}, \mathcal{D})}{\mathit{\Gamma} \vdash \mathtt{go}\ \lambda.R : \mathit{Proc}(\sigma', \mathcal{E}', \mathcal{D}', \rho)}\ (\text{Go})$$

$$\frac{\mathit{\Gamma} \vdash p : \mathit{Path}(\alpha) \quad \alpha \leq \rho}{\mathit{\Gamma} \vdash \mathtt{run}_p : \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)}\ (\text{Run})$$

$$\frac{\mathit{\Gamma} \vdash p : \mathit{Path}(\alpha) \ \ \mathit{\Gamma} \cup \mathit{\Gamma}_\chi \vdash P : \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho) \quad \alpha \leq \rho}{\mathit{\Gamma} \vdash \mathtt{read}_p(\chi).P : \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)}\ (\text{Read})$$

$$\frac{\begin{array}{c} \mathit{\Gamma} \vdash p : \mathit{Path}(\alpha) \quad \mathit{\Gamma} \vdash P : \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho) \quad \alpha \leq \rho \\ \mathit{\Gamma} \cup \mathit{\Gamma}_\chi \vdash \begin{cases} V : \mathit{Script}(\sigma, \mathcal{E}, \mathcal{D}) \text{ or} \\ V : \mathit{Pointer}(\beta) \text{ or} \\ V : \mathit{Tree}(\sigma, \mathcal{E}, \mathcal{D}, \tau', \zeta') \quad \alpha \leqq \tau' \end{cases} \\ \text{if } \chi = x^{(\sigma, \mathcal{E}, \mathcal{D}, \tau, \zeta)} \text{ then } \zeta \leq \rho \end{array}}{\mathit{\Gamma} \vdash \mathtt{change}_p(\chi, V).P : \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)}\ (\text{Change})$$

$$\frac{\mathit{\Gamma} \vdash p : \mathit{Path}(\alpha) \ \ \mathit{\Gamma} \vdash P : \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho) \quad (\rho, r) \in^+ \mathcal{E} \quad \alpha \leq \rho \quad \alpha \leqq \{r\}}{\mathit{\Gamma} \vdash \mathtt{enable}_p(r).P : \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)}\ (\text{Enable})$$

$$\frac{\mathit{\Gamma} \vdash p : \mathit{Path}(\alpha) \ \ \mathit{\Gamma} \vdash P : \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho) \quad (\rho, r) \in^- \mathcal{D} \quad \alpha \leq \rho}{\mathit{\Gamma} \vdash \mathtt{disable}_p(r).P : \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)}\ (\text{Disable})$$

$$\frac{\mathit{\Gamma} \vdash P : \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)}{\mathit{\Gamma} \vdash P^{\neg\rho} : \mathit{ProcRole}(\sigma, \mathcal{E}, \mathcal{D})}\ (\text{Role}) \qquad \frac{\mathit{\Gamma} \vdash R : \mathit{ProcRole}(\sigma, \mathcal{E}, \mathcal{D})}{\mathit{\Gamma} \vdash (\nu c^{Tv})R : \mathit{ProcRole}(\sigma, \mathcal{E}, \mathcal{D})}\ (\text{Res})$$

$$\frac{\mathit{\Gamma} \vdash R_1 : \mathit{ProcRole}(\sigma, \mathcal{E}, \mathcal{D}) \quad \mathit{\Gamma} \vdash R_2 : \mathit{ProcRole}(\sigma, \mathcal{E}, \mathcal{D})}{\mathit{\Gamma} \vdash R_1 \mid R_2 : \mathit{ProcRole}(\sigma, \mathcal{E}, \mathcal{D})}\ (\text{ParR})$$

Table 11. Typing rules for pure processes and processes with roles

Typing rules for processes with roles. If a pure process P is well typed for locations with some policy and a set of roles ρ, then the process with roles $P^{\urcorner\rho}$ is well typed for locations with the same policy.

3.3 Typing rules for networks

$$\frac{\vdash l : Loc(\sigma,\mathcal{E},\mathcal{D}) \quad \vdash T : Tree(\sigma,\mathcal{E},\mathcal{D},\tau,\zeta) \quad \vdash R : ProcRole(\sigma,\mathcal{E},\mathcal{D})}{\vdash l[\![\, T \parallel R \,]\!] : Net} \ (\text{NetLoc})$$

$$\frac{}{\vdash \mathbf{0} : Net} \ (\text{NetNil}) \qquad \frac{\vdash \mathbf{N} : Net}{\vdash (\nu c^{Tv})\mathbf{N} : Net} \ (\text{NetRes})$$

$$\frac{\vdash \mathbf{N}_1 : Net \quad \vdash \mathbf{N}_2 : Net \quad \mathcal{N}(\mathbf{N}_1) \cap \mathcal{N}(\mathbf{N}_2) = \emptyset}{\vdash \mathbf{N}_1 \mathbin{|||} \mathbf{N}_2 : Net} \ (\text{NetPar})$$

Table 12. Typing rules for networks

In typing rules for networks, given in Table 12, a location l is well typed if it consists of a tree and a process with roles that are well typed for locations with the policy of l. The function $\mathcal{N}$ associates to a network the set of its location names:

$$\mathcal{N}(\mathbf{0}) = \emptyset \qquad \mathcal{N}(l[\![\, T \parallel P \,]\!]) = \{l\} \qquad \mathcal{N}(\mathbf{N}_1 \mathbin{|||} \mathbf{N}_2) = \mathcal{N}(\mathbf{N}_1) \cup \mathcal{N}(\mathbf{N}_2).$$

It is used in rule (NetPar) to assure that each location name occurs at most once in a well-typed network.

A straightforward consequence of the type assignment rules are the following properties:

Proposition 1. *(i) Each location name occurs at most once in a well-typed network.*
(ii) If a location $l[\![\, T \parallel R \,]\!]$ is well typed, then both the tree T and the process with roles R do not contain occurrences of free variables.

The system satisfies subject reduction:

Theorem 1 (Subject reduction). *Let* $\vdash \mathbf{N} : Net$ *and* $\mathbf{N} \to \mathbf{N}'$*, then* $\vdash \mathbf{N}' : Net$.

The proof is the content of Appendix A. It uses some *Generation* and *Substitution* lemmas which are also presented in Appendix A.

4 Access Control

In this section, using subject reduction, we discuss the role based access control and the policy maintenance of well-typed networks. More precisely, we show the following properties.

Properties of location policies and communication:

P0 All trees and processes in a location agree with the location policy;
P1 A process with roles can communicate only values with at least one characteristic role lower than or equal to one role of the process.

Properties of migration between locations:

P2 A process with roles can migrate to another location only if it agrees with the policy of that location.

Properties of process access to local data trees:

P3 A process with roles looks for a path in the local tree only if the path is accessible to the process.
P4 A process with roles can get a data in the local tree only if the data is accessible to the process.

