Controlling security policies in a distributed environment

Francisco Martins
Vasco Vasconcelos

DI–FCUL TR–04–1

April 2004

Departamento de Informática
Faculdade de Ciências da Universidade de Lisboa
Campo Grande, 1749–016 Lisboa
Portugal

Technical reports are available at `http://www.di.fc.ul.pt/tech-reports`. The files are stored in PDF, with the report number as filename. Alternatively, reports are available by post from the above address.

Controlling security policies in a distributed environment

Francisco Martins* Vasco Vasconcelos†

April 2004

Abstract

This paper presents a type system to control the migration of code between nodes in a concurrent distributed framework, using Dπ. We express resource policies with types and enforce them via a type system. Sites are organised hierarchically in subnetworks that share the same security policies, statically specified by a network administrator. The type system guarantees that, at runtime, there are no security policies violations.

1 Introduction

The constant advancements in hardware miniaturisation and integration, allowing for extremely portable and flexible machines, like PDAs, cellular phones, and laptop computers; the increase of communications bandwidth, and the spread of wireless communications is promoting the interaction with a broad range of services, and encouraging the sharing of our own resources. A concern that arises almost simultaneously in one's mind is how can we protect our personal data or resources from being abusively used. This paper proposes a means to control the security of resources in a mobile distributed environment.

A natural method to define security policies in a network is to identify regions of nodes sharing the same security requirements, and to specify these requirements locally for each region. Then, we may use these regions, that we name *security groups*, as building blocks to obtain larger regions, exploiting the security policies already defined. Each security group represents a kind

*Department of Mathematics, University of Azores, Portugal.
†Department of Informatics, Faculty of Sciences, University of Lisbon, Portugal.

of firewall that dictates the rules, and supervises the migration of code that crosses its border. We conceive a security model where sites may belong to more than one security group, and security groups form an hierarchical structure. We are thus able to combine security policies from various regions in a straightforward manner.

Our notion of security group corresponds to an enriched view of the notion of groups introduced for the ambient calculus [4–6]. We choose Dπ [13, 14] as the underlying calculus, and extend it with the notion of security groups. Our main motivation is to devise a type system to check the integrity and consistency of user-declared security policies, guaranteeing that well-typed networks do not violate the specified security policies.

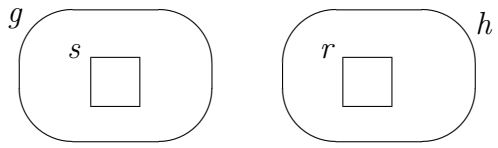
Sites form a network of computational shells where processes compete for memory, CPU cycles, and other local resources. Communication is local; therefore, the interaction between sites must be programmed explicitly via code migration. But running code from other sites opens the possibility of unauthorised use of resources. To set up secrecy—the protection of resources from unauthorised accesses—we use the notion of security groups.

The group's security officer defines a set of rules that enumerate what groups are able to perform what actions. We classify these actions as *communication actions*, *memory allocation actions*, and *code migration actions* and manage five control attributes: the installRes and useRes control attributes are used to specify what groups are able to read and to write from local channels, respectively; the createRes attribute identifies the set of groups that may create channels; and finally code migration is regulated through the tuning of acceptsCode and of go control attributes.

We allow hierarchical structures at security level but not at computational level, by considering a flat calculus (from the point of view of computational units, namely Dπ), and by hierarchically organising security policies.

In what follows we explain how security policies may be set up. We only specify positive granting privileges, that is, we consider that all actions are denied unless otherwise stated. Hence, we present a simple method to specify the security rules while avoiding contradictory policies: granting and denying the same privilege.

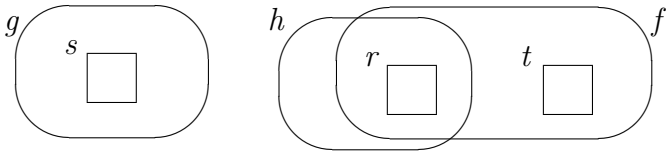
For our running example, we fix a site s belonging to a group, g, hosting a process goto $r.a!\langle\diamond\rangle$, and a site r under the security control of group h, running the process goto $s.b?(\diamond)$ stop. The first example refers to a network where groups g and h are disjoint and occur at top level, as depicted in the diagram below.



To establish the adequate security policies for the above network, allowing the code at site s and at site r to execute without violating the security rules, group h must accept code from group g (acceptsCode policy), and must allow g's code to use local resources (useRes policy); group g, in turn, must accept code from h (acceptsCode policy), and allow h's code to install resources (installRes policy). For each group, we can write down these policies using a simple notation: a pair of sets describing the security policies for the group, and its parent groups.

$$g\colon (\{\mathsf{acceptsCode}\colon h, \mathsf{installRes}\colon h\}, \emptyset)$$
$$h\colon (\{\mathsf{acceptsCode}\colon g, \mathsf{useRes}\colon g\}, \emptyset)$$

Now consider that site r is also under the security region defined by group f:



Here, site r is a member of group h and of group f simultaneously, which means that the operations performed in r are checked both against the policies specified for h and for f. Hence, h and f must *both* accept code from g and allow this code to use local resources. A possible type for f is that of h, namely:

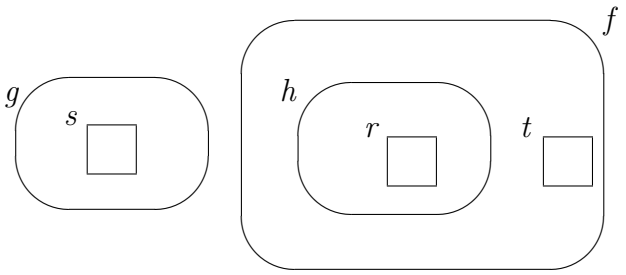
$$f\colon (\{\mathsf{acceptsCode}\colon g, \mathsf{useRes}\colon g\}, \emptyset)$$

On the other hand, the code at site r is trying to migrate code to site s. Then, group g must specifically grant privileges to group h, and to group f, since r belongs to both of them. An admissible type for g becomes

$$g\colon (\{\mathsf{acceptsCode}\colon \{f, h\}, \mathsf{installRes}\colon \{f, h\}\}, \emptyset)$$

and the type for h remains unchanged.

The notion of subgroups provides for another method to combine group policies. Consider now that group h is a subgroup of f, as depicted in the following diagram.



In order to migrate code from site s to site r, group f must accept code from group g and, furthermore, group h must (a) accept code from group g, and (b) allow this code to use local resources. The types for f and h become

$$f\colon (\{\mathsf{acceptsCode}\colon g\}, \emptyset)$$
$$h\colon (\{\mathsf{acceptsCode}\colon g, \mathsf{useRes}\colon g\}, \{f\})$$

Notice that group h is now a subgroup of f (as specified in the second component of the type for h), and that permission to use local resources is only specified at h. The idea is that each group specifies the policies for the sites that are directly under its control. When a site is under the control of a subgroup, the parent groups only concede the authority for code to cross their boundaries. The remaining policies are "delegated" to the groups where the sites directly belong to, thus avoiding the replication of policies at each group level.

Let us turn our attention to the migrating code, $\mathsf{goto}\ s.b?(\diamond)\ \mathsf{stop}$, running at site r. Site r is a member of group h, which, in turn, is a subgroup of f. So, r may be seen as a member of h, *or* as a member of f. Hence, group g may specify security policies addressed specifically at group h or at group f. Suppose that we want to express the facts that group g accepts code from group f, and allows this code to use its resources, but only code from group h may install resources. We could set up group g policies as

$$g\colon (\{\mathsf{acceptsCode}\colon f, \mathsf{useRes}\colon f, \mathsf{installRes}\colon h\}, \emptyset)$$

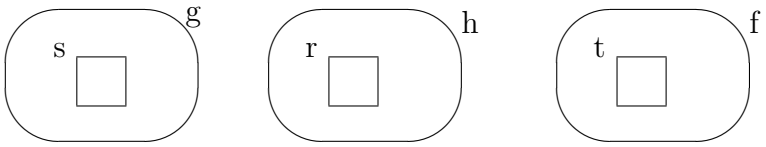
In addition, we may want to be more specific and accept code only from group h (thus denying code from site t). So, we could write

$$g\colon (\{\mathsf{acceptsCode}\colon h, \mathsf{useRes}\colon f, \mathsf{installRes}\colon h\}, \emptyset)$$

Letting a site be a member of more than one group, and having groups organised hierarchically, allows conjunctive *and* disjunctive security combination, respectively. Furthermore, group hierarchy allows for the inheritance of security policies granted to the parent groups. The following table summarises the composition of security policies.

Security composition	Conjunction	Disjunction
Organisation	Sites belong to more than one group	Groups defined within groups

Our last example addresses the go policy. Consider the network depicted in the following diagram



and suppose that the policies for group h, and for group f are

$$h\colon (\{\mathsf{acceptsCode}\colon g, \mathsf{useRes}\colon g\}, \emptyset)$$
$$f\colon (\{\mathsf{acceptsCode}\colon h, \mathsf{useRes}\colon h\}, \emptyset)$$

Processes $\mathsf{goto}\ r.a!\langle\diamond\rangle$ and $\mathsf{goto}\ t.a!\langle\diamond\rangle$, running at sites s and r, respectively, do not violate the security rules, whereas $\mathsf{goto}\ t.a!\langle\diamond\rangle$, running at site s, violates the security rules at group f. What about process $\mathsf{goto}\ r.\mathsf{goto}\ t.a!\langle\diamond\rangle$ running at site s? We interpret migration as two disjoint activities: the exiting of code from the source site, and the arriving of that code at the destination site. The reason to clarify the migration action, exploiting two distinct events, is related to the non-transitivity nature of trust. It is undesirable that a site uses another as a proxy to get access to resources at a third party that were not deliberately made available to it. In fact, sites run accepted code indistinctly from its own code. A security leak may arise when code arriving at a given site tries to migrate to a site belonging to a different group. This third site runs the piece of code without knowing its original location. The side effect of using a site as a proxy is highly undesirable and is able to turn the security model useless, because by accepting code from a third party, the group is giving this party the power to use the site as a proxy to other actions.

The go policy fills the gap and provides the security officer with a control mechanism to allow incoming code from a specific group but denying (or not) the code to migrate to a third group. It is up to the security officer of group h to define which behaviour should be implemented. The security policy for h, as set as above, does not allow migration of code from s to r and then to t. However, would that be desirable, the security officer may define that the sites at group h may be used as proxies for sites at group g, and allow incoming code to migrate outside. The type of h, would then become

$$h\colon (\{\mathsf{acceptsCode}\colon g, \mathsf{go}\colon g\}, \emptyset)$$

$v ::=$	*Values*	$n ::=$	*Names*
$a_{@}s$	located channel	a, b, c, x	channels
$\mid \diamond$	basic value	$\mid r, s, t, y$	sites
		$\mid f, g, h$	groups

$P, Q ::=$	*Processes*	$N, M ::=$	*Networks*
stop	termination	stop	termination
$\mid (\boldsymbol{\nu} n\colon T)\ P$	restriction	$\mid (\boldsymbol{\nu} n\colon T)\ N$	restriction
$\mid P \mid Q$	composition	$\mid N \mid M$	composition
$\mid$ goto $s.P$	migration	$\mid s[P]$	site
$\mid a!\langle v\rangle$	output		
$\mid a?(v)\ P$	input		
$\mid a?*(v)\ P$	replication		

(see figure 2 for the syntax of types T)

Figure 1: Syntax of Dπ.

Outline. The next section briefly introduces the Dπ syntax, and its operational semantics. Section 3 introduces our approach to the checking security policies, and presents the type assignment system. Section 4 is devoted to type safety; the result is achieved via a tagged version of language that we use to express runtime errors. Section 5 discusses the relationship between our approach and the Mikado core model.he last section presents the related work, and states our conclusions.

2 Dπ syntax and operational semantics

This section deals with the syntax, and the operational semantics of Dπ, mainly taken from Hennessy and Riely [13, 14].

2.1 Calculus

The syntax of the calculus is defined in figure 1. The main difference from Dπ concerns the usage of groups, and therefore we introduce a new constructor to create groups. We consider a monadic version of the calculus where only

located names can be passed around, since our main focus is the control of migration, not communication.

We briefly address the Dπ syntax; the interested reader should refer to [13, 14] for motivations and details. The calculus presents two main syntactic categories: processes and networks. At the processes level, besides the usual π-calculus constructs, there is also the ability to spawn a process into a specific location: the goto $s.P$ construct migrates process P to location s, for execution. Networks are made up from processes running at named locations that we call sites, $s[P]$. A site is basically a named place where resources live and computations occur.

The security policies are expressed by types, and discussed below.

2.2 Types

The syntax of types is depicted in figure 2. We assign types to channels, to sites, and to groups.

Channels can carry other channels, as well as basic values. The type for channel values assumes the form $C_{@}G$, where C is the type of the channels that can be carried, and G is the set of groups hosting the communicated channels. The unit value, $\diamond$, is typed with the type constant unit. Channel types trace the type of the values that are communicated along the channel, as well as its usage (input, output, or both), following Sangiorgi and Pierce [19]. The subtype relation that characterises channel tags is introduced in figure 5.

Site types simply record the set of groups to which the site belongs.

Group types are a central notion in our work. It is at group level that we record information for security. This information includes (a) the set of rules that govern the interaction with the network, and (b) the set of the parent groups (supergroups) of the group. A rule defines, for a given policy, the set of groups that are allowed to execute the action it protects. Instead of a set of groups, we may also specify $\star$ as a wild card, meaning that any group is allowed to perform the specified action. This definition is meant to be used for actions opened to "everyone in the world" and since it allows a great exposure of the sites protected under these kind of rules, it should be used carefully. We conjecture that if the group wild card is used for every policy of every group, we would regain the original Dπ.

We use a type and effect system where the actions that have a security impact are recorded and checked by the type system. The effects match the names of the policies. So, the installRes and useRes effects specify that a process performs input or output actions, respectively. When a process creates a new channel it is marked with the createRes. We divide migration

$T ::=$	*Types*		$L ::=$	*Name types*
L	local type		C	local channel
$\mid L@s$	global type		$\mid G$	site type
			$\mid (R, G)$	group type
$C ::=$	*Local channel types*			
$\langle V \rangle^I$	local channel		G	*set of groups*
$R ::=$	*Security rules*		$V ::=$	*Value types*
$\{\pi_1 : S_1, \ldots, \pi_n : S_n\}$			$C@G$	channel
			$\mid$ unit	basic type
$I ::=$	*Tags*			
r	input		$S ::=$	*Sources*
$\mid$ w	output		G	set of groups
$\mid$ rw	in/out		$\mid \star$	any group
$\pi ::=$	*Policies*		$\tau ::=$	*Effects*
τ	effects		useRes	output
$\mid$ acceptsCode	code reception		$\mid$ installRes	input
			$\mid$ createRes	ch. creation
			$\mid$ go	code sending

Figure 2: Syntax of types.

into two events: the exiting of the source site, identified by go, and the arriving at the destination host, described by the acceptsCode (*cf.* mikado core model [1]).

