Security Types for Dynamic Web Data [1]

Mariangiola Dezani-Ciancaglini [a] Silvia Ghilezan [b]
Jovanka Pantović [b] Daniele Varacca [c]

[a] *Dipartimento di Informatica, Università di Torino, Italy* dezani@di.unito.it

[b] *Faculty of Engineering, University of Novi Sad, Serbia* {gsilvia, pantovic}@uns.ns.ac.yu

[c] *PPS, University of Paris 7 & CNRS, France* varacca@pps.jussieu.fr

Abstract

We describe a type system for the X$d\pi$ calculus of Gardner and Maffeis. An X$d\pi$-network is a network of locations, where each location consists of both a data tree (which contains scripts and pointers to nodes in trees at different locations) and a process, for modelling process interaction, process migration and interaction between processes and data. Our type system is based on types for locations, data and processes, expressing security levels. A tree can store data of different security level, independently from the security level of the enclosing location. The access and mobility rights of a process depend on the security level of the "source" location of the process itself, i.e. of the location where the process was in the initial network or where the process was created by the activation of a script. The type system enjoys type preservation under reduction (subject reduction). In consequence of subject reduction we prove the following security properties. In a well-typed X$d\pi$-network, a process P whose source location is of level h can copy data of security level at most h and update data of security level less than h. Moreover, the process P can only communicate data and go to locations of security level equal or less than h.

1 Introduction

Information systems have evolved into open distributed systems that include decentralised peer-to-peer networks. An essential role of such systems is management of data, which appear to be semi-structured and distributed. Data-sharing applications require the integration of mobile processes and semi-structured data.

As information networks become more open and dynamic, the need for security and privacy grows stronger. Systems must be able to exchange data and processes while preserving security. One solution is to ground them on typed models. In such

[1] This work was partly funded by the project MMIT 1438 of PSNTR, by FP6-2004-510996 Coordination Action TYPES, by the project GLORA 144029 of MSEP and by the ANR project "ParSec" ANR-06-SETI-010-02.

models, a well-typed network must reduce only to well-typed networks, assuring access and movement rights.

In this paper we propose a type system for the X$d\pi$ calculus [9]. An X$d\pi$-network is a network of locations, where each location consists of both a data tree and a running process, for modelling process interaction, process migration and interaction between processes and data. The leaves of data trees contain pointers to nodes in trees of different locations, and scripts, i.e. static processes, which can be activated. In turn, scripts, pointers and trees can occur inside scripts and running processes.

In addition to the original syntax, we decorate location names with security levels taken from a partially ordered set of security levels with a bottom element. Therefore a location in a well-formed network will be of the shape:

$$l^h[\, T \parallel P \,]$$

where l is a location name, h is its security level, T is a tree of data and P is a running process. Pointers, scripts and running processes are assigned security levels by means of a typing system.

The access and mobility rights of a process depend on the security level of the "source" location of the process itself, i.e. of the location where the process was in the initial network or where the process was created by the activation of a script. Hence in a well-typed network each process has the security level of its source location. Security levels of scripts and pointers in trees, however, don't depend on the level of the enclosing location.

Processes migrate thanks to the go command. The go command can only move a process from one location to a location of security level lower than or equal to the level of the process itself. Processes can also communicate data via channels. The security levels of the communicated data will never exceed the security level of the process.

Running processes can activate scripts in the local tree by the command $\mathtt{run}_p$, where p is a path expression which identifies a set of nodes. In a well-typed network a scripted process can be activated only if its security level is at most the one of the enclosing location.

Running processes can also modify the local tree and use the information in that tree by means of the command `update`. All trees can be copied by all processes, but only trees containing no data can be deleted and possibly replaced. A process of security level h can only read data of security level at most h, and modify data of security level less than h. The only exception being that processes generated by activating scripts can modify scripts of the same security level and identified by the same path in trees.

An important feature of the X$d\pi$ calculus is the fact that a script can dynamically create links to the location where it is activated. On the one hand this is a desirable feature of mobile processes, for instance to allow a process to return to the activating location. On the other hand it increases the complexity of the typing system. Indeed we need to distinguish processes that contain unresolved references to their activating location (these processes only make sense as "dormant" scripts) from processes where all pointers are absolute, that are the only processes that are allowed to be actively running.

Related Work The X$d\pi$ calculus [9,14] models both localised, mobile processes and distributed, dynamic, semi-structured data, allowing to represent data-sharing applications. It can be seen as an extension of the Active XML model [1].

The locations and the processes of X$d\pi$ are essentially those of $d\pi$ [10] enriched with capabilities for data manipulation. The only difference is that a process in $d\pi$ can migrate to a location independently from the existence of the location itself in the current network, while in X$d\pi$ such an existence is a necessary condition for migration. The data trees of X$d\pi$ are related to those in [2,4] and the treatment of shared distributed data is inspired by [19]. We refer to [9] for further references related to the calculus design.

Many type systems controlling the use of resources and the mobility of processes have been proposed for the $d\pi$ calculus [10] and for related calculi [16,6,5]. The types discussed here are essentially inspired by the security types checking access rights for π-calculus of [11]. For simplicity we do not distinguish between reading and mobility rights, but our type system can be extended to take them into account. For the purpose of the present paper it is enough to consider elements of a partially ordered set with a bottom element as security levels instead of elements of a lattice as usual [20], this choice being justified by the fact that we do not use meets and joins. We formalise the network properties assured by our type system using the notions of network invariant and initial network as in [3].

The present paper is an expanded and revised version of [8], the main differences being:

- the data in trees can be of different security levels and do not depend on the security level of the enclosing location, while in [8] each location was only allowed to contain data of at most the security level of the location itself;
- the communication, copying, updating and mobility rights of processes only depend on their source location, while in [8] they were depending on the enclosing location, but for the possibility of processes to move back to their source locations.

Minor differences are:

- the capability of modifying data requires a higher security level than the capability of reading data, while in [8] there was no difference;
- in order to take into account the new fine-grained security labelling of data trees there is a special type for *data-less* trees, i.e. trees whose leaves contain no data, because only data-less trees can be deleted or replaced (they can be considered garbage).

Outline of the paper Section 2 and Section 3 introduce the syntax, the reduction rules, and the typing rules of typed X$d\pi$, exemplified by the examples in Section 5. The properties of the calculus are discussed in Section 4 and proved in Appendices B and C. Section 6 contains a few final remarks.

2 Syntax and Operational Semantics

The X$d\pi$ calculus we consider here is essentially the calculus introduced in [9], with a few important differences.

The main difference between the original X$d\pi$ and the present one is the use of a typed syntax. We decorate the location names with security levels and the channel names with value types. (An alternative approach could avoid these decorations by fixing an environment for locations and channels.)

More importantly, the syntax includes a typed matching function instead of an untyped one. Pattern matching needs to take types into account, in order to have type preservation under reduction. We will explain and motivate this choice at the end of the section.

In order to simplify the syntax we only allow monadic instead of polyadic communication and we do not distinguish between public channels (which cannot be restricted) and session channels (which must be restricted in the scripts). [2] These features of the original X$d\pi$ can be easily handled by our type system.

2.1 Syntax

Networks A network is a parallel composition ($|$) of locations consisting of a tree and a process, where processes at different locations can share communication channels. In a well-formed network the locations have different names. The syntax of networks is given in Table 1. We use l, m to range over location names, and h, i, j over security levels. The location $l^h[\ T \parallel P\ ]$ is well-formed if both the tree T and the process P do not contain occurrences of free variables. We use c to range over channel names and Tv to denote a value type as defined in Table 9. The binder ν is, as usual, the restriction operator.

Trees The data model is an unordered edge-labelled rooted tree with leaves containing empty trees, scripts and pointers. The syntax of trees is presented in Table 2, using $\mathtt{a}$ to denote an edge label.

A *script* is a static process embedded in a tree that can be activated by a process from the same location. We use Π to range over processes and variables, and a script is denoted by $\Box\Pi$.

A *path* identifies nodes in a tree. Table 3 gives the formation rules of paths, using p to range over paths. In a path, "$\mathtt{a}$" denotes a step along an edge $\mathtt{a}$, " // " denotes any node, ".." a step back, "$\bullet$" the path from the root to the current node, "x" a variable and "/ " the path composition. We will say that a path is a *local path* if it contains

[2] The distinction between public and session channels is important for implementation since otherwise one needs to alpha-convert the whole data tree of a location when a process, restricting a channel name, migrates.

$$\mathbf{N} ::= \mathbf{0} \;\boldsymbol{|}\; \mathbf{N} \mid \mathbf{N} \;\boldsymbol{|}\; l^h[T \parallel P] \;\boldsymbol{|}\; (\nu c^{Tv})\mathbf{N}$$

Table 1
Syntax of networks

$T ::=$	$\emptyset$	empty rooted tree
$\boldsymbol{|}$	x	tree variable
$\boldsymbol{|}$	$T \mid T$	composition of trees, joining the roots
$\boldsymbol{|}$	$\mathtt{a}[T]$	edge labeled a with subtree T
$\boldsymbol{|}$	$\mathtt{a}[\Box\Pi]$	edge labeled a with script $\Box\Pi$
$\boldsymbol{|}$	$\mathtt{a}[p@\lambda]$	edge labeled a with pointer $p@\lambda$

Table 2
Syntax of trees

$$p ::= \mathtt{a} \;\boldsymbol{|}\; // \;\boldsymbol{|}\; .. \;\boldsymbol{|}\; \bullet \;\boldsymbol{|}\; x \;\boldsymbol{|}\; p \,/\, p$$

Table 3
Syntax of paths

"•".[3] Local paths are only allowed to be present inside scripts. The string "•" is replaced by the actual path of the location of the script at the moment in which the script is activated.

We use λ to range over variables and location names super-scripted by security levels. A *pointer* $p@\lambda$ refers to the set of nodes identified by the path p in the tree at location λ.

Processes The processes that we are concerned with are essentially $d\pi$-calculus processes [10], where the local communication modelled by π-calculus processes [15,21] is extended with migration between locations (command `go`). There are two more commands for local communication between processes and data: one for updating (copy, paste, cut, etc.) the data tree (`update`) and the other one that activates the execution of scripts that are embedded in local data tree (`run`). We use P, Q, R to range over processes, and γ to range over channel names (decorated by value types) and variables. The syntax of processes is given in Table 4.

[3] The path syntax allows also meaningless paths, like "• / • / •": this could be clearly avoided either by typing or by refining the syntax.