Properties of manipulation of local data trees by processes:

P5 A script is activated in a location only if the corresponding process with roles agrees with the policy of that location;
P6 A process with roles generated by a read command in a location agrees with the policy of that location;
P7 A process with roles can erase a subtree of data only if it can access the whole data;
P8 A tree built by a change command in a location agrees with the policy of that location;
P9 A process with roles can add a role to an edge in the local tree only if this is allowed by the location policy;
P10 A tree built by an enable command in a location agrees with the policy of that location;
P11 A process with roles can erase a role from a subtree of the local tree only if this is allowed by the location policy;
P12 A tree built by a disable command in a location agrees with the policy of that location.

In order to formalise and prove these properties, we use $\rightarrowtriangle$ to denote the reflexive and transitive closure of $\rightarrow$ and $\boldsymbol{\nu}$ to denote a possibly empty sequence of channel restrictions. We can then express these properties by means of the following proposition:

Proposition 2. *If* $\mathbf{N}$ *is a well typed network and* $\mathbf{N} \rightarrowtriangle \boldsymbol{\nu}(l[\![\ T \parallel P^{\neg\rho} \mid R\ ]\!] \ \mathbin{|\!|\!|}\ \mathbf{N}')$*, then:*

P0 $\vdash T : Tree(\sigma, \mathcal{E}, \mathcal{D}, \tau, \zeta)$ *for some* τ, ζ *and* $\vdash P^{\neg\rho} \mid R : ProcRole(\sigma, \mathcal{E}, \mathcal{D})$*;*

P1 $P \equiv \bar{c}^{Tv}\langle v\rangle$ *implies* $\vdash v : Tv$ *and* $\mathcal{C}(Tv) \leq \rho$;

P2 $P \equiv \texttt{go}\ l'.R$ *implies* $\vdash R : ProcRole(\sigma', \mathcal{E}', \mathcal{D}')$ *and* $\mathcal{T}(l') = (\sigma', \mathcal{E}', \mathcal{D}')$*;*

P3 $P \equiv \texttt{run}_p$ *or* $P \equiv \texttt{read}_p(x).P'$ *or* $P \equiv \texttt{change}_p(\chi, V).P'$ *or* $P \equiv \texttt{enable}_p(r).P'$ *or* $P \equiv \texttt{disable}_p(r).P'$ *and* $\vdash p : Path(\alpha)$ *imply* $\alpha \leq \rho$;

P4 $P \equiv \texttt{run}_p$ *or* $P \equiv \texttt{read}_p(x).P'$ *or* $P \equiv \texttt{change}_p(\chi, V).P'$ *or* $P \equiv \texttt{enable}_p(r).P'$ *or* $P \equiv \texttt{disable}_p(r).P'$ *and* p *identifies a data term* V *in the tree* T *imply that* V *is connected to his father by an edge whose set of roles* τ *is such that* $\tau \leq \rho$*;*

P5 $P \equiv \texttt{run}_p$ *and* $\texttt{rn}(p, T) = \{R_1, \ldots, R_n\}$ *imply* $\vdash R_i : ProcRole(\sigma, \mathcal{E}, \mathcal{D})$ *for* $1 \leq i \leq n$*;*

P6 $P \equiv \texttt{read}_p(\chi).P'$ *and* $\texttt{sub}(p, \chi, T) = \{\texttt{s}_1, \ldots, \texttt{s}_n\}$ *imply* $\vdash P\texttt{s}_i : Proc(\sigma, \mathcal{E}, \mathcal{D}, \rho)$ *for* $1 \leq i \leq n$*;*

P7 $P \equiv \texttt{change}_p(x^{(\sigma', \mathcal{E}', \mathcal{D}', \tau, \zeta)}, V).P'$ *implies* $\zeta \leq \rho$*;*

P8 $P \equiv \texttt{change}_p(\chi, V).P'$ *and* $\texttt{ch}(p, \chi, V, T) = T'$ *imply* $\vdash T' : Tree(\sigma, \mathcal{E}, \mathcal{D}, \tau, \zeta')$ *for some* ζ'*;*

P9 $P \equiv \texttt{enable}_p(r).P'$ *implies* $(\rho, r) \in^{+} \mathcal{E}$;

P10 $P \equiv \texttt{enable}_p(r).P'$ *and* $\texttt{en}(p, r, T) = T'$ *imply* $\vdash T' : Tree(\sigma, \mathcal{E}, \mathcal{D}, \tau, \zeta')$ *for some* ζ'*;*

P11 $P \equiv \texttt{disable}_p(r).P'$ *implies* $(\rho, r) \in^{-} \mathcal{D}$;

P12 $P \equiv \texttt{disable}_p(r).P'$ *and* $\texttt{di}(p, r, T) = T'$ *imply* $\vdash T' : Tree(\sigma, \mathcal{E}, \mathcal{D}, \tau, \zeta')$ *for some* ζ'*;*

where $\mathcal{T}(l) = (\sigma, \mathcal{E}, \mathcal{D})$*.*

The proof of this proposition is the content of Appendix B.

5 Related work and conclusion

RBAC has been introduced in the seventies and first formalised by Ferraiolo and Kuhn [10]. There is a large literature on models and implementations for RBAC, we only mention [24, 9, 23, 11]. The standard defined in 2004 is currently under revision by the Committe CS1.1 within the International Committee for Information Technologies Standards [18].

Access control has been studied in various forms for many calculi modelling concurrent and distributed systems. Sophisticated types controlling the use of resources and the mobility of processes have been proposed for the $D\pi$ calculus [17, 15]. In $D\pi$ the resources are channels which support binary communication between processes. The typing system guarantees that distributed processes cannot access the resources without first being granted the capability to do so. Processes can augment their sets of capabilities via communication with other processes. In the SafeDpi calculus [16] parameterised code may be sent between locations and types restrict the capabilities and access rights of any processes launched by incoming code. The paper [8] discusses a type system for the X$d\pi$ calculus based on security levels: security levels in data trees are assigned only to the data in the leaves. Co-actions have been introduced for ambient calculi as a basic mechanism for regulating access to locations and use of their resources [19, 5, 13]. More refined controls for ambient calculi include passwords [21, 3], classifications in groups [4, 7], mandatory access control policies [2], membranes regulating the interaction between computing bodies and external environments [14].

The most related papers are [1] and [6]. Braghin et al. equip the π-calculus with the notion of user: they tag processes with names of users and with sets of roles. Processes can activate and deactivate roles. A mapping between roles and users and a mapping between read/write actions and roles control access rights. A type discipline statically guaranties that systems not respecting the above mappings are rejected. Compagnoni et al. define a boxed ambient calculus extended with a distributed RBAC mechanism where each ambient controls its own access policy. A process is associated with an owner and a set of activated roles that grant permissions for mobility and communication. The calculus includes primitives to activate and deactivate roles. The behaviour of these primitives is determined by the process's owner, its current location and its currently activated roles.

We discussed a model in which the pure processes are the users, the permissions are the accesses to data in trees and the administration policies of locations prescribe how the association between roles and data can change. Note that we do not have user identifiers, and so we cannot activate and deactivate roles for users. Our design choice is motivated by the focus on the interaction between processes and data trees and to the best of our knowledge this is a first attempt in this dirrection.

Other common features of RBAC system we did not consider here, since we could smoothly add them to the present calculus, are: incompatible roles, static and dynamic separation of roles, limits on the number of users authorised for a given role.