Types may be local or global: local types are used when creating names at a given site; global (or located types) are assigned to names when declared at network level as well as in typings (see section 3.3 on page 13 for details about the typings and typing rules).

The set A records the actions performed by processes. The type system keeps track of the actions performed by a process, and checks the security issues at migration action, that is, at goto operation.

1. $((N \mid M) \mid M') \equiv (N \mid (M \mid M'))$
   $(M \mid N) \equiv (N \mid M)$
   $(N \mid \mathsf{stop}) \equiv N$
2. $(\boldsymbol{\nu} n\colon T)\ N \mid M \equiv (\boldsymbol{\nu} n\colon T)\ (N \mid M)$ if $n \notin \mathrm{fn}(M)$
   $(\boldsymbol{\nu} n\colon T)\ (\boldsymbol{\nu} m\colon T')\ N \equiv (\boldsymbol{\nu} m\colon T')\ (\boldsymbol{\nu} n\colon T)\ N$ if m not in T, n not in T', and $n \neq m$
   $(\boldsymbol{\nu} n\colon L_{@}s)\ s[P] \equiv s[(\boldsymbol{\nu} n\colon L)\ P]$ if $n \neq s$
3. $s[P] \mid s[Q] \equiv s[P \mid Q]$
4. $(\boldsymbol{\nu} n\colon T)\ \mathsf{stop} \equiv \mathsf{stop}$
   $(\boldsymbol{\nu} s\colon T)\ s[\mathsf{stop}] \equiv \mathsf{stop}$

Figure 3: Structural congruence.

2.3 Operational semantics

Terms are taken up to α-congruence in such a way that bound names are different from free names, and different from each other. The binders of the calculus are the usual in π-calculis languages: name n is bound in $(\boldsymbol{\nu} n\colon T)\ P$, and in $(\boldsymbol{\nu} n\colon T)\ N$, whereas x and y are both bound in $a?(x_{@}y)\ P$. We define the operational semantics on top of a congruence relation, again as usual in the π-calculi languages. The *structural congruence relation*, $\equiv$, is the least congruence relation closed under the rules presented in figure 3. It closely follows the structural congruence relation introduced for Dπ [13, 14], apart from rule $(\boldsymbol{\nu} n\colon L_{@}s)\ s[P] \equiv s[(\boldsymbol{\nu} n\colon L)\ P]$ where the scope of a name declared at network level may only be restricted to the site it was created at. From a security point of view it is important to identify the site that creates a name in order to check the corresponding security event. This congruence rule would easily break down our subject reduction result: consider a site t with permission to create a name in site s but not in site r. For example, in Dπ, $s[(\boldsymbol{\nu} a\colon C)\ Q] \mid r[R] \equiv s[Q] \mid r[(\boldsymbol{\nu} a\colon C)\ R]$, if $a \notin \mathrm{fn}(Q) \cup \mathrm{fn}(R)$. Consider the network $t[\mathsf{goto}\ s.(\boldsymbol{\nu} a\colon C)\ P] \mid s[Q] \mid r[R]$, and suppose that site s allows site t to create resources. This network might reduce to $t[\mathsf{stop}] \mid s[Q] \mid r[R \mid (\boldsymbol{\nu} a\colon C)\ P]$ that would result in a security fault if site r does not allow site t to allocate memory.

Reduction is defined inductively by the rules in figure 4, taken from Dπ, except for minor syntactic adjustments to incorporate groups.

$$s[a!\langle b@r\rangle] \mid s[a?(x@y)\ P] \to s[P\{r/y\}\{b/x\}] \qquad (\textsc{Comc}_1)$$

$$s[a!\langle\diamond\rangle] \mid s[a?(\diamond)\ P] \to s[P] \qquad (\textsc{Comc}_2)$$

$$s[a!\langle b@r\rangle] \mid s[a?*(x@y)\ P] \to s[P\{r/y\}\{b/x\}] \mid s[a?*(x@y)\ P] \qquad (\textsc{Comr}_1)$$

$$s[a!\langle\diamond\rangle] \mid s[a?*(\diamond)\ P] \to s[P] \mid s[a?*(\diamond)\ P] \qquad (\textsc{Comr}_2)$$

$$s[\mathsf{goto}\ r.P] \to r[P] \qquad (\textsc{Mig})$$

$$\frac{N \to M}{(\boldsymbol{\nu} n\colon L@s)\ N \to (\boldsymbol{\nu} n\colon L@s)\ M} \qquad (\textsc{Res})$$

$$\frac{N \to N'}{N \mid M \to N' \mid M} \qquad (\textsc{Par})$$

$$\frac{N \equiv N' \qquad N' \to M' \qquad M' \equiv M}{N \to M} \qquad (\textsc{Str})$$

Figure 4: Reduction rules.

3 Typing system

In this section we define a type and effect system that checks whether networks respect the security policies specified for the security groups. The type system is based on a subtype relation à lá Sangiorgi and Pierce [19], and is parametric w.r.t. two functions that are used to check the security policies, namely, the allows, and the canEnter functions. We provide a subsection for the discussion of each of these topics: subtyping, checking security policies, and the presentation of the type system, and conclude with a subject-reduction theorem.

3.1 Subtyping

The *subtyping relation*, $<:$, is defined as the least binary relation on types that satisfies the rules in figure 5, where channels are tagged according to their usage: input, output, and input/output (r, w, and rw, respectively). We extend the subtyping relation to deal with types involving groups. The original intuitions remain unchanged, namely that the subtyping relation is covariant for inputs, contravariant for outputs, and invariant if the channels are used both for input and for output purposes. The subtyping rules are

Value subtyping

$$\mathsf{unit} <: \mathsf{unit} \qquad \frac{C_1 <: C_2 \quad G_1 \subseteq G_2}{C_1{@}G_1 <: C_2{@}G_2}$$

Local channel subtyping

$$\frac{i = \mathsf{r}, \mathsf{rw} \quad V_1 <: V_2}{\langle V_1 \rangle^i <: \langle V_2 \rangle^{\mathsf{r}}} \qquad \frac{i = \mathsf{w}, \mathsf{rw} \quad V_2 <: V_1}{\langle V_1 \rangle^i <: \langle V_2 \rangle^{\mathsf{w}}}$$

$$\frac{V_1 <: V_2 \quad V_2 <: V_1}{\langle V_1 \rangle^{\mathsf{rw}} <: \langle V_2 \rangle^{\mathsf{rw}}}$$

Global channel subtyping

$$\frac{C_1 <: C_2}{C_1{@}s <: C_2{@}s}$$

Figure 5: Subtyping relation.

straightforward. Notice the set inclusion to handle groups in value subtyping, and the last subtyping rule to relate located channels. In this work we do not consider subtyping for site types, or for group types; it would complicate the theory and it is not clear the benefits of such an inclusion.

3.2 Checking security policies

A *typing* Γ is a partial function of finite domain from channel names to local channel types $(a\colon C)$, from site names to sources $(s\colon S)$, and from group names to pairs of security rules, and parent groups $(g\colon (R, G))$. We write $\mathrm{dom}(\Gamma)$ for the domain of Γ. When $x \notin \mathrm{dom}(\Gamma)$, we write $\Gamma, x : T$ for the type environment Γ' such that $\mathrm{dom}(\Gamma') = \mathrm{dom}(\Gamma) \cup \{x\}$, $\Gamma'(x) = T$, and $\Gamma'(y) = \Gamma(y)$ for $y \neq x$.

The idea behind our type system is the following: at process level, we collect the effects of the actions performed by processes: either useRes, installRes, createRes, or go, standing for *input*, *output*, *channel creation*, and *migration* actions, respectively. (For instance, the process, $a!\langle b{@}r\rangle$, running at site s, has the effect useRes at site s.) Then, following to a goto action—when a source site launches code at a destination site—we check whether the source site has the right privileges to perform the intended action, in the present case an *output*. At network level there is nothing to be checked, since there is no computation taking place at that level. Also, we do not check code

$$\frac{\Gamma(g) = (R, G) \qquad f \in R(\pi)}{\Gamma \vdash g \text{ allows } f \colon \pi} \qquad \frac{\Gamma(g) = (R, G) \qquad R(\pi) = \star}{\Gamma \vdash g \text{ allows } f \colon \pi}$$

$$\frac{\Gamma(f) = (R, \{h\} \cup G) \qquad \Gamma \vdash g \text{ allows } h \colon \pi}{\Gamma \vdash g \text{ allows } f \colon \pi}$$

$$\frac{\Gamma \vdash g \text{ allows } f \colon \pi \qquad \forall g \in G, \forall f \in F, \forall \pi \in A}{\Gamma \vdash G \text{ allows } F \colon A}$$

$$\frac{\Gamma, s \colon G, r \colon F \vdash G \text{ allows } F \colon A}{\Gamma, s \colon G, r \colon F \vdash s \text{ allows } r \colon A} \qquad \Gamma \vdash s \text{ allows } s \colon A$$

Figure 6: allows relation.

running at its host site—code that is not in the continuation part of a goto process—, since we assume that there is no need to grant specific privileges to code in such circumstances.

The typing of processes is parametrised by two functions—allows and canEnter—that encapsulate the security checking details, defined in figures 6, and 7, respectively.

Function allows checks whether code from a group is able to perform an action at a given group; a formula g allows $f : \pi$ means that group g allows group f to perform action π. A formula g allows $\star : \pi$ means that group g allows sites from any group to execute action π. When the target group, f, is part of an hierarchy of groups, the allows function succeeds if some group in the hierarchy has permission to perform the intended action. The reflexivity of the function plays an important role when dealing with the tagged version of the language (refer to section 4 for details).

Function canEnter, defined in figure 7, checks whether code from a given group has permission to enter a target group. A formula g canEnter f means that group f accepts code from group g. This privilege is controlled by the acceptsCode policy. A group g is able to enter the frontier of group f, if there exists a path through the hierarchy, granting at each group in the path the acceptsCode right to the source site.

We consider functions allows and canEnter separately because of their distinct nature: allows checks if at least one group in the group hierarchy admits a certain action, whereas canEnter checks if there exists a path in the group's hierarchy granting the acceptsCode right.

$$\frac{\Gamma(f) = (R, \emptyset) \qquad \Gamma \vdash f \text{ allows } g\colon \text{acceptsCode}}{\Gamma \vdash g \text{ canEnter } f}$$

$$\frac{\Gamma(f) = (R, \{h\} \cup G) \qquad \Gamma \vdash g \text{ canEnter } h \qquad \Gamma \vdash f \text{ allows } g\colon \text{acceptsCode}}{\Gamma \vdash g \text{ canEnter } f}$$

$$\frac{\Gamma \vdash g \text{ canEnter } f \qquad \forall g \in G, \forall f \in F}{\Gamma \vdash G \text{ canEnter } F}$$

$$\frac{\Gamma, s\colon G, r\colon H \vdash G \text{ canEnter } H}{\Gamma, s\colon G, r\colon H \vdash s \text{ canEnter } r} \qquad \Gamma \vdash s \text{ canEnter } s$$

Figure 7: canEnter relation.

$$\text{E-Unit} \quad \diamond\colon \text{unit} \vdash \text{env} \qquad \text{E-Channel} \ \frac{\Gamma \vdash \text{env}}{\Gamma, a\colon C@s \vdash \text{env}}$$

$$\text{E-Site} \ \frac{\Gamma \vdash \text{env}}{\Gamma, s\colon G \vdash \text{env}} \qquad \text{E-Group} \ \frac{\Gamma \vdash \text{env} \qquad G \subseteq \text{dom}(\Gamma)}{\Gamma, g\colon (R, G) \vdash \text{env}}$$

Figure 8: Well-formed environments.

3.3 Typing networks

The type systems for environments, processes, and networks are presented in figures 8, 9, and 10, respectively, and include three kinds of judgements: (a) judgements for well-formed environments, of the form $\Gamma \vdash \text{env}$, asserting that Γ is a well-formed environment; (b) judgements for typing processes, $\Gamma \vdash_s P\colon A$, meaning that process P, running at site s, has the effects enumerated in set A, and is well typed under typing assumption Γ; and (c) judgements for typing networks, $\Gamma \vdash N$, denoting that network N is well typed under the Γ typing assumption.

Well-formed environments rules, figure 8, guarantee that group structures are not circular. Therefore, rule E-Group ensures that when we enlarge a typing with a new group definition, the parent groups of that new group are already in the typing. The remaining rules in figure 8 are simple and need no further comments.

Next we comment on the typing rules for processes (figure 9). Rule P-Outb is concerned with the typing of an output process that carries unit values. The rule enforces that typing Γ is well formed, and that channel a is an output, or a read-write channel located at the site where the process is

$$\textsc{P-Outb}\ \frac{\Gamma \vdash \text{env} \qquad \Gamma(a) <: \langle \mathsf{unit} \rangle^{\mathsf{w}}{}_{@} s}{\Gamma \vdash_s a!\langle \diamond \rangle : \{\mathsf{useRes}\}}$$

$$\textsc{P-Outc}\ \frac{\Gamma \vdash \text{env} \qquad \Gamma(a) <: \langle C_{@} G \rangle^{\mathsf{w}}{}_{@} s \qquad \Gamma(r) = G \qquad \Gamma(b) = C_{@} r}{\Gamma \vdash_s a!\langle b_{@} r \rangle : \{\mathsf{useRes}\}}$$

$$\textsc{P-Inpb}\ \frac{\Gamma \vdash_s P : A \qquad \Gamma(a) <: \langle \mathsf{unit} \rangle^{\mathsf{r}}{}_{@} s}{\Gamma \vdash_s a?(\diamond)\ P : A \cup \{\mathsf{installRes}\}}$$

$$\textsc{P-Inpc}\ \frac{\Gamma, x : C_{@} y, y : G \vdash_s P : A \qquad \Gamma(a) <: \langle C_{@} G \rangle^{\mathsf{r}}{}_{@} s \qquad y \text{ not in } \Gamma}{\Gamma \vdash_s a?(x_{@} y)\ P : A \cup \{\mathsf{installRes}\}}$$

$$\textsc{P-Inpr}\ \frac{\Gamma \vdash_s a?(v)\ P : A}{\Gamma \vdash_s a?*(v)\ P : A} \qquad \textsc{P-Par}\ \frac{\Gamma \vdash_s P : A_1 \qquad \Gamma \vdash_s Q : A_2}{\Gamma \vdash_s P \mid Q : A_1 \cup A_2}$$

$$\textsc{P-Ress}\ \frac{\Gamma, r : G_{@} s \vdash_s P : A \qquad r \text{ not in } \Gamma}{\Gamma \vdash_s (\boldsymbol{\nu} r : G)\ P : A}$$

$$\textsc{P-Resc}\ \frac{\Gamma, a : C_{@} s \vdash_s P : A}{\Gamma \vdash_s (\boldsymbol{\nu} a : C)\ P : A \cup \{\mathsf{createRes}\}}$$

$$\textsc{P-Resg}\ \frac{\Gamma, g : (R, G)_{@} s \vdash_s P : A \qquad g \text{ not in } \Gamma}{\Gamma \vdash_s (\boldsymbol{\nu} g : (R, G))\ P : A} \qquad \textsc{P-Nil}\ \frac{\Gamma \vdash \text{env}}{\Gamma \vdash_s \mathsf{stop}}$$

$$\textsc{P-Mig}\ \frac{\Gamma \vdash_r P : A \qquad \Gamma \vdash r\ \mathsf{allows}\ s : A \qquad \Gamma \vdash s\ \mathsf{canEnter}\ r}{\Gamma \vdash_s \mathsf{goto}\ r.P : \{\mathsf{go}\}}$$

Figure 9: Typing processes.

running, and capable of carrying unit values. Rule P-Outc types an output process that carries another channel. The difference to P-Outb regards the type for channel a, that now must carry channels of the type of b, located at the same set of groups as site r. To type an input process, $a?(x_{@} y)\ P$, channel a must be an input, or a read-write channel. The continuation process, P, must be well typed in a typing augmented with x and y. Notice that channel x is located at y, and that y is defined as a site, member of the groups that channel a can carry. Hence, we guarantee that the privileges for the actions involving x and y are correctly checked, since we verify policies against all groups in G. The subtyping rule is covariant for input, which means that, if the type of channel a is a subtype of $\langle C_{@} G \rangle^{\mathsf{r}}{}_{@} s$, then a carries channels

located at a subset of G. We remain at the safe side because we are able to enforce the security policies even for a larger set. Finally, the effect of the input action—installRes—is appended to the set of actions.