$P ::= 0$		the nil process
❙	$P \mid P$	composition of processes
❙	$(\nu c^{Tv})P$	declare new channel name c transmitting values of type Tv
❙	$\bar{\gamma}\langle v \rangle$	output value v on a channel γ
❙	$\gamma(x).P$	input parametrised by a variable x
❙	$!\gamma(x).P$	replication of an input process
❙	$\texttt{go}\ \lambda.P$	migrate to location λ, continue as P
❙	$\texttt{go}\ \circlearrowleft.P$	migrate to the source location, continue as P
❙	$\texttt{run}_p$	run the processes identified by the path expression p
❙	$\texttt{update}_p(\chi, V).P$	update command

Table 4
Syntax of processes

$$v ::= c^{Tv} \mid \Box P \mid l^h \mid p \mid T$$

Table 5
Syntax of values

$$\chi ::= \Box x^j \mid y^\star @ x^j \mid x^{DL} \mid x$$

$$V ::= \Box \Pi \mid p@\lambda \mid T$$

Table 6
Syntax of patterns and data terms

A *value* is either a channel name super-scripted with a value type, a script, a location name super-scripted with a security level, a path or a tree. Using v to range over values, the syntax of values is given in Table 5.

The argument of go is a location name (super-scripted with a security level) or a variable, or the symbol "$\circlearrowleft$", which can only occur in scripts to denote the location where the script will be activated.

The two arguments of the update command are respectively a *pattern* χ and a *data term* V, whose syntax is given in Table 6. A pattern is either a script pattern, or a pointer pattern, or a *data-less* tree variable, or a tree variable. In a pointer pattern j

is a security level and $\star \in \{Local, \epsilon\}$ [4] indicates whether y stands for a local path or for a path without occurrences of "$\bullet$". Data terms can be scripts, pointers or trees. The need to distinguish generic trees from data-less trees (that is trees whose leaves are empty) arises from the facts that trees themselves do not have security levels. In order for a process to delete or replace a tree, it has to have permission to delete all the data contained in the tree first. Then any process, regardless of its security level, can delete or replace data-less trees.

In $\texttt{update}_p(\chi, V).P$ the variables of χ can occur both in V and in P and they are bound. For this reason we allow variable occurrences in scripts, pointers, trees and processes.

2.2 Reduction rules

The reduction relation describes three forms of interactions:

- processes can communicate with each other within a location (rules (com) and (com!));
- processes can move between locations (rules (stay) and (go));
- process can interact with the local data (rules (update) and (run)).

The reduction relation is the least relation on networks which is closed with respect to structural equivalence, reduction rules given in Table 7 and reduction contexts, given by

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

The standard definition of structural equivalence is presented in Appendix A.

The communication rules (com) and (com!) are from the π-calculus [15,21]. 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.

In rules (run) and (update) we denote by $p(T)$ the tree T where all the nodes identified by the path p are underlined. When the node is a leaf we underline its label.

The command $\texttt{run}_p$ finds all the scripts in the local tree identified by the path p, by means of the update function $\rightsquigarrow$. Then it activates their parallel execution, after replacing "$\circlearrowleft$" and "$\bullet$" by the enclosing location and the path p, respectively.

The update command $\texttt{update}_p(\chi, V).P$ traversing top-down the local tree finds all the data terms V_k given by the path p and pattern matches these data terms with χ to obtain substitutions $\mathtt{s}_k$ when they exist. For each successful pattern matching it replaces the V_k with $V\mathtt{s}_k$ and starts $P\mathtt{s}_k$ in parallel. The $\texttt{match}$ function, in order to check if a data term agrees with a pattern, requires not only the data term to be, respectively, a script, a pointer, a data-less tree or a tree, according to the four

[4] Here and in the following we use ϵ to denote the empty string, so we get either $y^{Local}@x^j$ or $y@x^j$.

$$\text{(com)} \quad l^h[\, T \,||\, \bar{c}^{Tv}\langle v\rangle \mid c^{Tv}(z).P \mid Q \,] \rightarrow l^h[\, T \,||\, P\{v/z\} \mid Q \,]$$

$$\text{(com!)} \quad l^h[\, T \,||\, \bar{c}^{Tv}\langle v\rangle \mid !c^{Tv}(z).P \mid Q \,] \rightarrow l^h[\, T \,||\, !c^{Tv}(z).P \mid P\{v/z\} \mid Q \,]$$

$$\text{(stay)} \quad l^h[\, T \,||\, \mathtt{go}\ l^h.P \mid Q \,] \rightarrow l^h[\, T \,||\, P \mid Q \,]$$

$$\text{(go)} \quad l^h[\, T_1 \,||\, \mathtt{go}\ m^j.P \mid Q \,] \mid m^j[\, T_2 \,||\, R \,] \rightarrow l^h[\, T_1 \,||\, Q \,] \mid m^j[\, T_2 \,||\, P \mid R \,]$$

$$\text{(run)} \quad \frac{p(T) \rightsquigarrow_{p,l^h,\Box x^h,\Box x} T, \{\{\Box P_1/\Box x\}, \ldots, \{\Box P_n/\Box x\}\}}{l^h[\, T \,||\, \mathtt{run}_p \mid Q \,] \rightarrow l^h[\, T \,||\, P_1 \mid \ldots \mid P_n \mid Q \,]}$$

$$\text{(update)} \quad \frac{p(T) \rightsquigarrow_{p,l^h,\chi,V} T', \{\mathtt{s}_1, \ldots, \mathtt{s}_n\}}{l^h[\, T \parallel \mathtt{update}_p(\chi, V).P \mid Q \,] \rightarrow l^h[\, T' \parallel P\mathtt{s}_1 \mid \ldots \mid P\mathtt{s}_n \mid Q \,]}$$

Table 7
Reduction rules

$$\text{(Empty tree)} \quad \emptyset \rightsquigarrow_\theta \emptyset, \emptyset$$

$$\text{(Script)} \quad \Box P \rightsquigarrow_\theta \Box P, \emptyset$$

$$\text{(Pointer)} \quad p@l^h \rightsquigarrow_\theta p@l^h, \emptyset$$

$$\text{(Node)} \quad \frac{U \rightsquigarrow_\theta V, \Theta}{\mathtt{a}[U] \rightsquigarrow_\theta \mathtt{a}[V], \Theta}$$

$$\text{(Par)} \quad \frac{U_1 \rightsquigarrow_\theta T_1, \Theta_1 \qquad U_2 \rightsquigarrow_\theta T_2, \Theta_2}{U_1|U_2 \rightsquigarrow_\theta T_1|T_2, \Theta_1 \cup \Theta_2}$$

$$\text{(Id)} \quad \frac{\mathtt{match}(U, \chi) \text{ undefined} \qquad U \rightsquigarrow_\theta V, \Theta}{\mathtt{a}[\underline{U}] \rightsquigarrow_\theta \mathtt{a}[V], \Theta}$$

$$\text{(Up)} \quad \frac{\mathtt{match}(U, \chi) = \mathtt{s} \qquad V\mathtt{s} \rightsquigarrow_\theta V', \Theta \qquad \theta = p, l^h, \chi, V}{\mathtt{a}[\underline{U}] \rightsquigarrow_\theta \mathtt{a}[V'], \{\mathtt{s}\{l^h/\circlearrowleft, p/\bullet\}\} \cup \Theta}$$

Table 8
Definition of the update function $\rightsquigarrow$

shapes of the pattern (as in [9]), but it requires also the data terms to satisfy the type information given by the pattern. This means that:

(1) if the pattern is $\Box x^j$, then the data term must be a script of level j,
(2) if the pattern is $y^\star @ x^j$, then the data term must be a pointer in which *(i)* the path can be a local path only if $\star = \mathit{Local}$ and *(ii)* the location must be of level j,
(3) if the pattern is x^{DL}, then the data term must be a data-less tree,
(4) if the pattern is x, then the data term must be a tree.

These conditions are enforced by using the type assignment system of Section 3. If the typed match is successful, the function returns a substitution which replaces the variables in the pattern by the corresponding data terms. More precisely the definition of the `match` function is:

(1) $\mathtt{match}(\Box P, \Box x^j) = \{\Box P/\Box x\}$ if $\vdash P : \mathit{ProcLocal}(j)$;
(2) $\mathtt{match}(p@l^j, y^\star @x^j) = \{l^j/x, p/y\}$ if $\vdash p : \mathit{Path}^\star$;
(3) $\mathtt{match}(T, x^{DL}) = \{T/x\}$ if $\vdash T : DLTree$;
(4) $\mathtt{match}(T, x) = \{T/x\}$ if $\vdash T : Tree$.

In principle it would be desirable to avoid security level matching at run time, and rely on static typing only. However in this setting, static typing would be too restrictive. Data terms in a tree can have any security level, and we cannot statically know the security levels of data terms found using the path "//". This is why dynamic checking is necessary.

The reduction rules for `update` and `run` are based on the definition of the update function $\rightsquigarrow$ which is parametrised on p, l^h, χ, V. The argument of this function can be either a tree (with some underlined nodes), or a tree label (i.e. a script or a pointer) possibly underlined: we use U to range over these data terms enriched with underlining. The result is a data term and a set of substitutions.

Table 8 defines the function $\rightsquigarrow$. The only interesting rules are (Id) and (Up): we convene that the match function ignores underlining. If U and χ do not match, then the function $\rightsquigarrow$ is applied to U, since it could contain other underlined nodes. When the matching between U and χ gives a substitution $\mathtt{s}$, the data term $V\mathtt{s}$ is updated (notice that the range of $\mathtt{s}$ can contain underlined nodes and then $V\mathtt{s}$ can contain underlined nodes too) obtaining V' and the set of substitutions Θ. Finally U is replaced by with V' and the substitution $\mathtt{s}\{l^h/\circlearrowleft, p/\bullet\}$ is added to Θ. This addition is useful when $\mathtt{s} = \{\Box P/\Box x\}$ for solving the references to the enclosing location and to the current path.

The definition of substitution we use is standard, with the exception that occurrences of "$\circlearrowleft$" and "$\bullet$" inside scripts in the processes prefixed by the update command are not affected by this substitution. Similarly if $\mathtt{s} = \{T/x\}$, occurrences of "$\circlearrowleft$" and "$\bullet$" inside scripts in the leaves of T are not affected by this substitution. Concretely, $\{T/x\}\{l^h/\circlearrowleft, p/\bullet\} = \{T/x\}$ for any T, x, l^h, p.

We denote by $\hat{\chi}$ the data term obtained from the pattern χ by erasing the type information:

$$\hat{\chi} = \begin{cases} \Box x \text{ if } \chi = \Box x^j, \\ y@x \text{ if } \chi = y^\star @x^j, \\ x \text{ if } \chi = x^{DL} \text{ or } \chi = x. \end{cases}$$

Some special forms of the update command have been already defined in [9]:

$$\texttt{cut}_p(\chi).Q := \texttt{update}_p(\chi, \emptyset).Q$$
$$\texttt{copy}_p(\chi).Q := \texttt{update}_p(\chi, \hat{\chi}).Q$$
$$\texttt{paste}_p\langle T\rangle.Q := \texttt{update}_p(x, x|T).Q \qquad \text{where } x \text{ does not occur in } T, Q.$$

We will freely use these shorthands in the examples.