Future work includes the study of both type checking and type inference for the present calculus. We will also compare networks using the behavioural equivalences, extending the work in [20].

Acknowledgment. We gratefully thank the referees: the current version of the paper strongly improved due to referees' useful suggestions.

References

1. Chiara Braghin, Daniele Gorla, and Vladimiro Sassone. Role-based access control for a distributed calculus. *Journal of Computer Security*, 14(2):113–155, 2006.
2. Michele Bugliesi, Giuseppe Castagna, and Silvia Crafa. Access control for mobile agents: The calculus of boxed ambients. *ACM Transactions on Programming Languages and Systems*, 26(1):57–124, 2004.
3. Michele Bugliesi, Silvia Crafa, Massimo Merro, and Vladimiro Sassone. Communication and mobility control in boxed ambients. *Information and Computation*, 202(1):39–86, 2005.
4. Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Types for the ambient calculus. *Information and Computation*, 177(2):160–194, 2002.
5. Giuseppe Castagna, Jan Vitek, and Francesco Zappa Nardelli. The Seal calculus. *Information and Computation*, 201(1):1–54, 2005.
6. Adriana B. Compagnoni, Elsa L. Gunter, and Philippe Bidinger. Role-based access control for boxed ambients. *Theoretical Computer Science*, 398(1-3):203–216, 2008.
7. Mario Coppo, Mariangiola Dezani-Ciancaglini, and Elio Giovannetti. Types for ambient and process mobility. *Mathematical Structures in Computer Science*, 18:221–290, 2008.
8. Mariangiola Dezani-Ciancaglini, Silvia Ghilezan, Jovanka Pantovic, and Daniele Varacca. Security types for dynamic web data. *Theoretical Computer Science*, 402(2-3):156–171, 2008.

9. David F. Ferraiolo, John F. Barkley, and D. Richard Kuhn. A role-based access control model and reference implementation within a corporate intranet. *ACM Transactions on Information and System Security*, 2(1):34–64, 1999.
10. David F. Ferraiolo, D. Richard Kuhn, and Ravi S. Sandhu. Rôle-based access control. In *NIST-NSA National Computer Security Conference*, pages 554–563, 1992.
11. David F. Ferraiolo, Ravi S. Sandhu, Serban I. Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security*, 4(3):224–274, 2001.
12. Philippa Gardner and Sergio Maffeis. Modelling dynamic web data. *Theoretical Computer Science*, 342(1):104–131, 2005.
13. Pablo Garralda, Eduardo Bonelli, Adriana Compagnoni, and Mariangiola Dezani-Ciancaglini. Boxed ambients with communication interfaces. *Mathematical Structures in Computer Science*, 17:1–59, 2007.
14. Daniele Gorla, Matthew Hennessy, and Vladimiro Sassone. Security policies as membranes in systems for global computing. *Logical Methods in Computer Science*, 1(3:2):331353, 2005.
15. Matthew Hennessy. *A Distributed Pi-calculus*. Cambridge University Press, 2007.
16. Matthew Hennessy, Julian Rathke, and Nobuko Yoshida. SafeDpi: A language for controlling mobile code. *Acta Informatica*, 42(4-5):227–290, 2005.
17. Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. *Information and Computation*, 173(1):82–120, 2002.
18. D. Richard Kuhn, Edward J. Coyne, and Timothy R. Weil. Adding attributes to role-based access control. *Computer*, 43(6):79–81, 2010.
19. Francesca Levi and Davide Sangiorgi. Controlling interference in ambients. *Transactions on Programming Languages and Systems*, 25(1):1–69, 2003.
20. Sergio Maffeis and Philippa Gardner. Behavioural equivalences for dynamic Web data. *Journal of Logic and Algebraic Programming*, 75(1):86–138, 2008.
21. Massimo Merro and Matthew Hennessy. A bisimulation-based semantic theory of safe ambients. *ACM Transactions on Programming Languages and Systems*, 28(2):290–330, 2006.
22. Robin Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.
23. Sylvia Osborn, Ravi S. Sandhu, and Qamar Munawer. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Transactions on Information and System Security*, 3(2):85–106, 2000.
24. Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

A Subject Reduction

We prove that the typing of networks is preserved by structural equivalence and by reduction. These proofs use generation lemmas which allow the reversal of the typing rules. We use ω to range over all the types defined in Table 9.

Lemma 1 (Generation lemma for variables, locations, channels, scripts and pointers).

1. $\Gamma \vdash x : \omega \Rightarrow x : \omega \in \Gamma$.
2. $\Gamma \vdash l : \omega \Rightarrow \omega = Loc(\sigma, \mathcal{E}, \mathcal{D})$ *and* $\mathcal{T}(l) = (\sigma, \mathcal{E}, \mathcal{D})$.

3. $\Gamma \vdash c^{Tv} : \omega \Rightarrow \omega = Ch(Tv)$.
4. $\Gamma \vdash \Box\Pi : \omega \Rightarrow \omega = \mathit{Script}(\sigma, \mathcal{E}, \mathcal{D})$ and $\Gamma \vdash \Pi : \mathit{ProcRole}(\sigma, \mathcal{E}, \mathcal{D})$.
5. $\Gamma \vdash p@\lambda : \omega \Rightarrow \omega = \mathit{Pointer}(\alpha)$ and $\Gamma \vdash p : \mathit{Path}(\alpha)$ and $\Gamma \vdash \lambda : \mathit{Loc}(\sigma, \mathcal{E}, \mathcal{D})$.

Lemma 2 (Generation lemma for paths).

1. $\Gamma \vdash \mathtt{a}^{\alpha} : \omega \Rightarrow \omega = \mathit{Path}(\alpha)$.
2. $\Gamma \vdash p_1/p_2 : \omega \Rightarrow \omega = \mathit{Path}(\alpha)$ and $\Gamma \vdash p_1 : \mathit{Path}(\beta)$ and $\Gamma \vdash p_2 : \mathit{Path}(\alpha)$.

Lemma 3 (Generation lemma for trees).

1. $\Gamma \vdash \emptyset_T : \omega \Rightarrow \omega = \mathit{Tree}(\sigma, \mathcal{E}, \mathcal{D}, \{\top\}, \{\bot, \top\})$.
2. $\Gamma \vdash \mathtt{a}^{\tau}[p@\lambda] : \omega \Rightarrow \omega = \mathit{Tree}(\sigma, \mathcal{E}, \mathcal{D}, \tau, \tau)$ and $\Gamma \vdash p@\lambda : \mathit{Pointer}(\alpha)$ and $\Gamma \vdash \lambda : \mathit{Loc}(\sigma', \mathcal{E}', \mathcal{D}')$.
3. $\Gamma \vdash \mathtt{a}^{\tau}[\Box\Pi] : \omega \Rightarrow \omega = \mathit{Tree}(\sigma, \mathcal{E}, \mathcal{D}, \tau, \tau)$ and $\Gamma \vdash \Box\Pi : \mathit{Script}(\sigma, \mathcal{E}, \mathcal{D})$.
4. $\Gamma \vdash \mathtt{a}^{\tau}[T] : \omega \Rightarrow \omega = \mathit{Tree}(\sigma, \mathcal{E}, \mathcal{D}, \tau, \zeta)$ and $\Gamma \vdash T : \mathit{Tree}(\sigma, \mathcal{E}, \mathcal{D}, \tau', \zeta')$ and $\tau \leqq \tau'$ and $\tau \natural \zeta' = \zeta$.
5. $\Gamma \vdash T_1 \mid T_2 : \omega \Rightarrow \omega = \mathit{Tree}(\sigma, \mathcal{E}, \mathcal{D}, \tau_1 \cup \tau_2, \zeta_1 \sqcup \zeta_2)$ and $\Gamma \vdash T_1 : \mathit{Tree}(\sigma, \mathcal{E}, \mathcal{D}, \tau_1, \zeta_1)$ and $\Gamma \vdash T_1 : \mathit{Tree}(\sigma, \mathcal{E}, \mathcal{D}, \tau_2, \zeta_2)$.