The rules for typing the input of unit type values (P-Inpb), and for typing replicated inputs (P-Inpr) follow a similar pattern to the rule just described. The parallel composition of processes, $P \mid Q$, combines the set of actions gathered when typing the individual processes.

We split name restriction over three rules, P-Ress, P-Resc, and P-Resg, since channel creation, understood as resource allocation (memory consumption), deserves a special treatment. The rules are straightforward. Notice that we mark channel creation, rule P-Resc, with the createRes effect.

We could think of also controlling the creation of sites and groups. However, this does not interfere with the policies established for existing groups. In the creation of a site, we associate it with an existing group hierarchy, which means that we are simply creating another computational area that is regulated by security policies already defined. Whep we create a group, we could define new policies, but the interaction with existing groups is to some extent limited, because the policies defined for the actual groups can not talk about the new group, unless, indirectly, using the any group wild-card ($\star$). But in that case, sites are expecting to communicate with "everyone in the world" and must be prepared for such an exposure.

The inaction process, stop, is well-typed under any well-formed environment.

It is at code migration, goto $r.P$, that all the security checking takes place. When we reach a goto process we have all the information necessary to check security polices, namely, we know the site where the process lives (annotated under the turnstile)—the source site—, the site where we want to send code (indicated in the syntax of the goto process)—the target site—, as well as the actions performed by process P, the set A in the typing for P. Therefore, the typing of a goto process, checks whether the continuation process is well typed, and ensures that the target site allows the source site to perform the actions of the continuation process. We mark code migration with the go effect. The actions that P performs are not double checked at outermost levels. Nevertheless, we signal that the process is (possibly) sending code to a different site, which means that, it might be using this site as a proxy. The go action is checked when another goto process is found.

Network typing is described in figure 10. Security policies are not checked at network level, since no computation take place at this level. Therefore, the only interesting fact to stress is that when code is installed at a certain site, the actions that are not the continuation of a goto process are not checked (rule N-Site). Indeed, since functions allows and canEnter are reflexive, there

$$\textsc{N-Site}\ \frac{\Gamma \vdash_s P\colon A}{\Gamma \vdash s[P]} \qquad \textsc{N-Res}\ \frac{\Gamma, n\colon L@s \vdash N \quad n \text{ not in } \Gamma}{\Gamma \vdash (\boldsymbol{\nu} n\colon L@s)\ N}$$

$$\textsc{N-Par}\ \frac{\Gamma \vdash N \quad \Gamma \vdash M}{\Gamma \vdash N \mid M} \qquad \textsc{N-Nil}\ \frac{\Gamma \vdash \text{env}}{\Gamma \vdash \mathsf{stop}}$$

Figure 10: Typing networks.

is no point in checking policies at this level.

Next we enunciate the main result of this section, namely, that typings are preserved by reduction.

Theorem 1 (Subject Reduction). *If* $\Gamma \vdash N$*, and* $N \to M$*, then* $\Gamma \vdash M$*.*

Proof. First we establish a similar result for structural equivalence. Then we proceed by induction on the typing of $\Gamma \vdash N$, analysing every case in the reduction relation. See the appendix for details. □

4 Tagged language

In order to express type safety, we introduce a tagged version of the language. The need to decorate the language materialises when formally writing down what we consider a runtime error. The idea is that, to express runtime errors, we need to reason about the actions when they really happen and not just to talk about the effects of them *à posteriori* when we encounter a goto process. At that point it is too late to figure out the causes for the security fault (at least from a syntactic point of view). For instance, consider an output process, $a!\langle b@s\rangle$, running at site r; we need to explicit the circumstances that cause it to disregard the security rules established. We identify two possible causes: (a) the process is not allowed to output at the current site, because it might have come from a site belonging to a group that has no right to output at the present group, or (b) channel a is a channel with no output capabilities.

With the language introduced in figure 1, it is not possible to express these facts syntacticly, because we lack information about the site that triggered the action. Therefore, we slightly adjust the syntax and, consequently, the structural congruence, and the type system as well, to represent runtime errors and to allow for a type safety result.

This section presents the tagged version of the language, and discusses the changes in the syntax and the respective consequences in the operational

Syntax (all rules from figure 1, replacing site *and* restriction *by the following rules)*

$$s[P]^r_\Gamma \qquad\qquad (\boldsymbol{\nu}_t\, n\colon T)\ N$$

Structural congruence (group 1. from figure 3, plus the following rules)

2.	$(\boldsymbol{\nu}_t\, n\colon T)\ N \mid M \equiv_T (\boldsymbol{\nu}_t\, n\colon T)\ (N \mid M)$	if $n \notin \mathrm{fn}(M)$
	$(\boldsymbol{\nu}_t\, n\colon T)\ (\boldsymbol{\nu}_r\, m\colon T')\ N \equiv_T (\boldsymbol{\nu}_r\, m\colon T')\ (\boldsymbol{\nu}_t\, n\colon T)\ N$	if m not in $T \cup \{t\}$, n not in $T' \cup \{r\}$, and $n \neq m$
	$(\boldsymbol{\nu}_t\, n\colon L@s)\ s[P]^t_{\Gamma \sqcap n\colon L@s} \equiv_T s[(\boldsymbol{\nu} n\colon L)\ P]^t_\Gamma$	if $n \notin \mathrm{dom}(\Gamma) \cup \{s\}$
3.	$s[P]^t_\Gamma \mid s[Q]^t_\Gamma \equiv_T s[P \mid Q]^t_\Gamma$	
4.	$(\boldsymbol{\nu}_t\, n\colon T)\ \mathsf{stop} \equiv_T \mathsf{stop}$	
	$(\boldsymbol{\nu}_t\, s\colon T)\ s[\mathsf{stop}]^t_\Gamma \equiv_T \mathsf{stop}$	

Figure 11: The tagged language—syntax and structural congruence.

semantics; defines a tag function to relate the two versions of the language, and shows that the reduction relations for both languages mimic each other. The type system we present for the tagged language preserves types during reduction. Finally, we define the notion of runtime errors and prove our type safety result.

4.1 Syntax and operational semantics

On what concerns syntax, we need to know what site (if any) sent the code, and what policies are established for both sites involved: the source for the code, and the site running the code. Hence, we change the syntactic category for networks (figure 11—syntax): (a) we decorate the site constructor with the set of assumptions needed to check security policies for the processes it runs (in fact these assumptions are enough to type the process hosted by the site), and the name of the site that sent a given piece of code; (b) we add to name restriction the information of the site that creates the name, since, by scope extrusion, a name may appear at network level without indication of its creation site. Keep in mind that, in the tagged language, we need to express conditions about actions as they occur and, since no computation take place at network level, we require the information about the site where the name was created. All the remaining syntax is left unchanged.

Meet operator for channel types

$$
\begin{array}{rll}
\langle \mathsf{unit} \rangle^{I} \sqcap \langle \mathsf{unit} \rangle^{I'} = \langle \mathsf{unit} \rangle^{I} & & \text{if } I <: I' \\
\langle V \rangle^{I} \sqcap \langle V \rangle^{I} = \langle V \rangle^{I} & & \\
\langle C @ H \rangle^{\mathsf{rw}} \sqcap \langle C' @ H' \rangle^{\mathsf{w}} = \langle C @ H \rangle^{\mathsf{rw}} & & \text{if } C' <: C \text{ and } H' \subseteq H \\
\langle C @ H \rangle^{\mathsf{rw}} \sqcap \langle C' @ H' \rangle^{\mathsf{r}} = \langle C @ H \rangle^{\mathsf{rw}} & & \text{if } C <: C' \text{ and } H \subseteq H' \\
\langle C @ H \rangle^{\mathsf{w}} \sqcap \langle C' @ H' \rangle^{\mathsf{r}} = \langle C @ H \rangle^{\mathsf{rw}} & & \text{if } C <: C' \text{ and } H \subseteq H' \\
\langle C @ H \rangle^{\mathsf{w}} \sqcap \langle C' @ H' \rangle^{\mathsf{w}} = \langle (C \sqcap C') @ (H \cup H') \rangle^{\mathsf{w}} & & \\
\langle C @ H \rangle^{\mathsf{r}} \sqcap \langle C' @ H' \rangle^{\mathsf{r}} = \langle (C \sqcap C') @ (H \sqcap H') \rangle^{\mathsf{r}} & &
\end{array}
$$

Meet operator for site types

$$G \sqcap G' = G \cap G' \qquad \text{if } G \cap G' \neq \emptyset$$

Meet operator for located channels

$$\langle V \rangle^{I} @ r \sqcap \langle V' \rangle^{I'} @ r = (\langle I \rangle V \sqcap \langle I' \rangle V') @ r$$

Figure 12: The meet operator.

We now comment the rearrangements in the structural congruence rules, figure 11—structural congruence. In group 2, 3rd rule, the transfer of name creation from site to network level records the site where the name was created. Notice also that the set of assumptions at the left-hand side of the congruence operator enlarge with the name declared at network level, announcing the creation of the name; the meet operator (cf. [13, 14]) is used to combine the type of the new name with that in type assumption, is defined in figure 12, and the idea is we compute the greatest lower bound of the two types. The remaining scope extrusion rules just reflect the syntactic adjustments.

In group 3, merging sites is only possible when the tagged information agrees, meaning that is it not possible to merge sites that execute code coming from different locations, or that are governed by distinct security policies. In group 4, the rules regarding garbage collection underwent changes provoked by syntax modifications.

Reduction in the tagged language differs from the one introduced in figure 4, mainly on what concerns communication and code migration. Communication may occur under dissimilar views of the security policies of a site, in particular, the input process may not use or even have knowledge of the value

$$s[a!\langle b@r\rangle]^t_{\Gamma,r\colon G,b\colon T} \mid s[a?(x@y)\ P]^u_\Delta \mapsto s[P\{r/y\}\{b/x\}]^u_{\Delta\sqcap r\colon G\sqcap b\colon T} \qquad \text{(T-Comc}_1\text{)}$$

$$s[a!\langle\diamond\rangle]^t_\Gamma \mid s[a?(\diamond)\ P]^u_\Delta \mapsto s[P]^u_\Delta \qquad \text{(T-Comc}_2\text{)}$$

$$s[a!\langle b@r\rangle]^t_{\Gamma,r\colon G,b\colon T} \mid s[a?*(x@y)\ P]^u_\Delta \mapsto s[P\{r/y\}\{b/x\}]^u_{\Delta\sqcap r\colon G\sqcap b\colon T} \mid s[a?*(x@y)\ P]^u_\Delta \qquad \text{(T-Comr}_1\text{)}$$

$$s[a!\langle\diamond\rangle]^t_\Gamma \mid s[a?*(\diamond)\ P]^u_\Delta \mapsto s[P]^u_\Delta \mid s[a?*(\diamond)\ P]^u_\Delta \qquad \text{(T-Comr}_2\text{)}$$

$$s[\mathsf{goto}\ r.P]^t_\Gamma \mapsto r[P]^s_\Gamma \qquad \text{(T-Mig)}$$

$$\frac{N \mapsto M}{(\boldsymbol{\nu}_t\, n\colon T)\ N \mapsto (\boldsymbol{\nu}_t\, n\colon T)\ M} \qquad \text{(T-Res)}$$

(*plus rules* Par, *and* Str *from figure 4*)

Figure 13: The tagged language—reduction.

it is going to receive (besides its type). Therefore, communication updates the typing assumptions of the receiving process with information from the emitting one. Rule T-Comc$_1$ updates the resulting process with type information of the communicated values, channel b and site r, from the output part of the communication. When channel b or site r is not mentioned in process P, the type information is just appended to typing Δ, but if one of them (or both) is already referred in Δ, then we compute the least common supertype of the type figuring in Δ and the one communicated. The subject reduction theorem (theorem 4), also guarantees that, for well-typed processes, the meet operation, $\sqcap$, is always defined.

The persistent receptor, ruled by T-Comr$_1$, follows the same idea; the type assumptions for the persistent part remain constant, whereas the communicated values enlarge the type assumptions of the replica as for T-Comc$_1$. The rules governing the communication of *nil* values, rule T-Comc$_1$, and rule T-Comr$_2$, are closer to their counterpart in the untagged version, since no values are transmitted.

When code migrates, we inscribe the information about the site that sends the code and forget the site recorded previously, rule T-Mig. The information about the site that sent the code is fundamental to reason about security—it is the source site. In our model, we check the security policies between a source and a destination site alone, and do not keep history information on visited sites. Rule T-Res results from the syntax update, and needs no

$$
\begin{aligned}
\mathsf{tag}_\Gamma(\mathsf{stop}) &= \{\emptyset\} \\
\mathsf{tag}_\Gamma(N_1 \mid N_2) &= \{M_1 \mid M_2 \text{ s.t. } M_i \in \mathsf{tag}_\Gamma(N_i)\} \\
\mathsf{tag}_\Gamma(s[P]) &= \{s[P]^t_\Delta \text{ s.t. } \Delta \vdash_s P\colon A,\ \ \Gamma <\colon \Delta,\ \ \Delta \vdash s \ \mathsf{allows}\ t\colon A, \\
&\qquad\qquad \Delta \vdash t \ \mathsf{canEnter}\ s\} \\
\mathsf{tag}_\Gamma((\boldsymbol{\nu} a\colon C@s)\ N) &= \{(\boldsymbol{\nu}_t\, a\colon C@s)\ M \text{ s.t. } M \in \mathsf{tag}_{\Gamma,\mathsf{a}\colon \mathsf{C@s}}(N) \\
&\qquad\qquad \text{and } \Gamma \vdash s \ \mathsf{allows}\ t\colon \mathsf{createRes}\} \\
\mathsf{tag}_\Gamma((\boldsymbol{\nu} s\colon G)\ N) &= \{(\boldsymbol{\nu}_t\, s\colon G)\ M \text{ s.t. } M \in \mathsf{tag}_{\Gamma,\mathsf{s}\colon \mathsf{G@s}}(N)\} \\
\mathsf{tag}_\Gamma((\boldsymbol{\nu} g\colon (R,G))\ N) &= \{(\boldsymbol{\nu}_t\, g\colon (R,G))\ M \text{ s.t. } M \in \mathsf{tag}_{\Gamma,\mathsf{g}\colon \mathsf{(R,G)@s}}(N)\}
\end{aligned}
$$

Figure 14: The tag function.

further comments.