In Subsection 5.1 we will show examples of process reductions.

3 Type Assignment

The main goals of our type system are to control communication of values, access to data and migration of processes between locations. We will formally define these notions in Section 4.

We rely on a notion of security levels, and therefore we assume a fixed partial order $(\mathcal{L}, \leq)$ of security levels with a bottom $\perp$. As already said in Section 2 we use h, i, j to range over elements of $\mathcal{L}$.

The syntax of types is the content of Table 9. Clearly the types correspond to the syntactic categories of the previous section. We use the suffix $Local$ when we allow local paths. As we said, we need this distinction since a run or an update command containing a local path as index cannot be executed, and thus it can only appear inside a "dormant" script.

We will use $Path^\star$ as short for $Path$ or $PathLocal$ and similarly for the other types. When more than one $\star$ appears in a typing rule we always assume that all of them are replaced either by ϵ or by $Local$.

We define the *security level* of a value type (notation $|Tv|$) as follows:

- $|Ch(Tv)| = |Tv|$;
- $|Loc(i)| = |Script(i)| = i$;
- $|Path^\star| = |DLTree| = |Tree^\star| = \perp$.

An *environment* Σ gives the association between:

- variables and value types
- variables and local process types

i.e. we define:

$$\Sigma := \emptyset \mid \Sigma, x : Tv \mid \Sigma, x : ProcLocal(i).$$

We use the environment by means of a standard axiom:

$$\frac{}{\Sigma, x : \sigma \vdash x : \sigma}\,(axiom)$$

where σ ranges over value types and local process types.

Typing rules for channels, locations and scripts are as expected (recall that Π

ranges over processes and variables):

$$\frac{}{\Sigma \vdash c^{Tv} : Ch(Tv)}\,(chan) \qquad \frac{}{\Sigma \vdash l^i : Loc(i)}\,(loc)$$

$$\frac{\Sigma \vdash \Pi : Proc^\star(i)}{\Sigma \vdash \Box\Pi : Script(i)}\,(script)$$

Typing rules for paths are given in Table 10: a local path always gets the type $PathLocal$ instead of $Path$.

The typing rule for pointers

$$\frac{\Sigma \vdash \lambda : Loc(i) \qquad \Sigma \vdash p : Path^\star}{\Sigma \vdash p@\lambda : Pointer^\star(i)}\,(pointer)$$

gives a $Pointer$ or a $PointerLocal$ type according to the path type. The security level of the pointer is the security level of the pointed location.

Typing rules for trees are given in Table 11. According to these typing rules:

$Ch(Tv)$	type of channels communicating values of type Tv
$Loc(i)$	type of locations at security level i
$Script(i)$	type of scripts at security level i
$Path$	type of paths, not containing "$\bullet$"
$PathLocal$	type of paths, possibly containing "$\bullet$"
$Pointer(i)$	type of pointers, not containing local paths, at security level i
$PointerLocal(i)$	type of pointers, possibly containing local paths, at security level i
$DLTree$	type of data-less trees
$Tree$	type of trees, not containing local paths
$TreeLocal$	type of trees, possibly containing local paths
$Proc(i)$	type of processes, not containing local paths, at security level i
$ProcLocal(i)$	type of processes, possibly containing local paths, at security level i
Net	type of networks

where $i \in \mathcal{L}$ and Tv ranges over value types defined by

$$Tv ::= Ch(Tv) \;\big|\; Loc(i) \;\big|\; Script(i) \;\big|\; Path^\star \;\big|\; DLTree \;\big|\; Tree^\star$$

Table 9
Syntax of types

$$\frac{}{\Sigma \vdash \mathtt{a} : \mathit{Path}}\ (path\mathtt{a}) \qquad \frac{}{\Sigma \vdash // : \mathit{Path}}\ (path//)$$

$$\frac{}{\Sigma \vdash .. : \mathit{Path}}\ (path..) \qquad \frac{}{\Sigma \vdash \bullet : \mathit{PathLocal}}\ (path\bullet)$$

$$\frac{\Sigma \vdash p : \mathit{Path}^\star \quad \Sigma \vdash p' : \mathit{Path}^\star}{\Sigma \vdash p \,/\, p' : \mathit{Path}^\star}\ (path/) \qquad \frac{\Sigma \vdash p : \mathit{Path}}{\Sigma \vdash p : \mathit{PathLocal}}\ (pathL)$$

Table 10
Typing of paths

$$\frac{}{\Sigma \vdash \emptyset : DLTree}\ (treeEmpty) \qquad \frac{\Sigma \vdash T : \mathit{Tree}^\star}{\Sigma \vdash \mathtt{a}[T] : \mathit{Tree}^\star}\ (tree\mathtt{a})$$

$$\frac{\Sigma \vdash T_1 : DLTree \quad \Sigma \vdash T_2 : DLTree}{\Sigma \vdash T_1 \mid T_2 : DLTree}\ (treeDL|)$$

$$\frac{\Sigma \vdash T : DLTree}{\Sigma \vdash \mathtt{a}[T] : DLTree}\ (treeDL\mathtt{a}) \qquad \frac{\Sigma \vdash \Box\Pi : \mathit{Script}(i)}{\Sigma \vdash \mathtt{a}[\Box\Pi] : Tree}\ (treeScript)$$

$$\frac{\Sigma \vdash T_1 : \mathit{Tree}^\star \quad \Sigma \vdash T_2 : \mathit{Tree}^\star}{\Sigma \vdash T_1 \mid T_2 : \mathit{Tree}^\star}\ (tree|) \qquad \frac{\Sigma \vdash p@\lambda : \mathit{Pointer}^\star(i)}{\Sigma \vdash \mathtt{a}[p@\lambda] : \mathit{Tree}^\star}\ (treePointer)$$

$$\frac{\Sigma \vdash T : DLTree}{\Sigma \vdash T : Tree}\ (treeDL) \qquad \frac{\Sigma \vdash T : Tree}{\Sigma \vdash T : \mathit{TreeLocal}}\ (treeL)$$

Table 11
Typing of trees

- a tree is data-less, i.e. it has the type $DLTree$, only if all its leaves are labelled by $\emptyset$;
- a tree that has at least one node labelled by a local pointer will be typed by $\mathit{TreeLocal}$.

Typing rules for processes are given in Table 12. The rule (go) allows a process whose source location is of security level i to migrate to a location at security level j only if $j \leq i$.

In the typing rules for update we assume that $\chi \in \{\Box x^j, y^\star @x^j, x^{DL}, x\}$, and we define the environment Σ_χ for associating types to the variables bound by the

$$\frac{}{\Sigma \vdash 0 : \mathit{Proc}^\star(i)}\ (\mathit{proc}0) \qquad \frac{\Sigma \vdash P_1 : \mathit{Proc}^\star(i) \quad \Sigma \vdash P_2 : \mathit{Proc}^\star(i)}{\Sigma \vdash P_1 \mid P_2 : \mathit{Proc}^\star(i)}\ (\mathit{proc}|)$$

$$\frac{\Sigma \vdash P : \mathit{Proc}^\star(i) \quad |Tv| \leq i}{\Sigma \vdash (\nu c^{Tv})P : \mathit{Proc}^\star(i)}\ (\mathit{proc}\nu)$$

$$\frac{\Sigma \vdash v : Tv \quad \Sigma \vdash \gamma : Ch(Tv) \quad |Tv| \leq i}{\Sigma \vdash \bar{\gamma}\langle v \rangle : \mathit{Proc}^\star(i)}\ (\mathit{out})$$

$$\frac{\Sigma, x : Tv \vdash P : \mathit{Proc}^\star(i) \quad \Sigma \vdash \gamma : Ch(Tv) \quad |Tv| \leq i}{\Sigma \vdash \gamma(x).P : \mathit{Proc}^\star(i)}\ (\mathit{input})$$

$$\frac{\Sigma, x : Tv \vdash P : \mathit{Proc}^\star(i) \quad \Sigma \vdash \gamma : Ch(Tv) \quad |Tv| \leq i}{\Sigma \vdash !\gamma(x).P : \mathit{Proc}^\star(i)}\ (!\mathit{input})$$

$$\frac{\Sigma \vdash P : \mathit{Proc}^\star(i) \quad \Sigma \vdash \lambda : \mathit{Loc}(j) \quad j \leq i}{\Sigma \vdash \mathtt{go}\ \lambda.P : \mathit{Proc}^\star(i)}\ (\mathit{go})$$

$$\frac{\Sigma \vdash P : \mathit{Proc}^\star(i)}{\Sigma \vdash \mathtt{go} \circlearrowleft .P : \mathit{ProcLocal}(i)}\ (\mathit{goHome}) \qquad \frac{\Sigma \vdash p : \mathit{Path}^\star}{\Sigma \vdash \mathtt{run}_p : \mathit{Proc}^\star(i)}\ (\mathit{run})$$

$$\frac{\Sigma \vdash p : \mathit{Path}^\star \quad \Sigma \cup \Sigma_\chi \vdash P : \mathit{Proc}^\star(i) \quad |\chi| \leq i}{\Sigma \vdash \mathtt{update}_p(\chi, \hat{\chi}).P : \mathit{Proc}^\star(i)}\ (\mathit{copy})$$

$$\frac{\begin{array}{c}\Sigma \vdash p : \mathit{Path}^\star \quad \Sigma \cup \Sigma_\chi \vdash P : \mathit{Proc}^\star(i) \\ \chi \neq x \quad |\chi| < i \quad \Sigma_\chi \vdash V : TPS(j) \quad j \leq i\end{array}}{\Sigma \vdash \mathtt{update}_p(\chi, V).P : \mathit{Proc}^\star(i)}\ (\mathit{paste})$$

$$\frac{\begin{array}{c}\Sigma, x : \mathit{ProcLocal}(i) \vdash P : \mathit{ProcLocal}(i) \\ x : \mathit{ProcLocal}(i) \vdash V : TPS(j) \quad j \leq i\end{array}}{\Sigma \vdash \mathtt{update}_\bullet(\Box x^i, V).P : \mathit{ProcLocal}(i)}\ (\mathit{pasteHere})$$

Table 12
Typing of processes

pattern.

$$\Sigma_\chi = \begin{cases} x : \mathit{ProcLocal}(j) \text{ if } \chi = \Box x^j, \\ x : \mathit{Loc}(j), y : \mathit{Path}^\star \text{ if } \chi = y^\star @ x^j, \\ x : \mathit{DLTree} \text{ if } \chi = x^{DL}, \\ x : \mathit{Tree} \text{ if } \chi = x. \end{cases}$$

$$\frac{\emptyset \vdash T : Tree \quad \emptyset \vdash P : Proc(i)}{\vdash l^i[\, T \parallel P \,] : Net}\ (netIloc)$$

$$\frac{\emptyset \vdash T : Tree \quad \emptyset \vdash P : Proc(j)}{\vdash l^i[\, T \parallel P \,] : Net}\ (netOloc)$$

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

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

Table 13
Typing of networks

We define the *security level* of a pattern (notation $|\chi|$) as expected:

- $|\Box x^j| = |y^\star @ x^j| = j$;
- $|x^{DL}| = |x| = \bot$.