Lemma 4 (Generation lemma for processes).

1. $\Gamma \vdash 0 : \omega \Rightarrow \omega = \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)$.
2. $P_1 \mid P_2 : \omega \Rightarrow \omega = \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)$ and $\Gamma \vdash P_1 : \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)$ and $\Gamma \vdash P_2 : \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)$.
3. $\Gamma \vdash \bar{\gamma}\langle v \rangle : \omega \Rightarrow \omega = \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)$ and $\Gamma \vdash \gamma : Ch(Tv)$ and $\Gamma \vdash v : Tv$ and $\mathcal{C}(Tv) \leq \rho$.
4. $\Gamma \vdash \gamma(x).P : \omega \Rightarrow \omega = \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)$ and $\Gamma, x : Tv \vdash P : \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)$ and $\Gamma \vdash \gamma : Ch(Tv)$ and $\mathcal{C}(Tv) \leq \rho$.
5. $\Gamma \vdash !\gamma(x).P : \omega \Rightarrow \omega = \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)$ and $\Gamma, x : Tv \vdash P : \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)$ and $\Gamma \vdash \gamma : Ch(Tv)$ and $\mathcal{C}(Tv) \leq \rho$.
6. $\Gamma \vdash \mathtt{go}\ \lambda.R : \omega \Rightarrow \omega = \mathit{Proc}(\sigma', \mathcal{E}', \mathcal{D}', \rho)$ and $\Gamma \vdash \lambda : \mathit{Loc}(\sigma, \mathcal{E}, \mathcal{D})$ and $\Gamma \vdash R : \mathit{ProcRole}(\sigma, \mathcal{E}, \mathcal{D})$.
7. $\Gamma \vdash \mathtt{run}_p : \omega \Rightarrow \omega = \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)$ and $\Gamma \vdash p : \mathit{Path}(\alpha)$ and $\alpha \leq \rho$.
8. $\Gamma \vdash \mathtt{read}_p(\chi).P : \omega \Rightarrow \omega = \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)$ and $\Gamma \vdash p : \mathit{Path}(\alpha)$ and $\Gamma \cup \Gamma_\chi \vdash P : \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)$ and $\alpha \leq \rho$.
9. $\Gamma \vdash \mathtt{change}_p(\chi, V).P : \omega \Rightarrow \omega = \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)$ and $\Gamma \vdash p : \mathit{Path}(\alpha)$ and $\Gamma \vdash P : \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)$ and $\alpha \leq \rho$ and ($\Gamma \cup \Gamma_\chi \vdash V : \mathit{Script}(\sigma, \mathcal{E}, \mathcal{D})$ or $\Gamma \cup \Gamma_\chi \vdash V : \mathit{Pointer}(\beta)$ or ($\Gamma \cup \Gamma_\chi \vdash V : \mathit{Tree}(\sigma, \mathcal{E}, \mathcal{D}, \tau', \zeta')$ and $\alpha \leqq \tau'$)) and (if $\chi = x^{(\sigma, \mathcal{E}, \mathcal{D}, \tau, \zeta)}$ then $\zeta \leq \rho$).
10. $\Gamma \vdash \mathtt{enable}_p(r).P : \omega \Rightarrow \omega = \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)$ and $\Gamma \vdash p : \mathit{Path}(\alpha)$ and $\Gamma \vdash P : \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)$ and $(\rho, r) \in^{+} \mathcal{E}$ and $\alpha \leq \rho$ and $\alpha \leqq \{r\}$.
11. $\Gamma \vdash \mathtt{disable}_p(r).P : \omega \Rightarrow \omega = \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)$ and $\Gamma \vdash p : \mathit{Path}(\alpha)$ and $\Gamma \vdash P : \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)$ and $(\rho, r) \in^{-} \mathcal{D}$ and $\alpha \leq \rho$.

Lemma 5 (Generation lemma for processes with roles).

1. $\Gamma \vdash P^{\urcorner\rho} : \omega \Rightarrow \omega = \mathit{ProcRole}(\sigma, \mathcal{E}, \mathcal{D})$ and $\Gamma \vdash P : \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)$.

2. *$\mathit{\Gamma} \vdash R_1 \mid R_2 : \omega \Rightarrow \omega = ProcRole(\sigma,\mathcal{E},\mathcal{D})$ and $\mathit{\Gamma} \vdash R_1 : ProcRole(\sigma,\mathcal{E},\mathcal{D})$ and $\mathit{\Gamma} \vdash R_2 : ProcRole(\sigma,\mathcal{E},\mathcal{D})$.*
3. *$\mathit{\Gamma} \vdash (\nu c^{Tv})R : \omega \Rightarrow \omega = ProcRole(\sigma,\mathcal{E},\mathcal{D})$ and $\mathit{\Gamma} \vdash R : ProcRole(\sigma,\mathcal{E},\mathcal{D})$.*

Lemma 6 (Generation lemma for networks).

1. *$\vdash l[\![\, T \parallel R \,]\!] : \omega \Rightarrow \omega = Net$ and $\vdash l : Loc(\sigma,\mathcal{E},\mathcal{D})$ and $\vdash T : Tree(\sigma,\mathcal{E},\mathcal{D},\tau,\zeta)$ and $\vdash R : ProcRole(\sigma,\mathcal{E},\mathcal{D})$.*
2. *$\vdash \mathbf{0} : \omega \Rightarrow \omega = Net$.*
3. *$\vdash (\nu c^{Tv})\mathbf{N} : \omega \Rightarrow \omega = Net$ and $\vdash \mathbf{N} : Net$.*
4. *$\vdash \mathbf{N}_1 \;|\!|\!|\; \mathbf{N}_2 : \omega \Rightarrow \omega = Net$ and $\vdash \mathbf{N}_1 : Net$ and $\vdash \mathbf{N}_2 : Net$ and $\mathcal{N}(\mathbf{N}_1) \cap \mathcal{N}(\mathbf{N}_2) = \emptyset$.*

Lemma 7 (Substitution lemma for processes and data terms).