Tagged and untagged reductions are closely related. We define a tag function, $\mathsf{tag}_\Gamma(N)$, in figure 14, that takes an untagged term N and yields the set of tagged terms that can be obtained from N using the Γ.

The following theorem shows that the tagged and untagged reductions are closely related.

Theorem 2. *Let* $\Gamma \vdash N$.

(i) If $N \to N'$, *then* $\exists M \in \mathsf{tag}_\Gamma(N)$ *s.t.* $M \mapsto M' \in \mathsf{tag}_\Gamma(N')$.

(ii) If $M \in \mathsf{tag}_\Gamma(N) \mapsto M'$, *then* $\exists N'$ *s.t.* $N \to N'$ *and* $M' \in \mathsf{tag}_\Gamma(N')$.

Proof. The proof is by induction on the definition of reduction for the untagged and the tagged reductions, based on a similar result for the structural congruence relation. See appendix for details. □

4.2 Type system

The changes produced to the type system are described in figure 15. The typing rules for processes are left unchanged, but at network level, we propose two substantially different rules: T-Site and T-Resc. Rule T-Site checks, among other things, that the typing assumptions used to tag the process are enough to type it. Moreover, we let the conclusion's typing environment, Γ, be a relaxed version of the one used for tagging the process, Δ. Hence, it is possible to consider the minimum security requirements to type a process, and then enlarge the set of security properties at network level.

$$\textsc{T-Site}\ \frac{\Delta \vdash_s P\colon A \qquad \Delta \vdash s\ \mathsf{allows}\ t\colon A \qquad \Gamma <\colon \Delta \qquad \Delta \vdash t\ \mathsf{canEnter}\ s}{\Gamma \Vdash s[P]^t_\Delta}$$

$$\textsc{T-Resc}\ \frac{\Gamma, a\colon C@s \Vdash N \qquad \Gamma \vdash s\ \mathsf{allows}\ t\colon\ \mathsf{createRes}}{\Gamma \Vdash (\boldsymbol{\nu}_t\, a\colon C@s)\ N}$$

$$\textsc{T-Ress}\ \frac{\Gamma, s\colon G@s \Vdash N}{\Gamma \Vdash (\boldsymbol{\nu}_t\, s\colon G)\ N} \qquad \textsc{T-Resg}\ \frac{\Gamma, g\colon (R,G)@s \Vdash N \qquad g \text{ not in } \Gamma}{\Gamma \Vdash (\boldsymbol{\nu}_t\, g\colon (R,G))\ N}$$

(*plus all rules in figures 9, 10, except* N-Site, *and* N-Res)

Figure 15: The tagged language—typing networks.

Notice that we include the verification of security policies in the rule. It represents a modification to rule N-Site and might be seen contradictory with respect to previous arguments: we are not checking code at its host site, but with the inclusion of the site that sent the code (t), we are in the presence of a kind of an implicit goto process and, therefore, need to check if the code not nested within a goto process is authorised to execute its actions. If the process is running at its host site, then all actions are allowed, since function allows and function canEnter are reflexive (*vide* figures 6 and 7). To understand the need to check the security policies at site level, consider, for instance, that $\Delta \vdash_s P\colon A$. It is always possible to install a process coming from site s at site s, $\Delta <\colon \Gamma \Vdash s[P]^s_\Delta$, but if we want to indicate that the code came from, say site t, $s[P]^t_\Delta$, then it is only allowed if the security polices hold, that is, site s allows site t to execute actions in set A and site t is able to enter site's s border.

The declaration of new channels, at network level, needs also to be checked against the security rules. This fact is clear from the congruence rule $(\boldsymbol{\nu}_t\, n\colon L@s)\ s[P]^t_{\Delta \sqcap n\colon L@s} \equiv s[(\boldsymbol{\nu} n\colon L)\ P]^t_\Delta$. While scope expansion of a channel from a site to a network presents no security violation, since channel creation is verified at process level, the opposite direction needs to be checked. Indeed, we cannot guarantee that if $\Gamma \Vdash (\boldsymbol{\nu}_t\, n\colon L@s)\ s[P]^t_{\Delta \sqcap n\colon L@s}$, then it is the case that $\Gamma \Vdash s[(\boldsymbol{\nu} n\colon L)\ P]^t_\Delta$ without checking weather or not site t is able to create a new channel at site s. Again, there is an implicit goto process going on here. From the network declaration $(\boldsymbol{\nu}_t\, n\colon L@s)\ N$ we infer that there was a code migration from site t to site s, and then the creation of n took place. The remaining typing rules result from syntax modifications.

The following results ensure that types are preserved both by the tagging function, and by the tagged reduction.

Theorem 3 (Tagging preserves types). *if* $\Gamma \vdash N$, *and* $M \in \mathsf{tag}_\Gamma(N)$,

$$\text{R-Out} \quad s[a!\langle v\rangle]^r_\Gamma \overset{err}{\longmapsto} \text{ if } \Gamma \not\Vdash s \text{ allows } r\colon \mathsf{useRes} \text{ or } \Gamma(a) \not<: \langle\text{-}\rangle^{\mathsf{w}}_{@s}$$

$$\text{R-Inp} \quad s[a?(v)\ P]^r_\Gamma \overset{err}{\longmapsto} \text{ if } \Gamma \not\Vdash s \text{ allows } r\colon \mathsf{installRes} \text{ or } \Gamma(a) \not<: \langle\text{-}\rangle^{\mathsf{r}}_{@s}$$

$$\text{R-Mig} \quad s[\mathsf{goto}\ t.P]^r_\Gamma \overset{err}{\longmapsto} \text{ if } \Gamma \not\Vdash s \text{ allows } r\colon \mathsf{go} \text{ or } \Gamma \not\Vdash s\ \mathsf{canEnter}\ t$$

$$\text{R-Res}_1 \quad s[(\boldsymbol{\nu} a\colon T)\ P]^r_\Gamma \overset{err}{\longmapsto} \text{ if } \Gamma \not\Vdash s \text{ allows } r\colon \mathsf{createRes}$$

$$\text{R-Res}_2 \quad (\boldsymbol{\nu}_r\, a\colon T)\ s[P]^r_\Gamma \overset{err}{\longmapsto} \text{ if } \Gamma \not\Vdash s \text{ allows } r\colon \mathsf{createRes}$$

$$\text{R-Res}\ \frac{N \overset{err}{\longmapsto}}{(\boldsymbol{\nu}_r\, n\colon T)\ N \overset{err}{\longmapsto}} \qquad \text{R-Par}\ \frac{N \overset{err}{\longmapsto}}{N \mid M \overset{err}{\longmapsto}}$$

$$\text{R-Str}\ \frac{N \equiv_T M \qquad N \overset{err}{\longmapsto}}{M \overset{err}{\longmapsto}}$$

Figure 16: Runtime errors.

then $\Gamma \Vdash M$.

Proof. A straightforward induction on the structure of N. □

Theorem 4 (Subject reduction – Tagged language). *If* $\Gamma \Vdash N$, *and* $N \mapsto M$, *then* $\Gamma \Vdash M$.

Proof. First we need to establish a similar result for the tagged structural equivalence. Then we proceed by induction on the typing of $\Gamma \Vdash N$, analysing every case for the tagged reduction relation and establishing induction on the last rule applied. □

4.3 Type safety

We are now in position to define the notion of runtime errors. Our claim in this section is that well-typed processes are free of runtime errors. The unary relation, $\overset{err}{\longmapsto}$, defined in figure 16, identifies processes that misbehave either due to communication problems or, the important issue for us, because they break some security policy. Next, we comment on the definition.

The output (input) process fails, R-Outc (R-Inp), if the site that sent the code, r, has no permission to use (install) resources, or if channel a is not a write (read) or read-write channel. We omit the rule for replicated input, since it is similar to rule R-Inp.

For code migration, rule R-MIG states that a goto process incurs in a runtime error if it cannot enter the border of the groups where the target site resides, or if it cannot escape the current site. Notice the role of typing Γ—a placeholder for security policies—, and the need to talk about the site where the code is, s, the site that sent the code, r, and the site where the code is migrating to, t.

Rule R-RES$_1$ says that the channel creation operation fails if the current site does not give permission to create channels to the site that sent the code. Rule R-RES$_2$ is similar.

Rules R-INPC, R-INPR$_1$, R-INPU, and R-INPR$_2$ describe the possible runtime errors for an input process either replicated, or inputing a `unit` value or a channel. Similarly to the runtime errors of an output process, the errors concern communication and security. The communication errors have to do with the capacity of channel a to be an input channel located at the site where the resource is installed. A security error arises if the site that installed the resource has no permission for that, namely, site r installed a resource a at site s without the appropriate authority.

The type safety result states that well-typed networks do not incur in runtime errors.

Theorem 5. *If* $\Gamma \Vdash N$*, then* $M \not\overset{err}{\longmapsto}$.

Proof. We prove the contrapositive result, namely, that $\mathsf{tag}_\Gamma(N') \overset{err}{\longmapsto}$ implies that there is no Γ s.t. $\Gamma \Vdash N$. We proceed by induction on the definition of $\overset{err}{\longmapsto}$ relation. □

Corollary 6 (Type safety). *If* $\Gamma \vdash N$*, and* $N \to^\star N'$*, then* $\mathsf{tag}_\Gamma(N') \not\overset{err}{\longmapsto}$.

Proof. Use theorems 1, 3, and 5. □

5 The Mikado approach

In this section we discuss the relationship between our security proposal and the guidelines defined by the Mikado core model [1].

The Mikado's approach introduces the concept of a domain that may be composed with other domains to form a network. A domain consists of two distinct areas: (a) a *guardian*—an entity that monitors the interaction with other domains, and that supervises (with the power to intervene in) the computational area; (b) a *computational place* where processes execute their actions. Following a syntax similar to [1], a domain may be written as $s\{\!|G|\!\}[P]$, where s is the domain's name, G is its guardian, and P is a process running there.

The domain structure may be organised hierarchically, allowing the nesting of domains at the guardian part, or at the computation area of the domain. The interactions with other domains are supervised by the domain's guardian. All the incoming, and the outgoing migrations (messages and processes) targeted for a certain domain undergo the security sieve defined by the domain's controller. Thus, a message aiming at an inner domain needs to interact will all the guardian domains in the hierarchy until it reaches its final destination.

Many of the ideas described in our work intercept the concepts laid down by the Mikado model, and can be expressed within it. To a certain extend, we may understand our work as an instance of the Mikado model, apart from specific details like, for instance, the addressing of resources. Indeed, some features that arise naturally in our framework, and improve the writing of the security rules are not easily stated in the Mikado model. The ability to specify that a group is an offspring of several groups, or that a site may belong to several groups, helps setting up security. It seems that our group hierarchy—an acyclic graph—suits better the definition of security policies than the tree structure organisation suggested by the Mikado model.

Since domains can be nested at the membrane part, or at the computational area, we present two alternatives in which our calculus can be partially expressed as an instance of the Mikado core model. In the first alternative,

- *a site is a domain* with a controller that allows migrating processes to get in and out freely, since sites say nothing about security polices, and that executes the code of the site in the computational area. Site $s[P]$ is represented by the domain $s\{\!|I|\!\}[P]$, where I is the "identity" guardian driving in and out all the migrating code.

- *a group is a domain* containing a guardian that control group's security policies, and that executes the stop process, since a group has no computational power. Group g, enforcing security policies R, is represented by the domain $g\{\!|R|\!\}[\mathsf{stop}]$.

- the groups or sites that constitute a group appear as subdomains of the domain representing the intended group.

The other proposal is to consider

- *a site* as defined above;

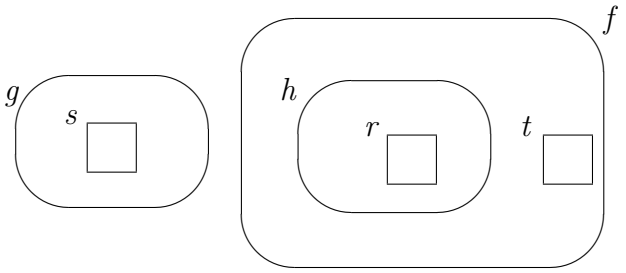
- *a group as a domain* with a controller that implement the group's security policies; sites and subgroups are represented as nested domains in the computational area.

These approaches do not differ significantly and, in particular, do not overcome the features that we identify as hard to represent in the Mikado core model. Of course there are other ways to represent our security approach using Mikado's ideas, but still, we do not see any benefits of doing so.

Let us focus on the first proposal we describe. Using such a method to represent sites and groups avoids the spread of security policies replicas through domain controllers. Of course one easier way to mimic our calculus with the Mikado model is just to write a flat network of domains having each controller equipped with security rules of the groups to which the site belongs, but is there any benefit in this approach? We are just making the life of the security officer more difficult, increasing the number of controllers to set up and to maintain. Notice that networks tend to have clusters of sites sharing the same security policies.

As a first example, consider a group g implementing security policies R, composed of two sites s, and r hosting process P, and process Q, respectively. This network can be sketched in the Mikado core model as $g\{\!|R \,|\, s\{\!|I|\!\}[P] \,|\, r\{\!|I|\!\}[Q]|\!\}[\mathsf{stop}]$. There exist three domains, g, s, and r, where domain g controls the security policies defined by R. Domain s and domain r do not interfere with security polices and just host process P, and process Q. Note, however, that if we follow the alternative method of representing a site as a domain, the resulting network would be $s\{\!|G|\!\}[P] \,|\, r\{\!|G|\!\}[R]$, where the security policies, G, appear twice.