In these rules $TPS(j)$ stands for $Tree$ or $Pointer^\star(j)$ or $Script(j)$.

The three typing rules for the updating command are necessary since we require:

- all processes to be allowed to copy all trees and to replace only data-less trees (rules ($copy$) and ($paste$));
- processes at the same security level of a leaf to be allowed to copy the leaf (rule ($copy$));
- processes at a higher security level than a leaf to be allowed to replace the leaf with a data term of a security level not greater than the security level of the process itself (rule ($paste$));
- a process script in a leaf to be allowed to replace itself with a data term of a security level not greater than its own security level (rule ($pasteHere$)).

As a consequence a process can replace a non data-less tree only if all the leaves of this tree contain data terms of security levels lower than the security level of the process itself. For this purpose the process needs first to replace all the leaves containing pointers and scripts by the empty tree and then to replace the so obtained data-less tree.

Typing rules for networks are given in Table 13. For typing a location in a network we have two typing rules: the initial rule ($netIloc$) and the ongoing rule ($netOloc$). The first rule requires the process to have the same security level of the enclosing location, while the second one allows a process of any security level. This reflects the requirement that access and mobility rights of processes depend on their source locations, as we will discuss in Section 4.

The function $\mathcal{N}$ associates to a network the set of its location names:

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

It is used in rule $(net|)$ to assure that each location name occurs at most once in a typed network.

The system satisfies subject reduction:

Theorem 3.1 (Subject reduction) *Let* $\vdash \mathbf{N} : Net$ *and* $\mathbf{N} \to \mathbf{N}'$*, then* $\vdash \mathbf{N}' : Net$.

The proof is presented in Appendix B. It uses some *Generation* and *Substitution* lemmas which are also presented in Appendix B.

4 Safety properties

In the present section, using the subject reduction, we can show some relevant properties of typed initial networks. We say that a network is *initial* when its locations can be typed by means of the initial typing rules.

More meaningful than the subject reduction theorem are the following properties of initial networks: [5]

P0 a channel in a process whose source location has level h can communicate only values whose security level is less than or equal to h;

P1 a process whose source location has level h can migrate to a location of level j only if $j \leq h$;

P2 a process whose source location has level h can copy from the local tree only data of level j with $j \leq h$;

P3 a process whose source location has level h can modify in the local tree only data of level j with $j < h$, unless the process itself was generated by running a script of security level h in a tree at path p, and in this case it can modify scripts which are both of the security level h and reachable by the path p;

P4 a script of level j which is a leaf of a tree in a location of level i can be activated only if $j \leq i$.

In order to discuss these properties we need to formalise the notion of "source" location of a process. Roughly by "source" location of a process we mean the location where the process was in the initial net or where the process was created by a `run` command.

We use $\twoheadrightarrow$ to denote the reflexive and transitive closure of $\to$ and $\vec{\nu}$ to denote a possibly empty sequence of channel restrictions. If $\mathbf{N}$ is an initial network and $\mathbf{N} \twoheadrightarrow \vec{\nu}(l^h[\ T \parallel P \mid Q\ ] \mid \mathbf{N}')$, then the *source* location of the process P in this reduction is defined by induction on the reduction $\twoheadrightarrow$ and by cases:

- if $\mathbf{N} \equiv \vec{\nu}(l^h[\ T \parallel P \mid Q\ ] \mid \mathbf{N}')$, then the source location of P is l^h;
- if $\mathbf{N} \twoheadrightarrow \vec{\nu}(l^h[\ T \parallel \mathtt{run}_p \mid Q'\ ] \mid \mathbf{N}') \to \vec{\nu}(l^h[\ T \parallel P \mid Q\ ] \mid \mathbf{N}')$ since $p(T) \leadsto_{p,l^h,\Box x^h,\Box x} T, \{\{\Box R_1/\Box x\}, \ldots, \{\Box R_n/\Box x\}\}$ and $R_1 \equiv P \mid R$ and $Q \equiv R \mid R_2 \mid \ldots \mid R_n \mid Q'$, then the source location of P is l^h;
- if $\mathbf{N} \twoheadrightarrow \vec{\nu}(l^h[\ T' \parallel \mathtt{update}_p(\chi, V).P' \mid Q'\ ]|\mathbf{N}') \to \vec{\nu}(l^h[\ T \parallel P \mid Q\ ]|\mathbf{N}')$ since $p(T') \leadsto_{p,l^h,\chi,V} T, \{\mathtt{s}_1, \ldots, \mathtt{s}_n\}$ and $P'\mathtt{s}_1 \equiv P|R$ and $Q \equiv R|\ P'\mathtt{s}_2|\ldots|\ P'\mathtt{s}_n|$

[5] Notice that **P0**, **P1**, **P2**, **P3** and **P4** are network invariants in the sense of [3].

Q', then the source location of P is the source location of $\texttt{update}_p(\chi, V).P'$ in the reduction without the last step;

- if $\mathbf{N} \twoheadrightarrow \vec{\nu}(l^h[\ T \parallel \bar{c}^{Tv}\langle v\rangle \mid c^{Tv}(z).P' \mid Q'\ ] \mid \mathbf{N}') \rightarrow \vec{\nu}(l^h[\ T \parallel P \mid Q\ ] \mid \mathbf{N}')$ and $P'\{v/z\} \equiv P \mid R$ and $Q \equiv R \mid Q'$, then the source location of P is the source location of $c^{Tv}(z).P'$ in the reduction without the last step;
- if $\mathbf{N} \twoheadrightarrow \vec{\nu}(l^h[\ T \parallel \bar{c}^{Tv}\langle v\rangle \mid !c^{Tv}(z).P' \mid Q'\ ] \mid \mathbf{N}') \rightarrow \vec{\nu}(l^h[\ T \parallel P \mid Q\ ] \mid \mathbf{N}')$ and $P'\{v/z\} \equiv P \mid R$ and $Q \equiv !c^{Tv}(z).P' \mid R \mid Q'$, then the source location of P is the source location of $!c^{Tv}(z).P'$ in the reduction without the last step;
- if $\mathbf{N} \twoheadrightarrow \vec{\nu}(l^h[\ T \parallel \texttt{go}\ l^h.P' \mid Q'\ ] \mid \mathbf{N}') \rightarrow \vec{\nu}(l^h[\ T \parallel P \mid Q\ ] \mid \mathbf{N}')$ and $P' \equiv P \mid R$ and $Q \equiv R \mid Q'$, then the source location of P is the source location of $\texttt{go}\ l^h.P'$ in the reduction without the last step;
- if $\mathbf{N} \twoheadrightarrow \vec{\nu}(l^h[T \parallel Q'\ ] \mid m^j[\ T' \parallel \texttt{go}\ l^h.P' \mid R\ ] \mid \mathbf{N}'') \rightarrow \vec{\nu}(l^h[\ T \parallel P \mid Q\ ] \mid \mathbf{N}')$ and $P' \equiv P \mid R'$ and $Q \equiv R' \mid Q'$ and $\mathbf{N}' \equiv m^j[\ T' \parallel R\ ] \mid \mathbf{N}''$, then the source location of P is the source location of $\texttt{go}\ l^h.P'$ in the reduction without the last step;
- if $\mathbf{N} \twoheadrightarrow \vec{\nu}(l^h[\ T' \parallel P \mid Q'\ ] \mid \mathbf{N}'') \rightarrow \vec{\nu}(l^h[\ T \parallel P \mid Q\ ] \mid \mathbf{N}')$, then the source location of P is the source location of P in the reduction without the last step.

The first two cases are the basic cases, in which the process P takes the current location as source location: in the first one the network is initial, in the second one the process P is generated by the last reduction step. In the last case the reduction does not modify the process P, which preserves its source location. In all other cases an action prefixing the process P (possibly in parallel with other processes and/or modulo the substitution of a value for a variable) is consumed and the source location of P is the source location of the process starting with that action in the reduction without the last step.

We can then formalise the above properties as follows.

Proposition 4.1 *If $\mathbf{N}$ is an initial network, and $\mathbf{N} \twoheadrightarrow \vec{\nu}(l^i[\ T \parallel P \mid Q\ ] \mid \mathbf{N}')$, and h is the security level of the source location of P, then:*

P0 $P \equiv \bar{c}^{Tv}\langle v\rangle$ *implies* $|Tv| \leq h$*;*
P1 $P \equiv \texttt{go}\ m^j.P'$ *implies* $j \leq h$*;*
P2 $P \equiv \texttt{update}_p(\chi, \hat{\chi}).P'$ *implies* $|\chi| \leq h$*;*
P3 $P \equiv \texttt{update}_p(\chi, V).P'$ *and* $V \neq \hat{\chi}$ *imply either* $\chi \neq x$ *and* $|\chi| < h$ *or* $\chi = \Box x^h$ *and P has been generated by activating a script in a tree at path p;*
P4 $P \equiv \texttt{run}_p$ *implies that the execution of P can only activate scripts at security level i.*

Property **P4** is an immediate consequence of the reduction rule (run). The remaining properties follow easily observing that each process has the security level of its source location: see Appendix C for the proof.

Proposition 4.2 *If $\mathbf{N}$ is an initial network, and $\mathbf{N} \twoheadrightarrow \vec{\nu}(l^i[\ T \parallel P \mid Q\ ] \mid \mathbf{N}')$, and h is the security level of the source location of P, then* $\vdash P : \mathit{Proc}(h)$.

5 Examples

We will consider natural numbers with their order as security levels in many of the following examples.