1. *If $\mathit{\Gamma}, x : Tv \vdash P : Proc(\sigma,\mathcal{E},\mathcal{D},\rho)$ and $\mathit{\Gamma} \vdash v : Tv$, then $\mathit{\Gamma} \vdash P\{v/x\} : Proc(\sigma,\mathcal{E},\mathcal{D},\rho)$.*
2. *If $\mathit{\Gamma} \cup \mathit{\Gamma}_\chi \vdash P : Proc(\sigma,\mathcal{E},\mathcal{D},\rho)$ and $\mathtt{match}(\chi, V)$, then $\mathit{\Gamma} \vdash P\{\!\{V/|\chi|\}\!\} : Proc(\sigma,\mathcal{E},\mathcal{D},\rho)$.*
3. *If $\mathit{\Gamma} \cup \mathit{\Gamma}_\chi \vdash \Box\mathit{\Pi} : Script(\sigma,\mathcal{E},\mathcal{D})$ and $\mathtt{match}(\chi, V)$, then $\mathit{\Gamma} \vdash \Box\mathit{\Pi}\{\!\{V/|\chi|\}\!\} : Script(\sigma,\mathcal{E},\mathcal{D})$.*
4. *If $\mathit{\Gamma} \cup \mathit{\Gamma}_\chi \vdash p@\lambda : Pointer(\alpha)$ and $\mathtt{match}(\chi, V)$, then $\mathit{\Gamma} \vdash p@\lambda\{\!\{V/|\chi|\}\!\} : Pointer(\alpha)$.*
5. *If $\mathit{\Gamma} \cup \mathit{\Gamma}_\chi \vdash T : Tree(\sigma,\mathcal{E},\mathcal{D},\tau,\zeta)$ and $\mathtt{match}(\chi, V)$, then $\mathit{\Gamma} \vdash T\{\!\{V/|\chi|\}\!\} : Tree(\sigma,\mathcal{E},\mathcal{D},\tau,\zeta)$.*

Lemma 8 (Properties of trees).

1. *If $\vdash T : Tree(\sigma,\mathcal{E},\mathcal{D},\tau,\zeta)$ and $\Box R$ occurs in T, then $\vdash R : ProcRole(\sigma,\mathcal{E},\mathcal{D})$.*
2. *If $\vdash T : Tree(\sigma,\mathcal{E},\mathcal{D},\tau,\zeta)$ and T' is the tree obtained from T by replacing a data term with a well-typed pointer or with a script of type $Script(\sigma,\mathcal{E},\mathcal{D})$ from the empty environment, then $\vdash T' : Tree(\sigma,\mathcal{E},\mathcal{D},\tau,\zeta')$ for some ζ'.*
3. *If $\vdash T : Tree(\sigma,\mathcal{E},\mathcal{D},\tau,\zeta)$ and T' is the tree obtained from T by replacing a data term identified by a path p such that $\vdash p : Path(\alpha)$ with a tree T_0 such that $\vdash T_0 : Tree(\sigma,\mathcal{E},\mathcal{D},\tau_0,\zeta_0)$ and $\alpha \leqq \tau_0$, then $\vdash T' : Tree(\sigma,\mathcal{E},\mathcal{D},\tau,\zeta')$ for some ζ'.*
4. *If $\vdash T : Tree(\sigma,\mathcal{E},\mathcal{D},\tau,\zeta)$ and T' is the tree obtained from T by replacing a tree T_0 identified by a path p such that $\vdash p : Path(\alpha)$ with the tree $(T_0)^{+r}$ and $\alpha \leqq \{r\}$, then $\vdash T' : Tree(\sigma,\mathcal{E},\mathcal{D},\tau,\zeta')$ for some ζ'.*
5. *If $\vdash T : Tree(\sigma,\mathcal{E},\mathcal{D},\tau,\zeta)$ and T' is the tree obtained from T by replacing a tree T_0 with the tree $(T_0)^{-r}$, then $\vdash T' : Tree(\sigma,\mathcal{E},\mathcal{D},\tau,\zeta')$ for some ζ'.*

Proof. 1. By induction on the tree T using Lemma 3(5), (4) and (3) we get that $\vdash \Box R : Script(\sigma,\mathcal{E},\mathcal{D})$, so we conclude by Lemma 1(4).

2. Easy from the typing rules of trees.
3. Let a^{τ_1} be the edge connecting the data term to his father in tree T. From $\vdash p : Path(\alpha)$ and from the definition of compliance we get $\tau_1 \leqq \alpha$. Since we assume $\alpha \leqq \tau_0$ we derive $\tau_1 \leqq \tau_0$, and this implies that we can derive the type $Tree(\sigma,\mathcal{E},\mathcal{D},\tau_1,\zeta_0)$ for $\mathsf{a}^{\tau_1}[T_0]$ using rule (TreeEdge). Therefore the whole tree T' is typeable.

4. Let $\mathtt{a}^{\tau_1}$ be as in the previous point, so that we get $\tau_1 \leqq \alpha$, which implies $\tau_1 \leqq \{r\}$. From $\vdash\ T\ :\ \mathit{Tree}(\sigma, \mathcal{E}, \mathcal{D}, \tau, \zeta)$ we get $\vdash\ T_0\ :\ \mathit{Tree}(\sigma, \mathcal{E}, \mathcal{D}, \tau_0, \zeta_0)$ for some τ_0, ζ_0 such that $\tau_1 \leqq \tau_0$ by Lemma 3(5) and (4). By definition of $(\ )^{+r}$ using the typing rules (TreeEdge) and (TreeComp) we have $\vdash\ (T_0)^{+r}\ :\ \mathit{Tree}(\sigma, \mathcal{E}, \mathcal{D}, \tau_0 \cup \{r\}, \zeta_0')$, where $\zeta_0' = \begin{cases} \zeta_0 \sqcup \{r\} & \text{if the hight of } T_0 \text{ is } 1, \\ \zeta_0 & \text{otherwise.} \end{cases}$. We can derive the type $\mathit{Tree}(\sigma, \mathcal{E}, \mathcal{D}, \tau_1, \zeta_0)$ for $\mathtt{a}^{\tau_1}[(T_0)^{+r}]$ using rule (TreeEdge), since by above $\tau_1 \leqq \tau_0 \cup \{r\}$.
5. The proof is similar to that of the previous point, and simpler, since by using τ_0, τ_1 with the same meaning, we immediately get $\tau_1 \leqq \tau_0 \backslash\backslash \{r\}$ from $\tau_1 \leqq \tau_0$ by definition of $\leqq$.

Theorem 1 (Subject reduction) *Let* $\vdash \mathbf{N} : \mathit{Net}$ *and* $\mathbf{N} \to \mathbf{N}'$, *then* $\vdash \mathbf{N}' : \mathit{Net}$.