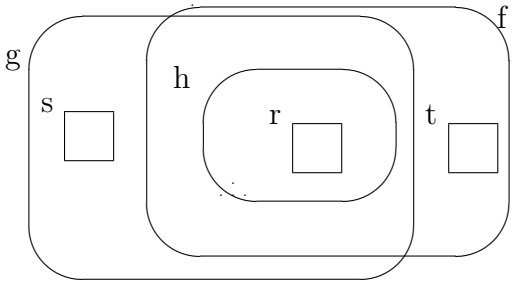
If we have a network like the one depicted below,



we can setup a domain network like

$$g\{\!|R_g \,|\, s\{\!|I|\!\}[P_s]|\!\}[\mathsf{stop}] \,|\, f\{\!|R_f \,|\, h\{\!|R_h \,|\, r\{\!|I|\!\}[P_r]|\!\}[\mathsf{stop}] \,|\, t\{\!|I|\!\}[P_t]|\!\}[\mathsf{stop}]$$

Let us consider now that group h is a subgroup of both group g and group f.



The only way we see to implement the network using domains is to destroy the hierarchy and specifically program the domain guardians to mimic this situation.

A possible solution is to keep group h as a subgroup of f and to program the guardian of the domain that represents g to redirect the accesses to site r via the domain that represents group f, an in

$$f\{R'_f \mid h\{R_h \mid r\{I\}[P_r]\}[\mathsf{stop}] \mid t\{I\}[P_t]\}[\mathsf{stop}] \mid g\{R_g \mid s\{I\}[P_s]\}[Q]$$

where Q is a process that redirects migrations to domain r to domain f, and R'_f is a change in the security policies of group f to allow the redirections from domain g to get through.

But this change is just half of the story. When site s migrates code to another site, it can present itself as a member of group g, a member of group f, or a member of group h. Since we check the security policies at destination domain, it is not entirely obvious how to achieve this kind of behaviour. To overcome the situation, we need to know the network topology, and the sites location in advance to be able to verify the policies at domain level.

6 Conclusions and related work

Summary. We present an approach to express and control security policies using types. We use Dπ as the underlying calculus and, on top of it, define an hierarchical structure of security groups. The security model that we propose is based on the notion of security groups and constitute an experiment of the Mikado core model, exploring the role of domain guardians. A security group delimits a region of the network with the same security requirements and may be understood as a firewall that dictates and supervises the sites under its control. We use a type system as the security mechanism to enforce that networks respect the security policies defined by groups

Beyond the Mikado core model we investigate two further directions: (a) the decoupling of the guardian part of the domain from its computational

area, and (b) the study of a different hierarchy model to express the dependencies between security groups. It seems that a richer hierarchy model—the use of an acyclic graph instead of a tree—is a promising line for further investigations, since it allows for the representation of plausible situations in a more natural way. The decoupling of the two parts of the domain is a simple schema that we use to focus on the security aspects of the model and to rehearse the alternative hierarchical approach.

Ongoing work includes the refinement of the type system to enforce a fine grained control on resources' security.

Related work. Many works concerning security in distributed, and mobile environments have been recently proposed, ranging from type systems [4, 5, 8, 10, 11, 13–15] to control flow analysis [3, 9, 12, 18], and to proof carrying code [17]. Refer to [2] for a general survey on concurrent mobile calculus, type systems, and security policies.

We conclude with a brief comparison with those works closer to ours: the Ambient calculus [4–6, 15], Dπ [13, 14], and KLAIM [7, 8, 10, 11]. As far as we known, our security model is the first to combine group policies, and to define an hierarchy for security groups, helping the writing of security rules, and allowing to reuse the existent ones.

Cardelli, Ghelli, and Gordon introduced a notion of groups for the Ambient calculus [4, 5] to control the movement, and the opening of ambients. They use groups to combine ambients in clusters, but specify the security properties for each ambient regardless the group the ambient belongs to. Instead, we use groups to specify security policies shared by the sites that compose each group. Lhoussaine and Sassone [15], use dependent types as an alternative to groups. The type system is far more complex, but allows greater flexibility when writing security rules.

Hennessy and Riely proposed advanced type systems [13, 14] to control resource access in Dπ. The control of migration is found along three aspects: a keyword **mig**, a subtype relation, and the ability to communicate site names. If a process "sees" the **mig** keyword as part of the type of a site, then it may migrate code to that site. The subtype relation, together with the capability to communicate site names, allows for a site to tailor the information (*e.g.* resource names, control keywords) that the target site is able to use. From a programming point of view, this approach does not seem very attractive, since security annotations are spread throughout the code, and it is difficult to understand what actions is a piece of code really allowed to execute.

KLAIM uses a capability type system to control operations on tuple spaces. The KLAIM approach is similar to ours in the sense that security

policies are declared at site level, but differs substantially when we consider how policies are programmed and checked. One main distinction concerns the place where the security policies are defined: security policies in KLAIM talk about what operations a site may perform on other sites, whereas in our framework each security group talks about what actions it allows others to perform on it. From the administrator's point of view this looks more adequate. Recent type systems proposed for μKLAIM tackle the compilation of open systems, using a kind of partial compilation mechanism that marks parts of the processes that cannot be checked statically to be analysed at runtime [10, 11].

Acknowledgments

The authors would like to acknowledge the financial support of the EU Global computing project Mikado, and the fruitful discussions with Matthew Hennessy and António Ravara. We also wish to acknowledge the hospitality of the School of Cognitive and Computing Sciences, University of Sussex.

References

[1] G. Boudol. A parametric model of migration and mobility, release 1. Mikado Deliverable D1.2.1, 2003.

[2] G. Boudol, I. Castellani, F. Germain, and M. Lacoste. Models of distribution and mobility: State of the art. Mikado Deliverable D1.1.1, 2002.

[3] C. Braghin, A. Cortesi, and R. Focardi. Security boundaries in mobile ambients. *Computer Languages*, 28(1):101–127, 2002.

[4] L. Cardelli, G. Ghelli, and A. Gordon. Mobility types for mobile ambients. In *Proceedings of ICALP'99*, volume 1644 of *LNCS*, pages 230–239. Springer-Verlag, 1999.

[5] L. Cardelli, G. Ghelli, and A. Gordon. Ambient groups and mobility types. In *Proceedings of TCS'00*, volume 1872 of *LNCS*, pages 333–347. Springer-Verlag, 2000.

[6] L. Cardelli and A. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.

[7] R. De Nicola, G. Ferrari, and R. Pugliese. KLAIM: a Kernel Language for Agents Interaction and mobility. *IEEE Trans. in Software Engineering*, 24(5):315–330, 1998.

[8] R. De Nicola, G. Ferrari, R. Pugliese, and B. Veneri. Types for access control. *Theoretical Computer Science*, 240(1):215–254, 2000.

[9] P. Degano, F. Levi, and C. Bodei. Safe ambients: Control flow analysis and security. In *Proceedings of ASIAN'00*, volume 1691 of *LNCS*, pages 199–214. Springer-Verlag, 2000.

[10] D. Gorla and R. Pugliese. Resource access and mobility control with dynamic privileges acquisition. In *Proceedings of ICALP'03*, volume 2719 of *LNCS*, pages 119–132. Springer-Verlag, 2003.

[11] D. Gorla and R. Pugliese. Controlling data movement in global computing applications. In *Proceedings of SAC'04*. ACM Press, 2004.

[12] R. Hansen, J. Jensen, F. Nielson, and H. Nielson. Abstract interpretation of mobile ambients. In *Proceedings of SAS'99*, volume 1694 of *LNCS*, pages 134–148. Springer-Verlag, 1999.

[13] M. Hennessy, M. Merro, and J. Rathke. Towards a behavioural theory of access and mobility control in distributed systems. *Theoretical Computer Science*, 2003.

[14] M. Hennessy and J. Riely. Resource access control in systems of mobile agents. *Journal of Information and Computation*, 173:82–120, 2002.

[15] C. Lhoussaine and V. Sassone. A dependently typed ambient calculus. In *Proceedings of ESOP'03*, LNCS. Springer-Verlag, 2003.

[16] F. Martins and V. Vasconcelos. Controlling security policies in a distributed environment. DI/FCUL TR 04–01, 2004.

[17] G. Necula. Proof-Carrying Code. In *Proceedings of POPL'97*, pages 106–119. Springer-Verlag, 1999.

[18] F. Nielson, H. Nielson, R. Hansen, and J. Jensen. Validating firewalls in mobile ambients. In *Proceedings of CONCUR'99*, volume 1664 of *LNCS*, pages 463–477. Springer-Verlag, 1999.

[19] B. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–454, 1996.

A Proofs

A.1 Proofs from section 3

In this section we present the proof for the subject reduction theorem for the untagged language, theorem 1. The proof uses two main auxiliary results, namely, a substitution lemma (lemma 13), and a structural congruence lemma (lemma 16).

The proof of the substitution lemma uses several results that we claim below, but it is mainly based on three auxiliary results: the substitution on environments, lemma 9, the substitution on sites, lemma 10, and the substitution on channels, lemma 12.

Lemma 7 (Weakening on allows). *If* $\Gamma \vdash F'$ allows $G' \colon A'$, $F \subseteq F'$, $G \subseteq G'$, *and* $A \subseteq A'$, *then* $\Gamma \vdash F$ allows $G \colon A$.

Proof. If $\Gamma \vdash F'$ allows $G' \colon A'$, then $\Gamma \vdash f$ allows $g \colon \pi$ for all f, g, and π in F', G', and A', respectively. Since $F \subseteq F'$, $G \subseteq G'$, and $A \subseteq A'$, then $\forall f \in F \Rightarrow f \in F'$, $\forall g \in G \Rightarrow g \in G'$, and $\forall \pi \in A \Rightarrow \pi \in A'$, respectively. Therefore,

$$\frac{\Gamma \vdash f \text{ allows } g \colon \pi \qquad \forall f \in F, \forall g \in G, \forall \pi \in A}{\Gamma \vdash F \text{ allows } G \colon A}.$$

□

Lemma 8 (Weakening on canEnter). *If* $\Gamma \vdash F'$ canEnter G', $F \subseteq F'$, *and* $G \subseteq G'$, *then* $\Gamma \vdash F$ canEnter G.

Proof. This lemma is proved using a similar scheme to the proof of lemma 7. □

Lemma 9 (Substitution on environments). *If* $\Gamma, s \colon G \vdash \text{env}$, *then* $\Gamma\{r/s\} \vdash \text{env}$.

Proof. Follows by straightforward induction on the typing derivation of $\Gamma \vdash \text{env}$. Proceed by case analysis on the structure of Γ, considering the last rule applied for the derivation of $\Gamma \vdash \text{env}$. □

Lemma 10 (Substitution on sites). *If* $\Gamma(r) = G$, $G \subseteq G'$, *and* $\Gamma, s \colon G' \vdash_t P \colon A$, *then* $\Gamma\{r/s\} \vdash_{t\{r/s\}} P\{r/s\} \colon A$

Proof. By induction on the typing derivation. We proceed by case analysis on the structure of P, tracing the last typing rule applied.

- Case $\Gamma, s\colon G' \vdash_t a!\langle b \circledcirc u\rangle\colon \{\mathsf{useRes}\}$.
  Then, the following derivation holds

$$\frac{\begin{array}{cc}\Gamma, s\colon G' \vdash \mathrm{env} & \Gamma(a) <\colon \langle C \circledcirc G''\rangle^{\mathsf{w}} \circledcirc t \\ \Gamma(u) = G'' & \Gamma(b) = C \circledcirc u\end{array}}{\Gamma, s\colon G' \vdash_t a!\langle b \circledcirc u\rangle\colon \{\mathsf{useRes}\}}$$

  We need to consider four subcases:

  - Case $s = t = u$.
    Since $\Gamma, s\colon G' \vdash \mathrm{env}$, then $\Gamma\{r/s\} \vdash \mathrm{env}$ by lemma 9. By hypothesis $\Gamma(r) = G \subseteq G' = G''$, then $\langle C \circledcirc G'\rangle^{\mathsf{w}} \circledcirc r <\colon \langle C \circledcirc G\rangle^{\mathsf{w}} \circledcirc r$. Therefore, the following type derivation holds.

$$\frac{\begin{array}{cc}\Gamma\{r/s\} \vdash \mathrm{env} & \Gamma\{r/s\}(a) <\colon \langle C \circledcirc G\rangle^{\mathsf{w}} \circledcirc s \\ \Gamma\{r/s\}(r) = G & \Gamma\{r/s\}(b) = C \circledcirc r\end{array}}{\Gamma\{r/s\} \vdash_r a!\langle b \circledcirc r\rangle\colon \{\mathsf{useRes}\}}$$

  - Case $s = t$ and $s \neq u$.
    By lemma 9, $\Gamma\{r/s\} \vdash \mathrm{env}$, and the type derivation holds.

$$\frac{\begin{array}{cc}\Gamma\{r/s\} \vdash \mathrm{env} & \Gamma\{r/s\}(a) <\colon \langle C \circledcirc G''\rangle^{\mathsf{w}} \circledcirc s \\ \Gamma\{r/s\}(u) = G'' & \Gamma\{r/s\}(b) = C \circledcirc u\end{array}}{\Gamma\{r/s\} \vdash_s a!\langle b \circledcirc u\rangle\colon \{\mathsf{useRes}\}}$$

  - Case $s \neq t$ and $s = u$.
    Since $G \subseteq G'$ by hypothesis, and $\Gamma\{r/s\} \vdash \mathrm{env}$ holds by lemma 9, so

$$\frac{\begin{array}{cc}\Gamma\{r/s\} \vdash \mathrm{env} & \Gamma\{r/s\}(a) <\colon \langle C \circledcirc G\rangle^{\mathsf{w}} \circledcirc t \\ \Gamma\{r/s\}(r) = G & \Gamma\{r/s\}(b) = C \circledcirc r\end{array}}{\Gamma\{r/s\} \vdash_t a!\langle b \circledcirc s\rangle\colon \{\mathsf{useRes}\}}$$

  - Case $s \neq t$ and $s \neq u$.
    The process is not affected by the substitution, and lemma 9 guarantees that $\Gamma\{r/s\} \vdash \mathrm{env}$. So, the following derivation concludes the case.

$$\frac{\begin{array}{cc}\Gamma\{r/s\} \vdash \mathrm{env} & \Gamma\{r/s\}(a) <\colon \langle C \circledcirc G''\rangle^{\mathsf{w}} \circledcirc t \\ \Gamma\{r/s\}(u) = G'' & \Gamma\{r/s\}(b) = C \circledcirc u\end{array}}{\Gamma\{r/s\} \vdash_t a!\langle b \circledcirc u\rangle\colon \{\mathsf{useRes}\}}$$

- Case $\Gamma, s\colon G' \vdash_t a!\langle \diamond\rangle\colon \{\mathsf{useRes}\}$
  Hence, the following derivation holds.

$$\frac{\Gamma, s\colon G' \vdash \mathrm{env} \qquad \Gamma(a) <\colon \langle \mathsf{unit}\rangle^{\mathsf{w}} \circledcirc t}{\Gamma, s\colon G' \vdash_t a!\langle b \circledcirc u\rangle\colon \{\mathsf{useRes}\}}$$

  We come across two subcases:

– Case $s = t$.
Therefore, by hypothesis and by lemma 9,

$$\frac{\Gamma\{r/s\} \vdash \text{env} \qquad \Gamma\{r/s\}(a) <: \langle \mathsf{unit} \rangle^{\mathsf{w}}{}_{@} r}{\Gamma\{r/s\} \vdash_r a!\langle \diamond \rangle : \{\mathsf{useRes}\}}$$