5.1 Examples of process reduction

Let us suppose that P and Q are processes of security level 2 and R is a process of security level 3. If

$$T \equiv \mathtt{c}[\mathtt{b}[\mathtt{a}[\Box P]] \mid \mathtt{b}[\mathtt{a}[\Box Q]] \mid \mathtt{b}[\mathtt{a}[\Box R]]]$$

and $p = \mathtt{c/b/a}$ then

$$p(T) \equiv \mathtt{c}[\mathtt{b}[\mathtt{a}[\underline{\Box P}]] \mid \mathtt{b}[\mathtt{a}[\underline{\Box Q}]] \mid \mathtt{b}[\mathtt{a}[\underline{\Box R}]]].$$

(a) If the source location of the process $\mathtt{copy}_{\mathtt{c/b/a}}(\Box x^2).P'$ is of level 2, then this process will find and copy scripts of level 2.

$$l^2[\ \mathtt{c}[\mathtt{b}[\mathtt{a}[\Box P]] \mid \mathtt{b}[\mathtt{a}[\Box Q]] \mid \mathtt{b}[\mathtt{a}[\Box R]]] \mid\mid \mathtt{copy}_{\mathtt{c/b/a}}(\Box x^2).P'] \rightarrow$$
$$l^2[\ \mathtt{c}[\mathtt{b}[\mathtt{a}[\Box P]] \mid \mathtt{b}[\mathtt{a}[\Box Q]] \mid \mathtt{b}[\mathtt{a}[\Box R]]] \mid\mid P'\{\Box P/\Box x\} \mid P'\{\Box Q/\Box x\}].$$

(b) If the source location of the process $\mathtt{update}_{\mathtt{c/b/a}}(\Box x^2, \emptyset).R'$ is of level 3, then the scripted processes of level 2 in the tree can even be modified.

$$l^2[\ \mathtt{c}[\mathtt{b}[\mathtt{a}[\Box P]] \mid \mathtt{b}[\mathtt{a}[\Box Q]] \mid \mathtt{b}[\mathtt{a}[\Box R]]] \mid\mid \mathtt{update}_{\mathtt{c/b/a}}(\Box x^2, \emptyset).R'] \rightarrow$$

$$l^2[\ \mathtt{c}[\mathtt{b}[\mathtt{a}[\emptyset]] \mid \mathtt{b}[\mathtt{a}[\emptyset]] \mid \mathtt{b}[\mathtt{a}[\Box R]]] \mid\mid R'\{\Box P/\Box x\} \mid R'\{\Box Q/\Box x\}].$$

(c) The activation of the process $Q \equiv \mathtt{update}_{\bullet}(\Box x^2, \emptyset).\mathtt{update}_{//}(x^{DL}, T_1).0$ of level 2 can modify the scripts of level 2 identified by the same path as the script $\Box Q$. In this way starting from the tree $\mathtt{c}[\mathtt{b}[\mathtt{a}[\Box P]] \mid \mathtt{b}[\mathtt{a}[\Box Q]]]$ a data-less tree is obtained and it is replaced by the tree T_1.

$$l^2[\ \mathtt{c}[\mathtt{b}[\mathtt{a}[\Box P]] \mid \mathtt{b}[\mathtt{a}[\Box Q]]] \mid\mid \mathtt{run}_{\mathtt{c/b/a}}] \rightarrow$$
$$l^2[\ \mathtt{c}[\mathtt{b}[\mathtt{a}[\Box P]] \mid \mathtt{b}[\mathtt{a}[\Box Q]]] \mid\mid P \mid \mathtt{update}_{\mathtt{c/b/a}}(\Box x^2, \emptyset).\mathtt{update}_{//}(x^{DL}, T_1).0] \rightarrow$$
$$l^2[\ \mathtt{c}[\mathtt{b}[\mathtt{a}[\emptyset]] \mid \mathtt{b}[\mathtt{a}[\emptyset]]] \mid\mid P \mid \mathtt{update}_{//}(x^{DL}, T_1).0] \rightarrow$$
$$l^2[\ \mathtt{c}[T_1] \mid\mid P].$$

Instead if the initial tree contains also a script of level 3 that script cannot be activated and it cannot be modified by Q. Therefore the final replacement does not involve the subtree containing that script.

$$l^2[\ \mathtt{c}[\mathtt{b}[\mathtt{a}[\Box P]] \mid \mathtt{b}[\mathtt{a}[\Box Q]] \mid \mathtt{b}[\mathtt{a}[\Box R]]] \mid\mid \mathtt{run}_{\mathtt{c/b/a}}] \rightarrow$$

$$l^2[\mathtt{c}[\mathtt{b}[\mathtt{a}[\Box P]] | \mathtt{b}[\mathtt{a}[\Box Q]] | \mathtt{b}[\mathtt{a}[\Box R]]] || P | \mathtt{update}_{\mathtt{c/b/a}}(\Box x^2, \emptyset).\mathtt{update}_{//}(x^{DL}, T_1).0] \rightarrow$$

$$l^2[\ \mathtt{c}[\mathtt{b}[\mathtt{a}[\emptyset]] \mid \mathtt{b}[\mathtt{a}[\emptyset]] \mid \mathtt{b}[\mathtt{a}[\Box R]]] \mid\mid P \mid \mathtt{update}_{//}(x^{DL}, T_1).0] \rightarrow$$

$$l^2[\ \mathtt{c}[\mathtt{b}[T_1] \mid \mathtt{b}[T_1] \mid \mathtt{b}[\mathtt{a}[\Box R]]] \mid\mid P].$$

5.2 Insensitivity to higher level data terms

The security policy enforced by our typing system should not be confused with non-interference. A high level process can easily declassify information of its security level to lower levels. However in the absence of high level processes, lower level processes are insensitive even to the existence of higher level data.

Consider the following networks

$$\mathbf{N}_1 = l^h[\, T \,||\, P \,]$$

and

$$\mathbf{N}_2 = l^h[\, T \,||\, P \,] \mid m^j[\, T' \,||\, 0 \,]$$

where $h < j$.

Then the following property holds

Proposition 5.1 *For each* $\mathbf{N}$ *such that* $\mathbf{N}_2 \twoheadrightarrow \mathbf{N}$ *there is* $\mathbf{N}'$ *such that* $\mathbf{N} = \mathbf{N}' \mid m^j[\, T' \,||\, 0 \,]$ *and* $\mathbf{N}_1 \twoheadrightarrow \mathbf{N}'$*.*

Proof We will give the proof by induction on $\twoheadrightarrow$. Let us suppose that:

$$\mathbf{N}_2 \twoheadrightarrow l^h[\, T_1 \,||\, P_1 \,] \mid m^j[\, T' \,||\, 0 \,] \rightarrow \mathbf{N} \text{ and } \mathbf{N}_1 \twoheadrightarrow l^h[\, T_1 \,||\, P_1 \,].$$

We have the following cases:

- $P_1 \equiv \bar{c}^{Tv}\langle v \rangle \mid c^{Tv}(z).P' \mid Q$ and $\mathbf{N}' \equiv l^h[\, T_1 \,||\, P'\{v/z\} \mid Q \,]$;
- $P_1 \equiv \bar{c}^{Tv}\langle v \rangle \mid !c^{Tv}(z).P' \mid Q$ and $\mathbf{N}' \equiv l^h[\, T_1 \,||\, !c^{Tv}(z).P' \mid P'\{v/z\} \mid Q \,]$;
- $P_1 \equiv \mathtt{go}\ l^h.P' \mid Q$ and $\mathbf{N}' \equiv l^h[\, T_1 \,||\, P' | Q \,]$;
- $P_1 \equiv \mathtt{run}_p \mid Q$ and $\mathbf{N}' \equiv l^h[\, T_1 \,||\, P'_1 \mid \ldots \mid P'_n \mid Q \,]$;
- $P_1 \equiv \mathtt{update}_p(\chi, V).P' \mid Q'$ and $\mathbf{N}' \equiv l^h[\, T_1 \,||\, P' s_1 \mid \ldots \mid P' s_n \mid Q' \,]$.

All these cases imply $\mathbf{N}_1 \twoheadrightarrow \mathbf{N}'$ and $\mathbf{N} = \mathbf{N}' \mid m^j[\, T' \,||\, 0 \,]$. By the property **P1** of Proposition 4.1 and the assumption $h < j$, the case $P_1 \equiv \mathtt{go}\ m^j.P' \mid Q$ is not possible.

Another similar result is the following. Let $\mathbf{N}$ be a network all whose locations have security level less than or equal to h. Let V, V' be data terms of security levels $j, j' > h$, respectively. Then

Proposition 5.2 *Under the above conditions we have* $\mathbf{N} \twoheadrightarrow \mathbf{N}'$ *if and only if* $\mathbf{N}[V'/V] \twoheadrightarrow \mathbf{N}'[V'/V]$*,where* $\mathbf{N}[V'/V]$ *is the network obtained by replacing in* $\mathbf{N}$ *some occurrences of* V *in the leaves with* V'*.*

Proof The proof is again by induction on $\twoheadrightarrow$. The only interesting cases is when a run or an update command is executed.

Let

$$\mathbf{N} \twoheadrightarrow l^{h'}[\, T \,||\, \mathtt{run}_p \mid Q' \,] \mid \mathbf{N}_1 \rightarrow l^{h'}[\, T \,||\, P_1 \mid \ldots \mid P_n \mid Q' \,] \mid \mathbf{N}_1$$

and

$$\mathbf{N}[V'/V] \twoheadrightarrow l^{h'}[\, T[V'/V] \,||\, \mathtt{run}_p \mid Q' \,] \mid \mathbf{N}_1[V'/V].$$

If V is a script it cannot be activated by property **P4** of Proposition 4.1. So we get $l^{h'}[\, T[V'/V] \,||\, \texttt{run}_p \mid Q' \,] \mid \mathbf{N}_1[V'/V] \;\rightarrow\; l^{h'}[\, T[V'/V] \,||\, P_1 \mid \ldots \mid P_n \mid Q' \,] \mid \mathbf{N}_1[V'/V]$.

Let

$\mathbf{N} \twoheadrightarrow l^{h'}[\, T \,||\, \texttt{update}_p(\chi, V_0).P \mid Q' \,]|\mathbf{N}_1 \rightarrow l^{h'}[\, T' \,||\, P\mathtt{s}_1 \mid \ldots \mid P\mathtt{s}_n \mid Q' \,]|\mathbf{N}_1$

and

$$\mathbf{N}[V'/V] \twoheadrightarrow l^{h'}[\, T[V'/V] \,||\, \texttt{update}_p(\chi, V_0).P \mid Q' \,]\, |\mathbf{N}_1[V'/V].$$

By properties **P2** and **P3** of Proposition 4.1 the security level of χ must be less than or equal to h. For this reason the occurrences of the data term V in T can be neither modified nor copied and therefore

$l^{h'}[\, T[V'/V] \,||\, \texttt{update}_p(\chi, V_0).P \mid Q' \,] \mid \mathbf{N}_1[V'/V] \rightarrow l^{h'}[\, T'[V'/V] \,||\, P\mathtt{s}_1 \mid \ldots \mid P\mathtt{s}_n \mid Q' \,] \mid \mathbf{N}_1[V'/V]$.

5.3 Remote Voting System

The next example models a remote voting for election of a leader from a given list of candidates, inspired by [13]. In this example, we allow leaves to contain integers, in order to represent the counters of votes. A pattern too can be a variable of type *Integer* and of a fixed security level.

The network consists of an authority location, a polling station location and a fixed number of voter locations. The authority location has level 3, while the polling station and all the voter locations have level 1.