Proof. We only consider some interesting cases.
• Rule ($\mathtt{com}$):

$$l[\![\, T \parallel \bar{c}^{Tv}\langle v\rangle^{\neg\rho'} \mid c^{Tv}(x).P^{\neg\rho} \mid R \,]\!] \quad\to\quad l[\![\, T \parallel P\{v/x\}^{\neg\rho} \mid R \,]\!]$$

From $\vdash \mathbf{N} : \mathit{Net}$, by Lemma 6(1), we get $\mathcal{T}(l) = (\sigma, \mathcal{E}, \mathcal{D})$, $\vdash T : \mathit{Tree}(\sigma, \mathcal{E}, \mathcal{D}, \tau, \zeta)$ and $\vdash \bar{c}^{Tv}\langle v\rangle^{\neg\rho'} \mid c^{Tv}(x).P^{\neg\rho} \mid R : \mathit{ProcRole}(\sigma, \mathcal{E}, \mathcal{D})$. Therefore, by Lemma 5(2) and (1), we have $\vdash\ \bar{c}^{Tv}\langle v\rangle\ :\ \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho')$ and $\vdash\ c^{Tv}(x).P\ :\ \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)$ and $\vdash\ R : \mathit{ProcRole}(\sigma, \mathcal{E}, \mathcal{D})$. By Lemma 4(4) and (3) it follows that $x : Tv \vdash P : \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)$ and $\vdash v : Tv$. Lemma 7(1) implies that $\vdash P\{v/x\} : \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)$, which by rule (Role) gives $\vdash\ P\{v/x\}^{\neg\rho}\ :\ \mathit{ProcRole}(\sigma, \mathcal{E}, \mathcal{D})$. Finally, we obtain $\vdash l[\![\, T \parallel P\{v/x\}^{\neg\rho} \mid R \,]\!] : \mathit{Net}$ by rules (ParR) and (NetLoc).
• Rule ($\mathtt{go}$):

$$l[\![\, T_1 \parallel \mathtt{go}\ m.R^{\neg\rho} \mid R_1 \,]\!] \ |||\ m[\![\, T_2 \parallel R_2 \,]\!] \ \to\ l[\![\, T_1 \parallel R_1 \,]\!] \ |||\ m[\![\, T_2 \parallel R \mid R_2 \,]\!]$$

From $\vdash\ \mathbf{N}\ :\ \mathit{Net}$, by Lemma 6(4), we get that $\vdash\ l[\![\, T_1 \parallel \mathtt{go}\ m.R^{\neg\rho} \mid R_1 \,]\!]\ :\ \mathit{Net}$ and $\vdash\ m[\![\, T_2 \parallel R_2 \,]\!] : \mathit{Net}$. By Lemma 6(1), it follows that $\mathcal{T}(l) = (\sigma, \mathcal{E}, \mathcal{D})$, $\vdash T_1 : \mathit{Tree}(\sigma, \mathcal{E}, \mathcal{D}, \tau, \zeta)$, $\vdash\ \mathtt{go}\ m.R^{\neg\rho} : \mathit{ProcRole}(\sigma, \mathcal{E}, \mathcal{D})$, $\vdash R_1 : \mathit{ProcRole}(\sigma, \mathcal{E}, \mathcal{D})$, $\mathcal{T}(m) = (\sigma', \mathcal{E}', \mathcal{D}')$, $\vdash T_2 : \mathit{Tree}(\sigma', \mathcal{E}', \mathcal{D}', \tau', \zeta')$ and $\vdash R_2 : \mathit{ProcRole}(\sigma', \mathcal{E}', \mathcal{D}')$. From $\vdash\ \mathtt{go}\ m.R^{\neg\rho}\ :\ \mathit{ProcRole}(\sigma, \mathcal{E}, \mathcal{D})$, by Lemma 5(1) we have $\vdash\ \mathtt{go}\ m.R\ :\ \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)$, which implies $\vdash R : \mathit{ProcRole}(\sigma', \mathcal{E}', \mathcal{D}')$ by Lemma 4(6). Hence, by rules (NetLoc) and (NetPar) we get $\vdash l[\![\, T_1 \parallel R_1 \,]\!] \ |||\ m[\![\, T_2 \parallel R \mid R_2 \,]\!] : \mathit{Net}$.
• Rule ($\mathtt{run}$):

$$l[\![\, T \parallel \mathtt{run}_p{}^{\neg\rho} \mid R \,]\!] \quad\to\quad l[\![\, T \parallel R_1 \mid \ldots \mid \ldots \mid R_n \mid R \,]\!]$$

where $\mathtt{rn}(p, T) = \{R_1, \ldots, R_n\}$. From $\vdash \mathbf{N} : \mathit{Net}$, by Lemma 6(1) we have $\mathcal{T}(l) = (\sigma, \mathcal{E}, \mathcal{D})$, $\vdash\ T\ :\ \mathit{Tree}(\sigma, \mathcal{E}, \mathcal{D}, \tau, \zeta)$ and $\vdash\ \mathtt{run}_p{}^{\neg\rho} \mid R : \mathit{ProcRole}(\sigma, \mathcal{E}, \mathcal{D})$. By Lemma 5(2) we get $\vdash R : \mathit{ProcRole}(\sigma, \mathcal{E}, \mathcal{D})$.Lemma 8(1) and the definition of $\mathtt{rn}$ imply $\vdash R_i : \mathit{ProcRole}(\sigma, \mathcal{E}, \mathcal{D})$ for $1 \leq i \leq n$, from which we derive $\vdash R_1 \mid \ldots \mid R_n : \mathit{ProcRole}(\sigma, \mathcal{E}, \mathcal{D})$ using rule (ParR). We conclude by rule (NetLoc).
• Rule ($\mathtt{read}$):

$$l[\![\, T \parallel \mathtt{read}_p(\chi).P^{\neg\rho} \mid R \,]\!] \quad\to\quad l[\![\, T \parallel P\mathtt{s}_1{}^{\neg\rho} \mid \ldots \mid P\mathtt{s}_n{}^{\neg\rho} \mid R \,]\!]$$

where $\mathtt{sub}(p,\chi,T)=\{\mathtt{s}_1,\ldots,\mathtt{s}_n\}$. By definition of $\mathtt{sub}$ we have that $\mathtt{s}_i=\{\!\{V_i/|\chi|\}\!\}$ where $\mathtt{match}(\chi,V_i)$ for $1\leq i\leq n$. From $\vdash\mathbf{N}:\mathit{Net}$, by Lemma 6(1) we get $\mathcal{T}(l)=(\sigma,\mathcal{E},\mathcal{D})$, $\vdash T:\mathit{Tree}(\sigma,\mathcal{E},\mathcal{D},\tau,\zeta)$ and $\vdash\mathtt{read}_p(\chi).P^{\neg\rho}\mid R:\mathit{ProcRole}(\sigma,\mathcal{E},\mathcal{D})$. From $\vdash\mathtt{read}_p(\chi).\ P^{\neg\rho}\ \mid\ R:\mathit{ProcRole}(\sigma,\mathcal{E},\mathcal{D})$, by Lemma 5(2) it follows $\vdash\mathtt{read}_p(\chi).P^{\neg\rho}:\mathit{ProcRole}(\sigma,\mathcal{E},\mathcal{D})$ and $\vdash R:\mathit{ProcRole}(\sigma,\mathcal{E},\mathcal{D})$ and therefore by Lemma 5(1) we have $\vdash\mathtt{read}_p(\chi).P:\mathit{Proc}(\sigma,\mathcal{E},\mathcal{D},\rho)$. Lemma 4(8) implies $\mathit{\Gamma}_\chi\vdash P:\mathit{Proc}(\sigma,\mathcal{E},\mathcal{D},\rho)$. Lemma 7(2) gives $\vdash P\mathtt{s}_i:\mathit{Proc}(\sigma,\mathcal{E},\mathcal{D},\rho)$, which implies by rule (Role) $\vdash P{\mathtt{s}_i}^{\neg\rho}:\mathit{ProcRole}(\sigma,\mathcal{E},\mathcal{D})$ for $1\leq i\leq n$. We conclude as in the previous case using rules (ParR) and (NetLoc).