– Case $s \neq t$.
Process $a!\langle b_{@} r \rangle$ is not affected by the substitution. The next derivation concludes the case.

$$\frac{\Gamma\{r/s\} \vdash \text{env} \qquad \Gamma\{r/s\}(a) <: \langle \mathsf{unit} \rangle^{\mathsf{w}}{}_{@} t}{\Gamma\{r/s\} \vdash_t a!\langle \mathsf{unit} \rangle : \{\mathsf{useRes}\}}$$

- Case $\Gamma, s : G' \vdash_t a?(x_{@} y)\ P : A \cup \{\mathsf{useRes}\}$.
By hypothesis the type derivation holds.

$$\frac{\begin{array}{c}\Gamma, s : G', x : C'', y : G'' \vdash_s P : A \\ \Gamma(a) <: \langle C''_{@} G'' \rangle^{\mathsf{r}}{}_{@} t\end{array}}{\Gamma, s : G' \vdash_t a?(x_{@} y)\ P : A \cup \{\mathsf{useRes}\}}$$

We consider two subcases:

– $s = t$.
From the facts that $\Gamma, s : G', x : G'', y : G''$, and $G \subseteq G'$, we can apply the induction hypothesis, and obtain the following type derivation.

$$\frac{\begin{array}{c}\Gamma\{r/s\}, x : C'', y : G'' \vdash_r P\{r/s\} : A \\ \Gamma(a) <: \langle C''_{@} G'' \rangle^{\mathsf{r}}{}_{@} r\end{array}}{\Gamma\{r/s\} \vdash_r a?(x_{@} y)\ P\{r/s\} : A \cup \{\mathsf{installRes}\}}$$

– $s \neq t$.
The substitution of s for r only affects P. Therefore the type derivation holds by induction hypothesis (as for the previous case).

$$\frac{\begin{array}{c}\Gamma\{r/s\}, x : C'', y : G'' \vdash_t P\{r/s\} : A \\ \Gamma(a) <: \langle C''_{@} G'' \rangle^{\mathsf{r}}{}_{@} t\end{array}}{\Gamma\{r/s\} \vdash_t a?(x_{@} y)\ P\{r/s\} : A \cup \{\mathsf{installRes}\}}$$

- Case $\Gamma, s : G' \vdash_t a?(\diamond)\ P : A \cup \{\mathsf{useRes}\}$.
This case has two subcases that are handled much like the previous case. Therefore we omit the proof.

- Case $\Gamma, s\colon G' \vdash_t a?*(v)\ P\colon A \cup \{\mathsf{useRes}\}$.
  By hypothesis,
  $$\frac{\Gamma, s\colon G' \vdash_t a?(v)\ P\colon A}{\Gamma, s\colon G' \vdash_t a?*(v)\ P\colon A}$$
  Applying the induction hypothesis,
  $$\frac{\Gamma\{r/s\} \vdash_{t\{r/s\}} a?(v)\ P\{r/s\}\colon A}{\Gamma\{r/s\} \vdash_{t\{r/s\}} a?*(v)\ P\{r/s\}\colon A}$$

- Case $\Gamma \vdash_t \mathsf{stop}$.
  Holds by lemma 9.
  $$\frac{\Gamma\{r/s\} \vdash \mathrm{env}}{\Gamma\{r/s\} \vdash_{t\{r/s\}} \mathsf{stop}}$$

- Case $\Gamma, s\colon G' \vdash_t (\boldsymbol{\nu} n\colon L)\ P\colon A$.
  Then, it must be the case that
  $$\frac{\Gamma, s\colon G', n\colon L@t \vdash_t P\colon A \qquad n \text{ not in } \Gamma, s\colon G'}{\Gamma, s\colon G' \vdash_t (\boldsymbol{\nu} n\colon L)\ P\colon A}$$
  Hence, applying lemma 9, induction hypothesis, and using the fact that $G \subseteq G'$,
  $$\frac{\Gamma\{r/s\}, n\colon L@t\{r/s\} \vdash_{t\{r/s\}} P\{r/s\}\colon A \qquad n \text{ not in } \Gamma\{r/s\}}{\Gamma\{r/s\} \vdash_{t\{r/s\}} (\boldsymbol{\nu} n\colon L)\ P\{r/s\}\colon A}$$
  holds.

- Case $\Gamma, s\colon G' \vdash_t P \mid Q\colon A \cup B$.
  This case (as the one before) easily follows by induction hypothesis.

- Case $\Gamma, s\colon G' \vdash_t \mathsf{goto}\ u.P\colon \{\mathsf{go}\}$.
  The following derivation holds.
  $$\frac{\begin{array}{c}\Gamma, s\colon G' \vdash_u P\colon A \\ \Gamma, s\colon G' \vdash u\ \mathsf{allows}\ t\colon A \qquad \Gamma, s\colon G' \vdash t\ \mathsf{canEnter}\ u\end{array}}{\Gamma, s\colon G' \vdash_t \mathsf{goto}\ u.P\colon \{\mathsf{go}\}}$$
  We need to discuss four subcases:

  - Case $s = t = u$.
    By induction hypothesis, $\Gamma\{r/s\} \vdash_s P\{r/s\}\colon A$, and from the fact that $\Gamma\{r/s\} \vdash r\ \mathsf{allows}\ r$, and $\Gamma\{r/s\} \vdash r\ \mathsf{canEnter}\ r$ for any r, $\Gamma\{r/s\} \vdash_r \mathsf{goto}\ r.P\colon \{\mathsf{go}\}$ holds.

– Case $s = t$ and $s \neq u$.
Using induction hypothesis, $\Gamma\{r/s\} \vdash_u P\{r/s\} \colon A$, holds. Since $G \subseteq G'$, $\Gamma, s \colon G' \vdash u$ allows $s \colon A$, and $\Gamma, s \colon G' \vdash s$ $\mathsf{canEnter}$ u, lemmas 7 and 8 guarantee that $\Gamma\{r/s\} \vdash u$ allows $r \colon A$, and $\Gamma\{r/s\} \vdash r$ $\mathsf{canEnter}$ u, hold, respectively. Hence, we conclude $\Gamma\{r/s\} \vdash_r \mathsf{goto}\ u.P\{r/s\} \colon \{\mathsf{go}\}$.

– Case $s \neq t$ and $t = u$.
We conclude $\Gamma\{r/s\} \vdash_t \mathsf{goto}\ s.P\{r/s\} \colon \{\mathsf{go}\}$ from induction hypothesis, lemma 7, and lemma 8, considering that $\Gamma, s \colon G' \vdash_s P \colon A$, $G \subseteq G'$, $\Gamma, s \colon G' \vdash s$ allows $t \colon A$, and $\Gamma, s \colon G' \vdash t$ $\mathsf{canEnter}$ s.

– Case $s \neq t$ and $s \neq u$.
The substitution only affects the continuation term P. Consequently, $\Gamma\{r/s\} \vdash_u P\{r/s\} \colon A$, by induction hypothesis. Hence, the following derivations holds.

$$\frac{\Gamma\{r/s\} \vdash_u P\{r/s\} \colon A \qquad \Gamma\{r/s\} \vdash u\ \mathsf{allows}\ t \colon A \qquad \Gamma\{r/s\} \vdash t\ \mathsf{canEnter}\ u}{\Gamma\{r/s\} \vdash_t \mathsf{goto}\ u.P \colon \{\mathsf{go}\}}$$

This concludes the proof. □

Lemma 11 (Strengthening on env**).** *If* $\Gamma, n \colon T \vdash \mathrm{env}$*, then* $\Gamma \vdash \mathrm{env}$*.*

Proof. Follows directly from the definition of $\Gamma \vdash \mathrm{env}$. □

Lemma 12 (Substitution on channels). *If* $\Gamma(b) = C_{@}s$*,* $C <: C'$*, and* $\Gamma, a \colon C'_{@}s \vdash_t P \colon A$*, then* $\Gamma \vdash_t P\{b/a\} \colon A$*.*

Proof. By induction on the typing derivation. We proceed by case analysis on the structure of P, tracing the last typing rule applied.

- Case $P \equiv c!\langle d_{@}u \rangle$.
There are three subcases: when a is c, when a is d, or when a is different from c and d. Notice that a cannot be simultaneously c and d, because we do not address recursive types. We only discuss the first to subcases, since the third subcase does not affect the output process, and therefore holds by hypothesis.

  – Case $a = c$. Therefore $s = t$.
  By hypothesis $\Gamma, a \colon C'_{@}s \vdash_s a!\langle d_{@}u \rangle \colon \{\mathsf{useRes}\}$ holds. Hence,

$$\frac{\Gamma, a \colon C'_{@}s \vdash \mathrm{env} \qquad C'_{@}s <: \langle C''_{@}G \rangle^{\mathsf{w}}{}_{@}s \qquad \Gamma(u) = G \qquad \Gamma(d) = C''_{@}u}{\Gamma, a \colon C'_{@}s \vdash_s a!\langle d_{@}u \rangle \colon \{\mathsf{useRes}\}}$$

Using the fact that $C <: C'$, and by lemma 11, the following type derivation holds.

$$\frac{\begin{array}{cc}\Gamma \vdash \text{env} & C @ s <: C' @ s <: \langle C'' @ G \rangle^{\mathsf{w}} @ s \\ \Gamma(u) = G & \Gamma(d) = C'' @ u\end{array}}{\Gamma \vdash_s b!\langle d @ u \rangle : \{\mathsf{useRes}\}}$$

– Case $a = c$. Therefore $s = u$.
So,

$$\frac{\Gamma, a : C' @ s \vdash \text{env} \quad \Gamma(c) <: \langle C' @ G \rangle^{\mathsf{w}} @ s \quad \Gamma(s) = G}{\Gamma, a : C' @ s \vdash_t c!\langle a @ s \rangle : \{\mathsf{useRes}\}}$$

holds.
Finally, applying lemma 11, and using the fact that if $C <: C'$, then $\langle C' @ G \rangle^{\mathsf{w}} @ t <: \langle C @ G \rangle^{\mathsf{w}} @ t$, we can write the type derivation

$$\frac{\begin{array}{cc}\Gamma \vdash \text{env} & \Gamma(c) <: \langle C' @ G \rangle^{\mathsf{w}} @ t <: \langle C @ G \rangle^{\mathsf{w}} @ t \\ \Gamma(s) = G & \Gamma(b) = C @ s <: C' @ s\end{array}}{\Gamma \vdash_t c!\langle b @ s \rangle : \{\mathsf{useRes}\}}$$

and conclude the case.

- Case $P \equiv c!\langle \diamond \rangle$.
We discuss the case when $a = c$, since the case when no substitution takes place holds trivially. Then, $t = s$ and by hypothesis

$$\frac{\Gamma, a : C' @ s \vdash \text{env} \quad C' @ s <: \langle \mathsf{unit} \rangle^{\mathsf{w}} @ s}{\Gamma, a : C' @ s \vdash_s a!\langle \diamond \rangle : \{\mathsf{useRes}\}}$$

holds. Since $\Gamma(b) <: \Gamma(a)$, and using lemma 11, the following type derivations concludes the case

$$\frac{\Gamma \vdash \text{env} \quad \Gamma(b) <: C' @ s <: \langle \mathsf{unit} \rangle^{\mathsf{w}} @ s}{\Gamma \vdash_s b!\langle \diamond \rangle : \{\mathsf{useRes}\}}$$

- Case $P \equiv c?(x @ y)\ Q$.
There are two subcases. Since $\Gamma, a : C' @ s \vdash_t P : A \cup \{\mathsf{installRes}\}$, then it must be the case that

$$\frac{\begin{array}{c}\Gamma, a : C' @ s, x : C'' @ y, y : G'' \vdash_t Q : A \\ \Gamma(c) <: \langle C'' @ G'' \rangle^{\mathsf{r}} @ t\end{array}}{\Gamma, a : C' @ s \vdash_t c?(x @ y)\ Q : A \cup \{\mathsf{installRes}\}}$$

holds.

– Case $a = c$. Therefore $s = t$.
Considering that $\Gamma(b) <: \Gamma(a)$, and applying induction hypothesis, the following type derivation holds.

$$\frac{\Gamma, x\colon C''@y, y\colon G'' \vdash_s Q\{b/a\}\colon A \quad \Gamma(b) = C@s <: C'@s <: \langle C''@G''\rangle^{\mathsf{r}}@s}{\Gamma \vdash_s b?(x@y)\ Q\{b/a\}\colon A \cup \{\mathsf{installRes}\}}$$

– Case $a \neq c$.
This case holds directly by applying induction hypothesis. Therefore,

$$\frac{\Gamma, x\colon C''@y, y\colon G'' \vdash_t Q\{b/a\}\colon A \quad \Gamma(c) <: \langle C''@G''\rangle^{\mathsf{r}}@t}{\Gamma \vdash_t c?(x@y)\ Q\{b/a\}\colon A \cup \{\mathsf{installRes}\}}$$

- Case $P \equiv c?(\diamond)\ Q$.
As for the previous case, we need to consider two subcases (for $a = c$, and for $a \neq c$) that are proved using similar arguments to the ones presented; so, we omit the proof.

- Case $P \equiv c?*(v)\ Q$.
There are two subcases: when $a = c$, and when $a \neq c$. We just elaborate on $a = c$, since the other case holds easily by induction hypothesis, and similar cases were illustrated previously. When $a = c$, we apply induction hypothesis to $\Gamma, a\colon C'@s \vdash_s a?(v)\ Q\colon A$ and get

$$\frac{\Gamma \vdash_s b?(v)\ Q\{b/a\}\colon A}{\Gamma \vdash_s b?*(v)\ Q\{b/a\}\colon A}$$

which concludes the case.

- Case $P \equiv (\boldsymbol{\nu} n\colon T)\ Q$, $P \equiv \mathsf{goto}\ u.Q$, or $P \equiv Q_1 \mid Q_2$.
All these cases are handled similarly, using induction hypothesis, so we elaborate on $P \equiv Q_1 \mid Q_2$ as and example. Since $\Gamma, a\colon C'@s \vdash_t Q_1 \mid Q_2$, then it must be the case that

$$\frac{\Gamma, a\colon C@s \vdash_t Q_1\colon A \quad \Gamma, a\colon C@s \vdash_t Q_2\colon B}{\Gamma, a\colon C@s \vdash_t Q_1 \mid Q_2\colon A \cup B}$$

holds. Considering $\Gamma(b) = C@s <: C'@s = \Gamma(a)$, and using induction hypothesis, the following type derivation holds, and we conclude the case.

$$\frac{\Gamma \vdash_t Q_1\{b/a\}\colon A \quad \Gamma \vdash_t Q_2\{b/a\}\colon B}{\Gamma \vdash_t Q_1\{b/a\} \mid Q_2\{b/a\}\colon A \cup B}$$

- Case $P \equiv \mathsf{stop}$
  Holds trivially.