The polling station location

$$station^1[\texttt{voterList}[\, \ldots \mid \texttt{voterId}[\, \Box P \,]|| \, \ldots] \mid \texttt{candList}[\, T \,] \,||\, 0 \,],$$

where $P = (\nu c^{Path})(\texttt{cut}_\bullet(\Box x^1).\texttt{go}\ voter^1.\bar{b}^{Ch(Path)}\langle c^{Path}\rangle \mid \bar{d}^{Ch(Path)}\langle c^{Path}\rangle)$ and $T = \ldots \mid \texttt{name}[\, 0^2 \,] \mid \ldots$, contains as data the voter list and the candidate list with counters of votes.

The voter list has for each voter an edge labelled by the voter identifier pointing to the scripted process $\Box P$ of security level 1. This script contains two processes. One process first destroys itself and then goes to the voter location, where it communicates a secret channel which the voter will use to express his vote. The other process simply communicates the same secret channel via the channel d.

The candidate list has for each candidate an edge labelled by the candidate name pointing to an integer (the vote counter, initially 0) of security level 2. This assures that the *voter* can copy the subtree with candidate list and see candidate names, but by property **P2** of Proposition 4.1 he cannot see and use already recorded votes to make his decision.

A voter location contains two processes: the first process goes to the station and activates the process P and the second one waits to receive a channel along which he will communicate his vote, after going to the station and making a choice (based on the candidate list):

$$voter^1[\ \ldots\ \|\ \texttt{go}\ station^1.\texttt{run}_{\texttt{voterList/voterId}}\ |$$
$$b^{Ch(Path)}(y).\texttt{go}\ station^1.Choice(z).\bar{y}\langle z\rangle\ |\ \ldots].$$

The process in the authority location starts the elections by going to the station where he repeatedly collects one private channel via the channel d, receives along this private channel one candidate name and increases by 1 the corresponding candidate counter:

$$authority^3[\ \texttt{Start}[\Box Q]\ |\ \ldots\ \|\ \texttt{run}_{\texttt{Start}}\ |\ \ldots\ ],$$

$$Q = \texttt{go}\ station^1.!d^{Path}(v).v(w).\texttt{update}_{\texttt{candVoteList}/w}(t^2, t+1).$$

Similarly the authority can end the election going to the station and erasing the voter list.

Notice that a malicious voter cannot vote more than once, since the process P destroys itself, and if he would send the identifier of another voter, the other voter would receive the secret channel to vote. Moreover by property **P3** of Proposition 4.1 a malicious voter cannot change the vote counters in the station location, since the vote counters have security level 2, while the voters have security level 1.

A malicious voter can send to the location of another voter a process which votes in place of the voter itself. We do not know how to avoid this kind of attacks, which model a voter stealing the position of another voter during the voting act. We cannot avoid this by giving incomparable security levels to the voter locations, since they all must have the same security level of the polling station in order to allow processes go from the polling station to the voter locations and vice versa.

The present encoding is simpler than the encoding of the same example given in [8].

5.4 Distributed Library

Let us consider a network consisting of a distributed library (main library and libraries of specific fields), readers, staff members and a head. The main library ($Library$) has data subtrees for management and catalogue. The library catalogue contains in its leaves pointers to full books which are distributed in leaves of specific field libraries.

$$Library^1\ [\texttt{Management}\ [\ \texttt{WorkingHours}[HourPlan^2]\ |\ \ldots\ ]\ |$$
$$\texttt{Catalog}\ [\ \ldots |\ \texttt{Pierce}[\texttt{Types}[\texttt{Pierce/Types}@LICS^1]\ |$$
$$\texttt{Category}[\texttt{Pierce/Category}@LICS^1]\ldots]\ |$$
$$|\ \texttt{Cohn}[\texttt{Universal}[\texttt{Cohn/Universal}@ALGEBRA^1]\ |\ \ldots]$$
$$\|\ \ldots],$$

$LICS^1[\ \dots |$ $\texttt{Pierce}\ [\ \texttt{Types}\ [\ Book.pdf^1\ ]\ |\ \texttt{Category}\ [\ Book.pdf^1]\ |\ \dots\ ]\ ||\ \dots\ ],$

$ALGEBRA^1\ [\ \dots | \dots \texttt{Cohn}\ [\ \texttt{Universal}\ [\ Book.pdf^1\ ]\ |\ \dots\ ]\ ||\ \dots\ ].$

For example, the reader

$Reader^1[\texttt{Book}[\texttt{Pierce}[\emptyset]\ \ |\ \dots\ ]\ ||\ \texttt{go}\ Library^1.\texttt{copy}_{\texttt{Catalog/Pierce/Types}}(y@x^1).$
$\texttt{go}\ x.\texttt{copy}_{\texttt{Pierce/Types}}(z^1).\texttt{go}\ Reader^1.\texttt{paste}_{\texttt{Book/Pierce}}\langle \texttt{Types}[z]\rangle\ ]$

goes to the library, reads in the catalogue the location of the book, goes to the sublibrary, copies the book and pastes the copy in the tree of his location.

The typing system introduced in the current paper assures that the reader can copy content of any book, but he cannot modify it (property **P3** of Proposition 4.1). Besides, he cannot see $HourPlan$ in the management leaf, because he is of a lower security level than the $HourPlan$ (property **P2** of Proposition 4.1).

The staff is given security level 2, such that they can update catalogue, modify the book contents, but only copy the $HourPlan$.

The head, being of security level 3, is the only one that can update all the data at the $Library$. He can, for example, change working hours.

6 Conclusion

We have discussed a typed version of the X$d\pi$ calculus in which the access to resources and the mobility of processes must respect a security policy. Since we used a typed pattern matching which includes a dynamic type checking we will investigate both type checking and type inference for this calculus, taking into account [7].

An obvious alternative approach is to tag active processes syntactically with their security levels. Initial networks must only have active processes tagged with levels at most equal to the levels of their containing locations. In this way one gains in flexibility since processes in the same location can have different security levels. The price to pay is an heavier syntax.

More expressivity could be achieved by associating security levels to tree branches. We leave the study of this variation as future work.

We plan to investigate modifications of our type system which prevent illegal flow of information [18], also in presence of dynamic flow policies [22].

We want to study the impact of our typing system in proving equivalence of networks, using different notions of behavioural equivalence.We plan to start from the untyped equivalences defined in [14] and [9], and to refine them using types as done for example in [17] and [12].

Acknowledgements We thank Philippa Gardner and Sergio Maffeis for their careful reading of an earlier version of the paper and for many useful remarks on it. We also thank the anonymous referees of TGC'06 submission and of the present submission for detailed and appropriate comments. The final version of the paper strongly improved due to their suggestions.

References

[1] Serge Abiteboul, Omar Benjelloun, Bogdan Cautis, and Tova Milo. Active XML, Security and Access Control. In Sérgio Lifschitz, editor, *SBBD'04*, pages 13–22, 2004.

[2] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web : From Relations to Semistructured Data and XML*. Data Management Systems. Morgan Kaufmann, 1999.

[3] Alex Ahern and Nobuko Yoshida. Formalising Java RMI with Explicit Code Mobility. In Richard P. Gabriel and Ralph Johnson, editors, *OOPSLA'05*, pages 403–422. ACM Press, 2005.

[4] Luca Cardelli and Giorgio Ghelli. A Query Language Based on the Ambient Logic. In David Sands, editor, *ESOP'01*, volume 2028 of *LNCS*, pages 1–22. Springer-Verlag, 2004. Invited Paper.

[5] Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Types for the Ambient Calculus. *Information and Computation*, 177(2):160–194, 2002.

[6] Giuseppe Castagna, Jan Vitek, and Francesco Zappa Nardelli. The Seal Calculus. *Information and Computation*, 201(1):1–54, 2005.

[7] Mario Coppo, Federico Cozzi, Mariangiola Dezani-Ciancaglini, Elio Giovannetti, and Rosario Pugliese. A Mobility Calculus with Local and Dependent Types. In Aart Middeldorp, Vincent van Oostrom, Femke van Raamsdonk, and Roel de Vrijer, editors, *Processes, Terms and Cycles: Steps on the Road to Infinity*, volume 3838 of *LNCS*, pages 404–444. Springer-Verlag, 2005.

[8] Mariangiola Dezani-Ciancaglini, Silvia Ghilezan, and Jovanka Pantovic. Security Types for Dynamic Web Data. In Ugo Montanari, Donald Sannella, and Roberto Bruni, editors, *TGC'06*, volume 4661 of *LNCS*, pages 263–280. Springer-Verlag, 2007.

[9] Philippa Gardner and Sergio Maffeis. Modelling Dynamic Web Data. *Theoretical Computer Science*, 342:104–131, 2005.

[10] Mattew Hennessy and James Riely. Resource Access Control in Systems of Mobile Agents. *Information and Computation*, 173:82–120, 2002.

[11] Mattew Hennessy and James Riely. Information Flow vs Resource Access in the Asynchronous π-calculus. *ACM Transactions on Programming Languages and Systems*, 5:566–591, 2003.

[12] Matthew Hennessy and Julian Rathke. Typed Behavioural Equivalences for Processes in the Presence of Subtyping. *Mathematical Structures in Computer Science*, 14(5):651–684, 2003.

[13] Joseph Kiniry, Alan Morkan, Fintan Fairmichael, Dermot Cochran, Patrice Chalin, Martijn Oostdijk, and Engelbert Hubbers. The KOA Remote Voting System: A Summary of Work To-Date. In Ugo Montanari, Donald Sannella, and Roberto Bruni, editors, *TGC'06*, volume 4661 of *LNCS*, pages 244–262. Springer-Verlag, 2007.

[14] Sergio Maffeis and Philippa Gardner. Behavioural Equivalencies for Dynamic Web Data. In Jean-Jacques Lévy, Ernst W. Mayr, and John C. Mitchell, editors, *TCS'04*, pages 541–554. Kluwer, 2004.

[15] Robin Milner, Joachim Parrow, and David Walker. A Calculus of Mobile Processes, I-II. *Information and Computation*, 100(1):1–77, 1992.

[16] Rocco De Nicola, Gianluigi Ferrari, Rosario Pugliese, and Betti Venneri. Types for Access Control. *Theoretical Computer Science*, 240(1):215–254, 2000.

[17] Benjamin Pierce and Davide Sangiorgi. Behavioral Equivalence in the Polymorphic Pi-Calculus. *Journal of the ACM*, 47(3):531–584, 2000.

[18] Andrei Sabelfeld and Andrew C. Myers. Language-Based Information-Flow Security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, 2003.

[19] Arnaud Sahuguet. *ubQL: A Distributed Query Language to Program Distributed Query Systems*. PhD thesis, Penn University, 2002.

[20] Ravi S. Sandhu. Lattice-based Access Control Models. *IEEE Computer*, 26(11):9–19, 1993.