- Rule ($\mathtt{change}$):

$$l[\![\,T\parallel\mathtt{change}_p(\chi,V).P^{\neg\rho}\mid R\,]\!]\quad\rightarrow\quad l[\![\,T'\parallel P^{\neg\rho}\mid R\,]\!]$$

where $\mathtt{ch}(p,\chi,V,T)=T'$. By definition of $\mathtt{ch}$ we have that T' is obtained from T by replacing some data terms U_i (identified by the path p and such that $\mathtt{match}(\chi,U_i)$) with the data terms $V\{\!\{U_i/|\chi|\}\!\}$ for $1\leq i\leq n$. From $\vdash\mathbf{N}:\mathit{Net}$, by Lemma 6(1) we get $\mathcal{T}(l)=(\sigma,\mathcal{E},\mathcal{D})$, $\vdash T:\mathit{Tree}(\sigma,\mathcal{E},\mathcal{D},\tau,\zeta)$ and $\vdash\mathtt{change}_p(\chi,V).P^{\neg\rho}\mid R:\mathit{ProcRole}(\sigma,\mathcal{E},\mathcal{D})$.

By Lemma 5(2) we have that $\vdash\mathtt{change}_p(\chi,V).P^{\neg\rho}:\mathit{ProcRole}(\sigma,\mathcal{E},\mathcal{D})$ and $\vdash R:\mathit{ProcRole}(\sigma,\mathcal{E},\mathcal{D})$. Then Lemma 5(1) implies $\vdash\mathtt{change}_p(\chi,V).P:\mathit{Proc}(\sigma,\mathcal{E},\mathcal{D},\rho)$. Lemma 4(9) gives $P:\mathit{Proc}(\sigma,\mathcal{E},\mathcal{D},\rho)$ and we obtain $\vdash P^{\neg\rho}:\mathit{ProcRole}(\sigma,\mathcal{E},\mathcal{D})$ by rule (Role). Lemma 4(9) gives also $\vdash p:\mathit{Path}(\alpha)$ and ($\mathit{\Gamma}_\chi\vdash V:\mathit{Script}(\sigma,\mathcal{E},\mathcal{D})$ or $\mathit{\Gamma}_\chi\vdash V:\mathit{Pointer}(\beta)$ or ($\mathit{\Gamma}_\chi\vdash V:\mathit{Tree}(\sigma,\mathcal{E},\mathcal{D},\tau_0,\zeta_0)$ and $\alpha\leqq\tau_0$)). Lemma 7(2) gives $\vdash V\{\!\{U_i/|\chi|\}\!\}:\mathit{Script}(\sigma,\mathcal{E},\mathcal{D})$ or $\vdash V\{\!\{U_i/|\chi|\}\!\}:\mathit{Pointer}(\beta)$ or $\vdash V\{\!\{U_i/|\chi|\}\!\}:\mathit{Tree}(\sigma,\mathcal{E},\mathcal{D},\tau_0,\zeta_0)$. In all cases we have $\vdash T':\mathit{Tree}(\sigma,\mathcal{E},\mathcal{D},\tau,\zeta')$ for some ζ' by Lemma 8(2) and (3), then by rules (ParR) and (NetLoc) we conclude $l[\![\,T_1\parallel P^{\neg\rho}\mid R\,]\!]:\mathit{Net}$.

- Rule ($\mathtt{enable}$):

$$l[\![\,T\parallel\mathtt{enable}_p(r).P^{\neg\rho}\mid R\,]\!]\quad\rightarrow\quad l[\![\,T'\parallel P^{\neg\rho}\mid R\,]\!]$$

where $\mathtt{en}(p,r,T)=T'$. By definition of $\mathtt{en}$ we have that T' is obtained from T by replacing the trees T_i, identified by the path p, with the trees $(T_i)^{+r}$ for $1\leq i\leq n$. From $\vdash\mathbf{N}:\mathit{Net}$, by Lemma 6(1) we get $\mathcal{T}(l)=(\sigma,\mathcal{E},\mathcal{D})$, $\vdash T:\mathit{Tree}(\sigma,\mathcal{E},\mathcal{D},\rho,\zeta)$ and $\vdash\mathtt{enable}_p(r).P^{\neg\rho}\mid R:\mathit{ProcRole}(\sigma,\mathcal{E},\mathcal{D})$. Then by Lemma 5(2) we obtain $\vdash\mathtt{enable}_p(r).P^{\neg\rho}:\mathit{ProcRole}(\sigma,\mathcal{E},\mathcal{D})$ and $\vdash R:\mathit{ProcRole}(\sigma,\mathcal{E},\mathcal{D})$. Lemma 5(1) implies $\vdash\mathtt{enable}_p(r).P:\mathit{Proc}(\sigma,\mathcal{E},\mathcal{D},\rho)$. From $\vdash\mathtt{enable}_p(r).P:\mathit{Proc}(\sigma,\mathcal{E},\mathcal{D},\rho)$, by Lemma 4(10) we have $\vdash P:\mathit{Proc}(\sigma,\mathcal{E},\mathcal{D},\rho)$ and $\vdash p:\mathit{Path}(\alpha)$ and $\alpha\leqq\{r\}$. From $\vdash P:\mathit{Proc}(\sigma,\mathcal{E},\mathcal{D},\rho)$ by rule (Role) we derive $\vdash P^{\neg\rho}:\mathit{ProcRole}(\sigma,\mathcal{E},\mathcal{D})$. Lemma 8(4) implies $\vdash T':\mathit{Tree}(\sigma,\mathcal{E},\mathcal{D},\tau,\zeta')$ for some ζ'. Hence by rules (ParR) and (NetLoc) we get the proof.

B Proof of Proposition 2

Proposition 2 *If* $\mathbf{N}$ *is a well typed network and* $\mathbf{N}\rightarrowtriangle\boldsymbol{\nu}(l[\![\;T\parallel P^{\neg\rho}\mid R\;]\!]\;\;|\!|\!|\;\;\mathbf{N}')$, *then:*

P0 $\vdash T : \mathit{Tree}(\sigma, \mathcal{E}, \mathcal{D}, \tau, \zeta)$ *for some* τ, ζ *and* $\vdash P^{\neg\rho} \mid R : \mathit{ProcRole}(\sigma, \mathcal{E}, \mathcal{D})$*;*

P1 $P \equiv \bar{c}^{Tv}\langle v\rangle$ *implies* $\vdash v : Tv$ *and* $\mathcal{C}(Tv) \leq \rho$;

P2 $P \equiv \mathtt{go}\ l'.R$ *implies* $\vdash R : \mathit{ProcRole}(\sigma', \mathcal{E}', \mathcal{D}')$ *and* $\mathcal{T}(l') = (\sigma', \mathcal{E}', \mathcal{D}')$*;*