This concludes the proof. □

Lemma 13 (Substitution lemma). *If* $\Gamma(b) = C_{@}r$, $C <: C'$, $\Gamma(r) \subseteq G'$, *and* $\Gamma, s\colon G', a\colon C'_{@}s \vdash_t P\colon A$, *then* $\Gamma\{r/s\} \vdash_{t\{r/s\}} P\{r/s\}\{b/a\}\colon A$.

Proof. Considering $\Gamma, s\colon G', a\colon C'_{@}s \vdash_t P$, and $\Gamma(r) \subseteq G'$, by lemma 10 we conclude that $\Gamma\{r/s\}, a\colon C'_{@}r \vdash_{t\{r/s\}} P\{r/s\}\colon A$. Since $\Gamma(b) = C_{@}r$ and $C <: C'$, then, by lemma 12, $\Gamma\{r/s\} \vdash_{t\{r/s\}} P\{r/s\}\{b/a\}\colon A$. □

Lemma 14 (Weakening lemma).

1. *If* $\Gamma \vdash_t P$, *and* $\Gamma, m\colon L_{@}s \vdash \mathrm{env}$, *then* $\Gamma, m\colon L_{@}s \vdash_t P$.
2. *If* $\Gamma \vdash N$, *and* $\Gamma, m\colon L_{@}s \vdash \mathrm{env}$, *then* $\Gamma, m\colon L_{@}s \vdash M$.

Proof. The proof of case 1 is by induction on the typing of $\Gamma \vdash_t P$. We proceed by case analysis on the structure of P, discussing the last typing rule applied. The proof is straightforward, so we just discuss two cases, namely,

- Case P is stop.
  Therefore, the following type derivation holds.

  $$\frac{\Gamma \vdash_t \mathrm{env}}{\Gamma \vdash_t \mathsf{stop}}$$

  Since, by hypothesis, $\Gamma, m\colon L_{@}s \vdash \mathrm{env}$, then $\Gamma, m\colon L_{@}s \vdash_t \mathsf{stop}$.

- Case P is $(\boldsymbol{\nu} r\colon G)\, Q$.
  The last (instance of the) typing rule applied is

  $$\frac{\Gamma, r\colon G_{@}t \vdash_t Q\colon A \qquad r \text{ not in } \Gamma}{\Gamma \vdash_t (\boldsymbol{\nu} r\colon G)\, Q\colon A}$$

  We can assume that the names in $m\colon T$ are different from the $\mathrm{bn}(Q) \cup \{r\}$ (or else we could $\alpha-$convert them). Therefore, using the induction hypothesis

  $$\frac{\Gamma, m\colon T, r\colon G_{@}t \vdash_t Q\colon A \qquad r \text{ not in } \Gamma, m\colon T}{\Gamma, m\colon T \vdash_t (\boldsymbol{\nu} r\colon G)\, Q}$$

  which concludes the case.

The proof of the second clause is also obtained by straightforward induction on the typing of $\Gamma \vdash N$, and analysing the structure of N. We discuss the following case

- Case $N \equiv s[P]$
  By hypothesis,
  $$\frac{\Gamma \vdash_s P\colon A}{\Gamma \vdash s[P]}$$
  holds. Using clause 1 we deduce $\Gamma, m\colon T \vdash_s P\colon A$, and thus $\Gamma, m\colon T \vdash s[P]$.

□

Lemma 15 (Strengthening lemma).

1. *If* $\Gamma, m\colon T \vdash_s P\colon A$, *m not in* Γ, *and* $m \notin \mathrm{fn}(P)$, *then* $\Gamma \vdash_s P\colon A$.
2. *If* $\Gamma, m\colon T \vdash N$, *m not in* Γ, *and* $m \notin \mathrm{fn}(N)$, *then* $\Gamma \vdash N$.

Proof. Case 1. is proved by induction on the typing of $\Gamma \vdash_t P\colon A$. We proceed by case analysis on the structure of P, and analyse the last typing rule applied. The proof is straightforward, so we present only some cases.

- P is $a!\langle b @ r\rangle$.
  Consequently, m is not a, b, r, or s. Then, still $\Gamma \vdash_s P\colon A$.
- P is $a?(x @ y)\ P$.
  By hypothesis,
  $$\frac{\begin{array}{c}\Gamma, m\colon T, x\colon C @ y, y\colon G \vdash_s P\colon A \\ \Gamma(a) <\colon \langle C @ G\rangle^{\mathsf{r}} @ s \qquad y \text{ not is } \Gamma\end{array}}{\Gamma, m\colon T \vdash_s a?(x @ y)\ P\colon A \cup \{\mathsf{installRes}\}}$$
  By induction hypothesis, $\Gamma, x\colon C @ y, y\colon G \vdash_s P\colon A$, and by rule P-Inpc, we conclude the case.

The proof for 2. is also obtained by straightforward induction on the typing of $\Gamma \vdash N$, and analysing the structure of N. We discuss the following case

- Case $N \equiv s[P]$.
  By hypothesis, $\Gamma, m\colon T \vdash [s]P$, then $\Gamma, m\colon T \vdash_s P\colon A$, that, using clause 1., we deduce $\Gamma \vdash_s P\colon A$, and thus $\Gamma \vdash s[P]$.

□

Lemma 16 (Congruence lemma). *If* $\Gamma \vdash N, N \equiv M$*, then* $\Gamma \vdash M$*.*

Proof. By induction on the typing of $\Gamma \vdash N$. We proceed by case analysis on each rule of the congruence relation defined in figure 4, inducting on the last rule applied.

- Case $(N \mid M) \mid M' \equiv N \mid (M \mid M')$, $M \mid N \equiv N \mid M$, or $N \mid \mathsf{stop} \equiv N$. All these three cases hold trivially. However, we illustrate how the proof can be done using the first case as and example. Since, by hypothesis, $\Gamma \vdash (N \mid M) \mid M'$, then the following derivation holds.

$$\dfrac{\dfrac{\Gamma \vdash N \quad \Gamma \vdash M}{\Gamma \vdash N \mid M} \qquad \Gamma \vdash M'}{\Gamma \vdash (N \mid M) \mid M'}$$

  Therefore,

$$\dfrac{\Gamma \vdash N \qquad \dfrac{\Gamma \vdash M \quad \Gamma \vdash M'}{\Gamma \vdash M \mid M'}}{\Gamma \vdash N \mid (M \mid M')}$$

- Case $(\boldsymbol{\nu} n\colon L_{@s})\ N \mid M \equiv (\boldsymbol{\nu} n\colon L_{@s})\ (N \mid M)$, for $n \notin \mathrm{fn}(M)$. By hypothesis the following type inference holds

$$\dfrac{\dfrac{\Gamma, n\colon T \vdash N \quad n \text{ not in } \Gamma}{\Gamma \vdash (\boldsymbol{\nu} n\colon T)\ N} \qquad \Gamma \vdash M}{\Gamma \vdash (\boldsymbol{\nu} n\colon T)\ N \mid M}$$

  Applying lemma 14 to $\Gamma \vdash M$, we conclude $\Gamma, n\colon T \vdash M$. Therefore, it is easy to show that $\Gamma \vdash (\boldsymbol{\nu} n\colon T)\ (N \mid M)$. The symmetric case is proved similarly, but using lemma 15 instead. Thus,

$$\dfrac{\dfrac{\Gamma, n\colon T \vdash N \quad \Gamma, n\colon T \vdash M}{\Gamma, n\colon T \vdash N \mid M} \qquad n \text{ not in } \Gamma}{\Gamma \vdash (\boldsymbol{\nu} n\colon T)\ (N \mid M)}$$

  holds by hypothesis. Hence,

$$\dfrac{\dfrac{\Gamma, n\colon T \vdash N \quad n \text{ not in } \Gamma}{\Gamma \vdash (\boldsymbol{\nu} n\colon T)\ N} \qquad \dfrac{\Gamma, n\colon T \vdash M \quad n \text{ not in } \Gamma \quad n \notin \mathrm{fn}(M)}{\Gamma \vdash M}\ (\text{lem } 15)}{\Gamma \vdash (\boldsymbol{\nu} n\colon T)\ N \mid M}$$

- Case $(\boldsymbol{\nu} n\colon T)\ (\boldsymbol{\nu} m\colon T')\ N \equiv (\boldsymbol{\nu} m\colon T')\ (\boldsymbol{\nu} n\colon T)\ N$, if m not in T, n not in T', and $n \neq m$.
  The type judgement, $\Gamma \vdash (\boldsymbol{\nu} n\colon T)\ (\boldsymbol{\nu} m\colon T')\ N$, holds by hypothesis. Therefore, we can derive the following type inference

$$\dfrac{\dfrac{\Gamma, n\colon T, m\colon T' \vdash N \quad m \text{ not in } \Gamma, n\colon T}{\Gamma, n\colon T \vdash (\boldsymbol{\nu} m\colon T')\ N} \quad n \text{ not in } \Gamma}{\Gamma \vdash (\boldsymbol{\nu} n\colon T)\ (\boldsymbol{\nu} m\colon T')\ N}$$

  Since n not in Γ, $n \neq m$, and n not in T', we conclude that n not in $\Gamma, m\colon T'$. By hypothesis, we know also that m not in $\Gamma, n\colon T$, which means that m not in Γ. Hence, the proof for $\Gamma \vdash (\boldsymbol{\nu} m\colon T')\ (\boldsymbol{\nu} n\colon T)\ N$ is as follows

$$\dfrac{\dfrac{\Gamma, n\colon T, m\colon T' \vdash N \quad n \text{ not in } \Gamma, m\colon T'}{\Gamma, m\colon T' \vdash (\boldsymbol{\nu} n\colon T)\ N} \quad m \text{ not in } \Gamma}{\Gamma \vdash (\boldsymbol{\nu} m\colon T')\ (\boldsymbol{\nu} n\colon T)\ N}$$

  The symmetric case holds as well, because we require also that m does not occur in T.

- $(\boldsymbol{\nu} n\colon L_{@s})\ s[P] \equiv s[(\boldsymbol{\nu} n\colon L)\ P]$, if $n \neq s$.
  From, $\Gamma \vdash (\boldsymbol{\nu} n\colon L_{@s})\ s[P]$, we derive

$$\dfrac{\dfrac{\Gamma, n\colon L_{@s} \vdash_s P\colon A}{\Gamma, L_{@s} \vdash s[P]} \quad n \text{ not in } \Gamma}{\Gamma \vdash (\boldsymbol{\nu} n\colon L_{@s})\ s[P]}$$

  Thus,

$$\dfrac{\dfrac{\dfrac{\Gamma, n\colon L_{@s} \vdash_s P\colon A \quad n \text{ not in } \Gamma}{\Gamma \vdash_s (\boldsymbol{\nu} n\colon L)\ :\ A \cup \{\mathsf{createRes}\}}}{\Gamma \vdash s[(\boldsymbol{\nu} n\colon L)\ P]}$$

  that finishes the case.

- $s[P] \mid s[Q] \equiv s[P \mid Q]$
  The subsequent type derivation holds by hypothesis.

$$\dfrac{\dfrac{\Gamma \vdash_s P\colon A}{\Gamma \vdash s[P]} \quad \dfrac{\Gamma \vdash_s Q\colon B}{\Gamma \vdash s[Q]}}{\Gamma \vdash s[P] \mid s[Q]}$$

  Hence, we are able to derive $\Gamma \vdash s[P \mid Q]$.

$$\dfrac{\dfrac{\Gamma \vdash_s P\colon A \quad \Gamma \vdash_s Q\colon B}{\Gamma \vdash_s P \mid Q\colon A \cup B}}{\Gamma \vdash s[P \mid Q]}$$

- $(\boldsymbol{\nu} n\colon T)\ \mathsf{stop} \equiv \mathsf{stop}$
  This case holds trivially because stop is well typed using any type environment.

- $(\boldsymbol{\nu} s\colon T)\ s[\mathsf{stop}] \equiv \mathsf{stop}$
  The case for $(\boldsymbol{\nu} s\colon T)\ s[\mathsf{stop}] \equiv \mathsf{stop}$ is trivial, since stop is well typed using any type environment. For the symmetric case, we can always α-convert s to not appear in Γ, and then the type derivation holds.

This concludes the proof. □

Lemma 17 (Subsuming lemma). *If* $\Gamma, n\colon T \vdash_s P$, *and* $T' <: T$, *then* $\Gamma, n\colon T' \vdash_s P$.

Proof. The proof is by induction on the typing of P. We proceed by analysing the structure of P.

- Case $P \equiv \mathsf{stop}$. The inaction network is typed with any well-formed environment. So, if $\Gamma, c\colon T \vdash_s \mathsf{stop}$, then $\Gamma, c\colon T'$ also types stop.

- Case $P \equiv a!\langle b @ r\rangle$. We analyse three subcases, namely, when we substitute channel a, channel b, or any other channel.

  - Case $c = a$. By hypothesis, $\Gamma, a\colon T \vdash_s a!\langle b @ r\rangle\colon \{\mathsf{useRes}\}$. Then, the following derivation holds.

$$\frac{\begin{array}{cc}\Gamma, a\colon C @ s \vdash \mathrm{env} & C @ s <: \langle C' @ G\rangle^{\mathsf{w}} @ s \\ \Gamma(r) = G & \Gamma(b) = C' @ r\end{array}}{\Gamma, a\colon C @ s \vdash_s a!\langle b @ r\rangle\colon \{\mathsf{useRes}\}}$$

    Since $T' = C'' @ s <: C @ s <: \langle C' @ G\rangle^{\mathsf{w}} @ s$, rule P-Outc concludes that $\Gamma, a\colon C'' @ s \vdash_s a!\langle b @ r\rangle\colon \{\mathsf{useRes}\}$

  - Case $c = b$. The following derivation holds by hypothesis.

$$\frac{\Gamma, b\colon C' @ r \vdash \mathrm{env} \qquad \Gamma(a) <: \langle C' @ G\rangle^{\mathsf{w}} @ s \qquad \Gamma(r) = G}{\Gamma, b\colon C' @ r \vdash_s a!\langle b @ r\rangle\colon \{\mathsf{useRes}\}}$$

    Since, by hypothesis, $T' = C'' @ r <: C' @ r$, and output channels are contravariant, then $\langle C' @ G\rangle^{\mathsf{w}} @ s <: \langle C'' @ G\rangle^{\mathsf{w}} @ s$. Therefore,

$$\frac{\begin{array}{cc}\Gamma(a) <: \langle C' @ G\rangle^{\mathsf{w}} @ s <: \langle C'' @ G\rangle^{\mathsf{w}} @ s \\ \Gamma, b\colon C'' @ r \vdash \mathrm{env} & \Gamma(r) = G\end{array}}{\Gamma, b\colon C'' @ r \vdash_s a!\langle b @ r\rangle\colon \{\mathsf{useRes}\}}$$

– Case $c \neq a$, and $c \neq b$. This case holds trivially, since the change in the types does not affect the typing derivation.