[21] Davide Sangiorgi and David Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

[22] Steve Zdancewic. Challenges for Information-flow Security. In Roberto Giacobazzi, editor, *PLID'04*, 2004. Invited Paper.

A Structural Equivalence

The structural equivalence for the X$d\pi$ calculus is the least equivalence relation on networks that satisfies alpha-conversion, the commutative monoid properties for $(\emptyset, \mid)$ on trees, for $(0, \mid)$ on processes and for $(\mathbf{0}, \mid)$ on networks, and the axioms of Table A.1. As usual *fn* is the set of free channel names occurring in a process or in a tree or in a network.

(trees)	$V \equiv V' \Rightarrow \mathtt{a}[V] \equiv \mathtt{a}[V']$
(scripts)	$P \equiv P' \Rightarrow \Box P \equiv \Box P'$
(processes)	$(\nu c^{Tv})0 \equiv 0$
	$v \equiv v' \Rightarrow \bar{c}^{Tv}\langle v\rangle \equiv \bar{c}^{Tv}\langle v'\rangle$
	$(\nu c^{Tv})(\nu d^{Tv'})P \equiv (\nu d^{Tv'})(\nu c^{Tv})P$
	$c^{Tv} \notin fn(P) \Rightarrow P \mid (\nu c^{Tv})Q \equiv (\nu c^{Tv})(P \mid Q)$
	$V \equiv V' \Rightarrow \mathtt{update}_p(\chi, V).P \equiv \mathtt{update}_p(\chi, V').P$
(networks)	$(\nu c^{Tv})\mathbf{0} \equiv \mathbf{0}$
	$(\nu c^{Tv})(\nu d^{Tv'})\mathbf{N} \equiv (\nu d^{Tv'})(\nu c^{Tv})\mathbf{N}$
	$c^{Tv} \notin fn(\mathbf{N}) \Rightarrow \mathbf{N} \mid (\nu c^{Tv})\mathbf{N}' \equiv (\nu c^{Tv})(\mathbf{N} \mid \mathbf{N}')$
	$T \equiv T' \wedge P \equiv P' \Rightarrow l^h[\, T \parallel P \,] \equiv l^h[\, T' \parallel P' \,]$
	$c^{Tv} \notin fn(T) \Rightarrow l^h[\, T \parallel (\nu c^{Tv})P \,] \equiv (\nu c^{Tv})l^h[\, T \parallel P \,]$

Table A.1
Structural equivalence

B 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. Notice that for networks we need to distinguish initial and ongoing typing rules.

We use τ to range over all types of Table 9.

Lemma B.1 (Generation lemma for variables, channels, locations and paths)

(1) $\Sigma \vdash x : \tau \Rightarrow x : \tau \in \Sigma$.
(2) $\Sigma \vdash c^{Tv} : \tau \Rightarrow \tau = Ch(Tv)$.
(3) $\Sigma \vdash l^i : \tau \Rightarrow \tau = Loc(i)$.
(4) $\Sigma \vdash p : \tau$ *and* p *is a local path* $\Rightarrow \tau = PathLocal$.
(5) $\Sigma \vdash p : \tau$ *and* p *is not a local path* $\Rightarrow \tau = Path^{\star}$.

Lemma B.2 (Generation lemma for scripts, pointers and trees)

(1) $\Sigma \vdash \Box\Pi : \tau \Rightarrow \tau = \mathit{Script}(i)$ *and* $\Sigma \vdash \Pi : \mathit{Proc}^\star(i)$.
(2) $\Sigma \vdash p@\lambda : \tau \Rightarrow \tau = \mathit{Pointer}^\star(i)$ *and* $\Sigma \vdash \lambda : \mathit{Loc}(i)$ *and* $\Sigma \vdash p : \mathit{Path}^\star$.
(3) $\Sigma \vdash \emptyset : \tau \Rightarrow$ *either* $\tau = DLTree$ *or* $\tau = \mathit{Tree}^\star$.
(4) $\Sigma \vdash T_1 \mid T_2 : \tau \Rightarrow$ *either* $\tau = DLTree$ *and* $\Sigma \vdash T_1 : DLTree$ *and* $\Sigma \vdash T_2 : DLTree$ *or* $\tau = \mathit{Tree}^\star$ *and* $\Sigma \vdash T_1 : \mathit{Tree}^\star$ *and* $\Sigma \vdash T_2 : \mathit{Tree}^\star$.
(5) $\Sigma \vdash \mathtt{a}[T] : \tau \Rightarrow$ *either* $\tau = DLTree$ *and* $\Sigma \vdash T : DLTree$ *or* $\tau = \mathit{Tree}^\star$ *and* $\Sigma \vdash T : \mathit{Tree}^\star$.
(6) $\Sigma \vdash \mathtt{a}[p@\lambda] : \tau \Rightarrow \tau = \mathit{Tree}^\star$ *and* $\Sigma \vdash p@\lambda : \mathit{Pointer}^\star(i)$.
(7) $\Sigma \vdash \mathtt{a}[\Box\Pi] : \tau \Rightarrow \tau = \mathit{Tree}^\star$ *and* $\Sigma \vdash \Box\Pi : \mathit{Script}(i)$.

Lemma B.3 (Generation lemma for processes)

(1) $\Sigma \vdash 0 : \tau \Rightarrow \tau = \mathit{Proc}^\star(i)$.
(2) $\Sigma \vdash P_1 \mid P_2 : \tau \Rightarrow \tau = \mathit{Proc}^\star(i)$ *and* $\Sigma \vdash P_1 : \mathit{Proc}^\star(i)$ *and* $\Sigma \vdash P_2 : \mathit{Proc}^\star(i)$.
(3) $\Sigma \vdash (\nu c^{Tv})P : \tau \Rightarrow \tau = \mathit{Proc}^\star(i)$ *and* $\Sigma \vdash P : \mathit{Proc}^\star(i)$ *and* $|Tv| \leq i$.
(4) $\Sigma \vdash \bar{\gamma}\langle v\rangle : \tau \Rightarrow \tau = \mathit{Proc}^\star(i)$ *and* $\Sigma \vdash v : Tv$ *and* $\Sigma \vdash \gamma : Ch(Tv)$ *and* $|Tv| \leq i$.
(5) $\Sigma \vdash \gamma(x).P : \tau \Rightarrow \tau = \mathit{Proc}^\star(i)$ *and* $\Sigma, x : Tv \vdash P : \mathit{Proc}^\star(i)$ *and* $\Sigma \vdash \gamma : Ch(Tv)$ *and* $|Tv| \leq i$.
(6) $\Sigma \vdash !\gamma(x).P : \tau \Rightarrow \tau = \mathit{Proc}^\star(i)$ *and* $\Sigma, x : Tv \vdash P : \mathit{Proc}^\star(i)$ *and* $\Sigma \vdash \gamma : Ch(Tv)$ *and* $|Tv| \leq i$.
(7) $\Sigma \vdash \mathtt{go}\,\lambda.P : \tau \Rightarrow \tau = \mathit{Proc}^\star(i)$ *and* $\vdash \lambda : \mathit{Loc}(j)$ *and* $j \leq i$ *and* $\Sigma \vdash P : \mathit{Proc}^\star(i)$.
(8) $\Sigma \vdash \mathtt{go}\,\circlearrowleft.P : \tau \Rightarrow \tau = \mathit{ProcLocal}(i)$ *and* $\Sigma \vdash P : \mathit{Proc}^\star(i)$.
(9) $\Sigma \vdash \mathtt{run}_p : \tau \Rightarrow \tau = \mathit{Proc}^\star(i)$ *and* $\Sigma \vdash p : \mathit{Path}^\star$.
(10) $\Sigma \vdash \mathtt{update}_p(\chi, \hat{\chi}).P : \tau \Rightarrow \tau = \mathit{Proc}^\star(i)$ *and* $\Sigma \vdash p : \mathit{Path}^\star$ *and* $\Sigma \cup \Sigma_\chi \vdash P : \mathit{Proc}^\star(i)$ *and* $|\chi| \leq i$.
(11) $\Sigma \vdash \mathtt{update}_p(\chi, V).P : \tau$ *and* $\chi \neq x$ *and* $V \neq \hat{\chi}$ *and* ($p \neq \bullet$ *or* $\chi \neq \Box x^j$ *for all* j) $\Rightarrow \tau = \mathit{Proc}^\star(i)$ *and* $\Sigma \vdash p : \mathit{Path}^\star$ *and* $\Sigma \cup \Sigma_\chi \vdash P : \mathit{Proc}^\star(i)$ *and* $|\chi| < i$ *and* $\Sigma_\chi \vdash V : TPS(j)$ *and* $j \leq i$.
(12) $\Sigma \vdash \mathtt{update}_\bullet(\Box x^i, V).P : \tau \Rightarrow \tau = \mathit{ProcLocal}(i)$ *and* $\Sigma, x : \mathit{ProcLocal}(i) \vdash P : \mathit{ProcLocal}(i)$ *and* $x : \mathit{ProcLocal}(i) \vdash V : TPS(j)$ *and* $j \leq i$.

Lemma B.4 (Generation lemma for networks) *(1)* $\vdash \mathbf{0} : \tau \Rightarrow \tau = Net$.
(2) $\vdash \mathbf{N}_1 \mid \mathbf{N}_2 : \tau \Rightarrow \tau = Net$ *and* $\vdash \mathbf{N}_1 : Net$ *and* $\vdash \mathbf{N}_2 : Net$ *and* $\mathcal{N}(\mathbf{N}_1) \cap \mathcal{N}(\mathbf{N}_2) = \emptyset$.
(3) $\vdash l^i[\, T \parallel P \,] : \tau \Rightarrow \tau = Net$ *and* $\emptyset \vdash T : Tree$ *and*
- *either* **(initial)** $\emptyset \vdash P : \mathit{Proc}(i)$,
- *or* **(ongoing)** $\emptyset \vdash P : \mathit{Proc}(j)$.

(4) $\vdash (\nu c^{Tv})\mathbf{N} : \tau \Rightarrow \tau = Net$ *and* $\vdash \mathbf{N} : Net$.

The following two propositions point out some properties of our type system and can be easily verified by induction of deductions.

By replacing in an arbitrary process "$\circlearrowleft$" by a location name (whose security level agrees with that of the process) and "$\bullet$" by a path not containing "$\bullet$" we get a process typeable with a process type.

Proposition B.5 *If* $\Sigma \vdash P : Proc^\star(i)$ *and* $\Sigma \vdash p : Path$ *and* $j \le i$, *then* $\Sigma \vdash P\{l^j/\circlearrowleft, p/\bullet\} : Proc(i)$.

A process which has a given security level has also all bigger security levels. The proof follows easily observing that the nil process can be typed with an arbitrary security level and that all typing rules only check that the security level of the current process is bigger than other security levels.

Proposition B.6 $\Sigma \vdash P : Proc^\star(i)$ *and* $i \le j$ *imply* $\Sigma \vdash P : Proc^\star(j)$.