P3 $P \equiv \mathtt{run}_p$ *or* $P \equiv \mathtt{read}_p(x).P'$ *or* $P \equiv \mathtt{change}_p(\chi, V).P'$ *or* $P \equiv \mathtt{enable}_p(r).P'$ *or* $P \equiv \mathtt{disable}_p(r).P'$ *and* $\vdash p : \mathit{Path}(\alpha)$ *imply* $\alpha \leq \rho$;

P4 $P \equiv \mathtt{run}_p$ *or* $P \equiv \mathtt{read}_p(x).P'$ *or* $P \equiv \mathtt{change}_p(\chi, V).P'$ *or* $P \equiv \mathtt{enable}_p(r).P'$ *or* $P \equiv \mathtt{disable}_p(r).P'$ *and* p *identifies a data term* V *in the tree* T *imply that* V *is connected to his father by an edge whose set of roles* τ *is such that* $\tau \leq \rho$*;*

P5 $P \equiv \mathtt{run}_p$ *and* $\mathtt{rn}(p, T) = \{R_1, \ldots, R_n\}$ *imply* $\vdash R_i : \mathit{ProcRole}(\sigma, \mathcal{E}, \mathcal{D})$ *for* $1 \leq i \leq n$*;*

P6 $P \equiv \mathtt{read}_p(\chi).P'$ *and* $\mathtt{sub}(p, \chi, T) = \{\mathtt{s}_1, \ldots, \mathtt{s}_n\}$ *imply* $\vdash P\mathtt{s}_i : \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)$ *for* $1 \leq i \leq n$*;*

P7 $P \equiv \mathtt{change}_p(x^{(\sigma', \mathcal{E}', \mathcal{D}', \tau, \zeta)}, V).P'$ *implies* $\zeta \leq \rho$*;*

P8 $P \equiv \mathtt{change}_p(\chi, V).P'$ *and* $\mathtt{ch}(p, \chi, V, T) = T'$ *imply* $\vdash T' : \mathit{Tree}(\sigma, \mathcal{E}, \mathcal{D}, \tau, \zeta')$ *for some* ζ'*;*

P9 $P \equiv \mathtt{enable}_p(r).P'$ *implies* $(\rho, r) \in^{+} \mathcal{E}$;

P10 $P \equiv \mathtt{enable}_p(r).P'$ *and* $\mathtt{en}(p, r, T) = T'$ *imply* $\vdash T' : \mathit{Tree}(\sigma, \mathcal{E}, \mathcal{D}, \tau, \zeta')$ *for some* ζ'*;*

P11 $P \equiv \mathtt{disable}_p(r).P'$ *implies* $(\rho, r) \in^{-} \mathcal{D}$;

P12 $P \equiv \mathtt{disable}_p(r).P'$ *and* $\mathtt{di}(p, r, T) = T'$ *imply* $\vdash T' : \mathit{Tree}(\sigma, \mathcal{E}, \mathcal{D}, \tau, \zeta')$ *for some* ζ'*;*

where $\mathcal{T}(l) = (\sigma, \mathcal{E}, \mathcal{D})$.

Proof.

- Point **P0** follows from Theorem 1 and Lemma 6(3), (4), (1).
- For Points **P1**, **P2**, **P3**, **P7**, **P9** and **P11** notice that from **P0** we get $\vdash P^{\neg\rho} \mid R : \mathit{ProcRole}(\sigma, \mathcal{E}, \mathcal{D})$, which implies $\vdash P^{\neg\rho} : \mathit{ProcRole}(\sigma, \mathcal{E}, \mathcal{D})$ by Lemma 5(2) and $\vdash P : \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)$ by Lemma 5(1). Therefore **P1**, **P2**, **P3**, **P7**, **P9** and **P11** follow from Points (3), (6), (7)-(11), (9), (10), (11) of Lemma 4, respectively.
- Point **P4** easily follows from **P3** by definition of compatibility between paths and tree paths.
- Point **P5** is true by Lemma 8(1), since $\vdash T : \mathit{Tree}(\sigma, \mathcal{E}, \mathcal{D}, \tau, \zeta)$ and by definition the function rn returns processes with roles whose scripts are stored in the local tree.
- In case **P6**

  $$\boldsymbol{\nu}(l[\![\ T \parallel P^{\neg\rho} \mid R\ ]\!]\ \mathbin{|||}\ \mathbf{N}') \rightarrow \boldsymbol{\nu}(l[\![\ T \parallel P\mathtt{s}_1{}^{\neg\rho} \mid \ldots \mid P\mathtt{s}_n{}^{\neg\rho} \mid R\ ]\!]\ \mathbin{|||}\ \mathbf{N}')$$

  by rule (read), and this last process is typeable by Theorem 1. Therefore we conclude by Lemmas 6(3), (4), (1), and 5(2), (1).
- In **P8** we get $\vdash T : \mathit{Tree}(\sigma, \mathcal{E}, \mathcal{D}, \tau, \zeta)$ and $\vdash \mathtt{change}_p(\chi, V).P' : \mathit{Proc}(\sigma, \mathcal{E}, \mathcal{D}, \rho)$. Lemma 4(9) gives $\vdash p : \mathit{Path}(\alpha)$ and $(\mathit{\Gamma}_\chi \vdash V : \mathit{Script}(\sigma, \mathcal{E}, \mathcal{D})$ or $\mathit{\Gamma}_\chi \vdash V : \mathit{Pointer}(\beta)$ or $(\mathit{\Gamma}_\chi \vdash V : \mathit{Tree}(\sigma, \mathcal{E}, \mathcal{D}, \tau_0, \zeta_0)$ and $\alpha \leqq \tau_0))$. Lemma 7(2) gives $\vdash V\{\!\{U_i/|\chi|\}\!\} : \mathit{Script}(\sigma, \mathcal{E}, \mathcal{D})$ or $\vdash V\{\!\{U_i/|\chi|\}\!\} : \mathit{Pointer}(\beta)$ or $\vdash V\{\!\{U_i/|\chi|\}\!\} : \mathit{Tree}(\sigma, \mathcal{E}, \mathcal{D}, \tau_0, \zeta_0)$. In all cases we get $\vdash T' : \mathit{Tree}(\sigma, \mathcal{E}, \mathcal{D}, \tau, \zeta')$ for some ζ' by definition of the function ch and by Lemma 8(2) and (3).

- In **P10** we get $\vdash T : Tree(\sigma, \mathcal{E}, \mathcal{D}, \tau, \zeta)$ and $\vdash \mathtt{enable}_p(r).P' : Proc(\sigma, \mathcal{E}, \mathcal{D}, \rho)$. By Lemma 4(10) we have $\vdash p : Path(\alpha)$ and $\alpha \leqq \{r\}$. Lemma 8(4) and the definition of the function en imply $\vdash T' : Tree(\sigma, \mathcal{E}, \mathcal{D}, \tau, \zeta')$ for some ζ'.
- In **P12** we get $\vdash T : Tree(\sigma, \mathcal{E}, \mathcal{D}, \tau, \zeta)$. Lemma 8(5) and the definition of the function di imply $\vdash T' : Tree(\sigma, \mathcal{E}, \mathcal{D}, \tau, \zeta')$ for some ζ'.