As usual the "core" of the subject reduction proofs are substitution lemmas.

Lemma B.7 (Substitution lemma for trees, pointers, scripts and processes)

(1) *If* $\Sigma, x : Tv \vdash V : TPS(i)$ *and* $\Sigma \vdash v : Tv$, *then* $\Sigma \vdash V\{v/x\} : TPS(i)$.
(2) *If* $\Sigma, x : ProcLocal(j) \vdash V : TPS(i)$ *and* $\Sigma \vdash P : ProcLocal(j)$, *then* $\Sigma \vdash V\{\Box P/\Box x\} : TPS(i)$.
(3) *If* $\Sigma, x : Tv \vdash P : Proc^\star(i)$ *and* $\Sigma \vdash v : Tv$, *then* $\Sigma \vdash P\{v/x\} : Proc^\star(i)$.
(4) *If* $\Sigma, x : ProcLocal(j) \vdash P : Proc^\star(i)$ *and* $\Sigma \vdash Q : ProcLocal(j)$, *then* $\Sigma \vdash P\{\Box Q/\Box x\} : Proc^\star(i)$.
(5) *If* $\Sigma \vdash \mathtt{update}_p(\chi, V).P : Proc(i)$ *and* $\Sigma \vdash T : Tree$ *and* $p(T) \leadsto_{p,l^i,\chi,V} T', \Theta$, *then* $\Sigma \vdash T' : Tree$.

Proof The proofs of the first four points are standard by induction on V and P, respectively.

For (5) we need to consider three cases according to the shape of χ. We give the proof for $\chi = y^\star @ x^j$, the remaining cases being similar. Let $\Theta = \{\mathtt{s}_1, \dots, \mathtt{s}_n\}$ and $1 \le k \le n$. By construction $\mathtt{s}_k = \{m^j/x, p'_k/y\}$, for some m^j and p'_k such that $\vdash p'_k : Path^\star$. By Lemma B.3(10) or (11) $\Sigma, x : Loc(j), y : Path^\star \vdash V : TPS(h)$ for some $h \le i$. By Point (1) $\Sigma \vdash V\mathtt{s}_k : TPS(h)$. By construction T' is obtained from T by replacing top-down the nodes $m^j @ p'_k$ by $V\mathtt{s}_k$, so we can easily check that $\Sigma \vdash T' : Tree$ using the typing rules for trees.

Theorem 3.1 (Subject reduction) *Let* $\vdash \mathbf{N} : Net$ *and* $\mathbf{N} \to \mathbf{N}'$, *then* $\vdash \mathbf{N}' : Net$.

Proof We only consider some interesting cases.

Case $\mathbf{N} \equiv l^h[\, T_1 \,||\, \mathsf{go}\ m^j.P \mid Q \,] \mid m^j[\, T_2 \,||\, R \,]$ and the reduction is by rule (go):

$$l^h[\, T_1 \,||\, \mathsf{go}\ m^j.P \mid Q \,] \mid m^j[\, T_2 \,||\, R \,] \to l^h[\, T_1 \,||\, Q \,] \mid m^j[\, T_2 \,||\, P \mid R \,].$$

From $\vdash \mathbf{N} : Net$, by Lemma B.4(2) it follows that $\vdash \mathbf{N}_1 \equiv l^h[\, T_1 \,||\, \mathsf{go}\ m^j.P \mid Q \,] : Net$ and $\vdash \mathbf{N}_2 \equiv m^j[\, T_2 \,||\, R \,] : Net$. From $\mathbf{N}_1 : Net$, by Lemma B.4(3) we get $\emptyset \vdash T_1 : Tree$ and

- either (initial) $\emptyset \vdash \mathsf{go}\ m^j.P \mid Q : Proc(h)$;
- or (ongoing) $\emptyset \vdash \mathsf{go}\ m^j.P \mid Q : Proc(i)$.

We consider the ongoing case, the proof for the initial case being the same. In this case by Lemma B.3(2) we have that $\emptyset \vdash \mathsf{go}\ m^j.P : Proc(i)$ and then by Lemma B.3(7) $\emptyset \vdash P : Proc(i)$. We conclude by applying the ongoing typing rules taking into account Proposition B.6.

Case $\mathbf{N} \equiv l^h[\, T \,||\, \mathtt{run}_p \mid Q]$ and the reduction is by rule (run):

$$l^h[\, T \,||\, \mathtt{run}_p \mid Q] \rightarrow l^h[\, T \,||\, P_1 \mid \ \ldots \ \mid P_n \mid Q]$$

where $p(T) \rightsquigarrow_{p,l^h,\Box x^h,\Box x} T, \{\{\Box P_1/\Box x\}, \ldots, \{\Box P_n/\Box x\}\}$. From $\vdash \mathbf{N} : Net$, by Lemma B.4(3) $\emptyset \ \vdash \ T : Tree$. By construction $P_k = P'_k\{l^h/\circlearrowleft, p/\bullet\}$, where $\Box P'_k$ matches $\Box x^h$ and therefore $\emptyset \ \vdash \ P'_k : \mathit{ProcLocal}(h)$ and then $\emptyset \ \vdash \ P_k : \mathit{Proc}(h)$ by Proposition B.5. We conclude by applying the ongoing typing rules taking into account Proposition B.6.

Case $\mathbf{N} \equiv l^h[\, T \,||\, \mathtt{update}_p(\chi, V).P \mid Q]$ and the reduction is by rule (update):

$$l^h[\, T \,||\, \mathtt{update}_p(\chi, V).P \mid Q] \rightarrow l^h[\, T' \,||\, P\mathtt{s}_1 \mid \ \ldots \ \mid P\mathtt{s}_n \mid Q]$$

where $p(T) \rightsquigarrow_{p,l^h,\chi,V} T', \{\mathtt{s}_1, \ldots, \mathtt{s}_n\}$. From $\vdash \ \mathbf{N} \ : \ Net$, by Lemma B.4(3) $\emptyset \ \vdash \ T : Tree$ and

- either (initial) $\emptyset \ \vdash \ \mathtt{update}_p(\chi, V).P \mid Q : \mathit{Proc}(h)$,
- or (ongoing) $\emptyset \ \vdash \ \mathtt{update}_p(\chi, V).P \mid Q : \mathit{Proc}(i)$.

We consider the ongoing case with $\chi = y^\star @ x^j$, the proof for the other cases being similar. In this case by Lemma B.7(5) $\emptyset \ \vdash \ T' : Tree$. By Lemma B.3(2) we have that $\emptyset \ \vdash \ \mathtt{update}_p(\chi, V).P : \mathit{Proc}(i)$ and then by Lemma B.3(10) or (11) $x : \mathit{Loc}(j), y : \mathit{Path}^\star \ \vdash \ P : \mathit{Proc}(i)$. By construction $\mathtt{s}_k = \{m^j/x, p'_k/y\}$, for some m^j and p'_k such that $\vdash \ p'_k : \mathit{Path}^\star$. By Lemma B.7(3) this gives $\emptyset \ \vdash \ P\mathtt{s}_k : \mathit{Proc}(i)$. We conclude by applying the ongoing typing rules for processes.

C Safety proof

Proposition 4.2 *If* $\mathbf{N}$ *is an initial network, and* $\mathbf{N} \twoheadrightarrow \vec{\nu}(l^i[\, T \,||\, P \mid Q \,] \mid \mathbf{N}')$*, and* h *is the security level of the source location of* P*, then* $\vdash P : \mathit{Proc}(h)$*.*

Proof The proof is by induction on $\twoheadrightarrow$ and by cases on the definition of source location using Generation and Substitution Lemmas.

Case $\mathbf{N} \equiv \vec{\nu}(l^i[\, T \,||\, P \mid Q \,] \mid \mathbf{N}')$. In this case $i = h$ and $\vdash \mathbf{N} : Net$ using the initial typing rules. By Lemma B.4(4), (2), (3) $\vdash P \mid Q : \mathit{Proc}(h)$ which implies $\vdash P : \mathit{Proc}(h)$ by Lemma B.3(2).

Case $\mathbf{N} \twoheadrightarrow \vec{\nu}(l^h[\, T \,||\, \mathtt{run}_p \mid Q' \,] \mid \mathbf{N}') \rightarrow \vec{\nu}(l^h[\, T \,||\, P \mid Q \,] \mid \mathbf{N}')$ since $p(T) \rightsquigarrow_{p,l^h,\Box x^h,\Box x} T, \{\{\Box R_1/\Box x\}, \ldots, \{\Box R_n/\Box x\}\}$ and $R_1 \equiv P \mid R$ and $Q \equiv R \mid R_2 \mid \ \ldots \ \mid R_n \mid Q'$. Then $p(T) \rightsquigarrow_{p,l^h,\Box x^h,\Box x} T, \{\{\Box R_1/\Box x\}, \ldots, \{\Box R_n/\Box x\}\}$ implies $\mathtt{match}(\Box R'_1, \Box x^h) = \{\Box R_1/\Box x\}$ and $R_1 \equiv R'_1\{l^h/\circlearrowleft, p_1/.\}$ for some R'_1, p_1 such that $\vdash R'_1 : \mathit{ProcLocal}(h)$ and p_1 is a path without occurrences of ".". Then $\vdash p_1 : Path$ which together with $\vdash R'_1 : \mathit{ProcLocal}(h)$ imply $\vdash R_1 : \mathit{Proc}(h)$ by Proposition B.5. So we conclude $\vdash P : \mathit{Proc}(h)$ by Lemma B.3(2).

Case $\mathbf{N} \twoheadrightarrow \vec{\nu}(l^i[\, T' \,||\, \mathtt{update}_p(y^\star @ x^j, V).P' \mid Q' \,] \mid \mathbf{N}') \rightarrow \vec{\nu}(l^i[\, T \,||\, P \mid Q \,] \mid \mathbf{N}')$ since $p(T') \rightsquigarrow_{p,l^i,y^\star @ x^j,V} T, \{\mathtt{s}_1, \ldots, \mathtt{s}_n\}$ and $P'\mathtt{s}_1 \equiv P \mid R$ and $Q \equiv R \mid P'\mathtt{s}_2 \mid \ \ldots \ |P'\mathtt{s}_n \mid Q'$. By induction we have that $\emptyset \ \vdash \ \mathtt{update}_p(\chi, V).P :$

$Proc(h)$ and then by Lemma B.3(10) or (11) $x : Loc(j), y : Path^{\star} \vdash P : Proc(h)$. By construction $\mathtt{s}_k = \{m^j/x, p'_k/y\}$, for some m^j, p'_k such that $\vdash p'_k : Path^{\star}$. By Lemma B.7(3) this gives $\emptyset \vdash P\mathtt{s}_k : Proc(h)$.

The proofs for the remaining cases are similar to the proof of the last case.