There is no point to consider the simultaneous substitution of channel a, and channel b, because we do not deal with recursive types. The substitution of r is also not taken into account, since there is no subtype relation defined for site types. The case for $P \equiv a!\langle\diamond\rangle$ is handled similarly to the present case.

- Case $P \equiv a?(x@y)\ P_1$. We only consider the case when $c = a$, since the case when $c \neq a$ holds trivially. By hypothesis, the following derivation holds.

$$\frac{\begin{array}{cc}\Gamma, C'@s, x\colon C@y, y\colon G \vdash_s P\colon A & C'@s <\colon \langle C@G\rangle^{\mathsf{r}}@s \\ \Gamma(r) = G & \Gamma(b) = C@r\end{array}}{\Gamma, a\colon C'@s \vdash_s a?(x@y)\ P\colon A \cup \{\mathsf{installRes}\}}$$

Since $T' = C''@s <\colon C'@s$, rule P-Inpc, concludes the case. The cases for $P \equiv a?(\diamond)\ P$, $P \equiv a?*(\diamond)\ P$, and $P \equiv a?*(x@y)\ P$, are handled similarly to the present case.

- Case $P \equiv P_1 \mid P_2$. By hypothesis, $\Gamma, c\colon T \vdash_s P_1 \mid P_2\colon A \cup B$, Therefore, using P-Par, $\Gamma, n\colon T \vdash_s P_1\colon A$, and $\Gamma, n\colon T \vdash_s P_2\colon B$. Since $T' <\colon T$, and by induction hypothesis, $\Gamma, n'\colon T \vdash_s P_1\colon A$, and $\Gamma, n\colon T' \vdash_s P_2\colon B$ that, by P-Par, concludes the case. The case for $P \equiv (\boldsymbol{\nu} n_1\colon T_1)\ P_1$ is handled similarly to the present case.

□

Lemma 18. *If $\Gamma \vdash_s P\colon A, \Gamma \sqcap n\colon T$ is well defined, then $\Gamma \sqcap n\colon T \vdash_s P\colon A$.*

Proof. We consider to cases, namely, when $n \in \mathrm{dom}(\Gamma)$, and when $n \notin \mathrm{dom}(\Gamma)$. If $n \in \mathrm{dom}(\Gamma)$, then $\Gamma(n) \sqcap n\colon T <\colon \Gamma(n)$, and by lemma 17, $\Gamma \setminus \{n\}, n\colon \Gamma(n) \sqcap T \vdash_s P\colon A$. If $n \notin \mathrm{dom}(\Gamma)$, then, by lemma 14, we conclude that $\Gamma, n\colon T \vdash_s P\colon A$. □

Next we prove the main result of this section.

Theorem 1 (Subject reduction). *If $\Gamma \vdash N, N \to M$, then $\Gamma \vdash M$.*

Proof. By induction on the typing derivation of $\Gamma \vdash N$. We proceed by case analysis on the reduction relation, and examine the last typing rule of the typing derivation.

1. Case $s[a!\langle\diamond\rangle] \mid s[a?(\diamond)\ P] \to s[P]$.
   Since $\Gamma \vdash s[a!\langle\diamond\rangle] \mid s[a?(\diamond)\ P]$, then the following type derivation holds.

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash \text{env} \quad \Gamma(a) <: \langle \mathsf{unit} \rangle^{\mathsf{w}}{}_{@}s}{\Gamma \vdash_s a!\langle\diamond\rangle : \{\mathsf{useRes}\}}}{\Gamma \vdash s[a!\langle\diamond\rangle]} \qquad \dfrac{\dfrac{\Gamma \vdash_s P : A \quad \Gamma(a) <: \langle \mathsf{unit} \rangle^{\mathsf{r}}{}_{@}s}{\Gamma \vdash_s a?(\diamond)\ P : A \cup \{\mathsf{installRes}\}}}{\Gamma \vdash s[a?(\diamond)\ P]}}{\Gamma \vdash s[a!\langle\diamond\rangle] \mid s[a?(\diamond)\ P]}$$

   Hence, the proof for $\Gamma \vdash s[P]$ follows directly from the fact that $\Gamma \vdash_s P : A$ holds by hypothesis.

2. Case $s[a!\langle b_{@}r\rangle] \mid s[a?(x_{@}y)\ P] \to s[P\{r/y\}\{b/x\}]$.
   The proof for $\Gamma \vdash s[a!\langle b_{@}r\rangle] \mid s[a?(x_{@}y)\ P]$ is

$$\dfrac{\Gamma \vdash s[a!\langle b_{@}r\rangle] \qquad \Gamma \vdash s[a?(x_{@}y)\ P]}{\Gamma \vdash s[a!\langle b_{@}r\rangle] \mid s[a?(x_{@}y)\ P]}$$

   where,

$$\dfrac{\dfrac{\Gamma \vdash \text{env} \quad \Gamma(a) <: \langle C_{@}G \rangle^{\mathsf{w}}{}_{@}s \quad \Gamma(r) = G \quad \Gamma(b) = C_{@}r}{\Gamma \vdash_s a!\langle b_{@}r\rangle : \{\mathsf{useRes}\}}}{\Gamma \vdash s[a!\langle b_{@}r\rangle]}$$

   and

$$\dfrac{\dfrac{\Gamma, y : G', x : C'_{@}y \vdash_s P : A \quad \Gamma(a) <: \langle C'_{@}G' \rangle^{\mathsf{r}}{}_{@}s}{\Gamma \vdash_s a?(x_{@}y)\ P : A \cup \{\mathsf{installRes}\}}}{\Gamma \vdash s[a?(x_{@}y)\ P]}$$

   Hence, $\Gamma(a) <: \langle C_{@}G \rangle^{\mathsf{w}}{}_{@}s$, and $\Gamma(a) <: \langle C'_{@}G' \rangle^{\mathsf{r}}{}_{@}s$, so a must be a read/write channel. Let $\Gamma(a) = \langle C''_{@}G'' \rangle^{\mathsf{rw}}$. Therefore $\langle C''_{@}G'' \rangle^{\mathsf{rw}}{}_{@}s <: \langle C_{@}G \rangle^{\mathsf{w}}{}_{@}s$, which means that $C_{@}G <: C''_{@}G''$. From $\langle C''_{@}G'' \rangle^{\mathsf{rw}}{}_{@}s <: \langle C'_{@}G' \rangle^{\mathsf{r}}{}_{@}s$, we conclude that $C''_{@}G'' <: C'_{@}G'$. By transitivity, $C_{@}G <: C''_{@}G'' <: C'_{@}G'$.

   Thus, using the hypothesis, $\Gamma(b) = C_{@}r$, $C <: C'$, $\Gamma(r) = G \subseteq G'$, and $\Gamma, y : G, x : C'_{@}y \vdash_s P : A$. So, we meet the conditions to apply the substitution lemma (lemma 13), which yields $\Gamma \vdash_s P\{r/y\}\{b/x\} : A$, since y (bound) does not appear in Γ, and is different from s (free). Hence, we conclude that $\Gamma \vdash s[P\{r/y\}\{b/x\}]$.

3. Case $s[a!\langle\diamond\rangle] \mid s[a?*(\diamond)\ P] \to s[P] \mid s[a?*(\diamond)\ P]$.
   The proof for this case follows a pattern similar to case 1. The type judgement $\Gamma \vdash s[P] \mid s[a?*(\diamond)\ P]$ is a consequence of $\Gamma \vdash_s P : A$, and $\Gamma \vdash_s a?(\diamond)\ P : A \cup \{\mathsf{installRes}\}$, that hold by hypothesis.

4. Case $s[a!\langle b_{@}r\rangle] \mid s[a?*(x_{@}y)\ P] \to s[P\{r/y\}\{b/x\}] \mid s[a?*(x_{@}y)\ P]$.
From the proof of $\Gamma \vdash s[a!\langle b_{@}r\rangle] \mid s[a?*(x_{@}y)\ P]$, we deduce that:

   (a) $\Gamma \vdash_s a!\langle b_{@}r\rangle : \{\mathsf{useRes}\}$,

   (b) $\Gamma \vdash s[a?*(x_{@}y)\ P]$, and

   (c) $\Gamma \vdash_s a?(x_{@}y)\ P : A \cup \{\mathsf{installRes}\}$.

   From (a) and (c), using the proof for case 2, we conclude that (d) $\Gamma \vdash s[P\{r/y\}\{b/x\}]$, and using (b), and (d), we establish that $\Gamma \vdash s[P\{r/y\}\{b/x\}] \mid s[a?*(x_{@}y)\ P]$.

5. Case $s[\mathsf{goto}\ r.P] \to r[P]$.
   By hypothesis, the following derivation holds.

$$\dfrac{\dfrac{\Gamma \vdash_r P : A}{\dfrac{\Gamma \vdash r\ \mathsf{allows}\ s : A \qquad \Gamma \vdash s\ \mathsf{canEnter}\ r}{\Gamma \vdash_s \mathsf{goto}\ r.P : \{\mathsf{go}\}}}}{\Gamma \vdash s[\mathsf{goto}\ r.P]}$$

   Thus, $\Gamma \vdash r[P]$ holds from $\Gamma \vdash_r P : A$.

6. Case $\dfrac{N \to M}{(\boldsymbol{\nu} n : L_{@}s)\ N \to (\boldsymbol{\nu} n : L_{@}s)\ M}$ and $\dfrac{N \to N'}{N \mid M \to N' \mid M}$.

   These two cases are handled similarly, and follow by direct application of the induction hypothesis. We illustrate the proof reasoning about the first case. By hypothesis $\Gamma \vdash (\boldsymbol{\nu} n : L_{@}s)\ N$, therefore we can derive

$$\dfrac{\Gamma, n : L_{@}s \vdash N \qquad n \text{ not in } \Gamma}{\Gamma \vdash (\boldsymbol{\nu} n : L_{@}s)\ N}$$

   Since $\Gamma, n : L_{@}s \vdash N$ and $N \to N'$, then, by induction hypothesis, $\Gamma, n : L_{@}s \vdash N'$, and thus,

$$\dfrac{\Gamma, n : L_{@}s \vdash N' \qquad n \text{ not in } \Gamma}{\Gamma \vdash (\boldsymbol{\nu} n : L_{@}s)\ N'}.$$

7. Case $\dfrac{N \equiv N' \qquad N' \to M' \qquad M' \equiv M}{N \to M}$.

   By hypothesis, $\Gamma \vdash N$ and $N \equiv N'$, then, by lemma 16, $\Gamma \vdash N'$. Applying induction hypothesis to $\Gamma \vdash N'$, and to $N' \to M'$, we conclude that $\Gamma \vdash M'$. Finally, using lemma 16 applied to $\Gamma \vdash M'$, and to $M' \equiv M$, we deduce that $\Gamma \vdash M$.

This concludes the proof. □

A.2 Proofs from section 4

A.2.1 Proofs from subsection 4.1

Lemma 19. $\Gamma \vdash N$, *if and only if,* $\mathsf{tag}_\Gamma(N) \neq \emptyset$.

Proof. We prove that $\Gamma \vdash N$, then $\mathsf{tag}_\Gamma(N) \neq \emptyset$ by induction on the structure of N.

- Case $N \equiv \mathsf{stop}$. By definition, $\mathsf{tag}_\Gamma(\mathsf{stop}) = \{\mathsf{stop}\} \neq \emptyset$.
- Case $N \equiv N_1 \mid N_2$. Since, by hypothesis, $\Gamma \vdash N_1 \mid N_2$, then

$$\frac{\Gamma \vdash N_1 \qquad \Gamma \vdash N_2}{\Gamma \vdash N_1 \mid N_2}$$

  Hence, by induction hypothesis, $\mathsf{tag}_\Gamma(N_1) \neq \emptyset$, and $\mathsf{tag}_\Gamma(N_2) \neq \emptyset$. Thus, $\mathsf{tag}_\Gamma(N_1 \mid N_2) = \{M_1 \mid M_2 \text{ s.t. } M_i \in \mathsf{tag}\,\Gamma(N_i)\} \neq \emptyset$.

- Case $N \equiv (\boldsymbol{\nu} s\colon G)\ N_1$. The following derivation holds.

$$\frac{\Gamma, s\colon G \vdash N_1}{\Gamma \vdash (\boldsymbol{\nu} s\colon G)\ N_1}$$

  Therefore, by induction hypothesis, $\mathsf{tag}_{\Gamma, \mathsf{s}\colon \mathsf{G}}(N_1) \neq \emptyset$. Hence,

$$\mathsf{tag}_\Gamma((\boldsymbol{\nu} s\colon G)\ N_1) = \{(\boldsymbol{\nu}_t\, s\colon G)\ M \text{ s.t. } M\colon \mathsf{tag}_{\Gamma, \mathsf{s}\colon \mathsf{G@s}}(N)\} \neq \emptyset.$$

  The case for $N \equiv (\boldsymbol{\nu} g\colon (R, G))\ N_1$ is handled similarly.

- Case $N \equiv (\boldsymbol{\nu} a\colon C@s)\ N_1$. Then, $\Gamma, a\colon C@s \vdash N_1$ is guarantied by hypothesis. Induction hypothesis assures that $\mathsf{tag}_{\Gamma, \mathsf{a}\colon \mathsf{C@s}}(N_1) \neq \emptyset$. By definition $\mathsf{tag}_\Gamma(N) = \{(\boldsymbol{\nu}_t\, a\colon C@s)\ M \text{ s.t. } M \in \mathsf{tag}_{\Gamma, \mathsf{a}\colon \mathsf{C@s}}(N_1), \text{ and } \Gamma \vdash s \ \mathsf{allows}\ t\colon \{\mathsf{createRes}\}\}$. Since a process needs no specific authorisation to create a resource at its host site ($\Gamma \vdash s \ \mathsf{allows}\ s\colon \{\mathsf{createRes}\}$ is always true), the network $(\boldsymbol{\nu}_s\, a\colon C@s)\ M \in \mathsf{tag}_\Gamma(N)$, which concludes the case.

- Case $N \equiv s[P]$. Then, $\Gamma \vdash_s P\colon A$, since $\Gamma \vdash [s]P$. On the other hand, the following derivation holds for the tagged type system.

$$\frac{\Gamma \vdash_s P\colon A \qquad \Gamma <\colon \Gamma \qquad \Gamma \vdash s \ \mathsf{allows}\ s\colon A \qquad \Gamma \vdash s \ \mathsf{canEnter}\ s}{\Gamma \Vdash s[P]^s_\Gamma}$$

  Therefore, $s[P]^s_\Gamma \in \mathsf{tag}_\Gamma(s[P])$, proving that $\mathsf{tag}_\Gamma(s[P]) \neq \emptyset$, and concluding the first part of the proof.

The prove that $\mathsf{tag}_\Gamma(N) \neq \emptyset$ implies $\Gamma \vdash N$, proceeds, also, by induction on the structure of N.

- Case $N \equiv \mathsf{stop}$. The stop network is typed under any well-formed environment, and therefore if $\mathsf{tag}_\Gamma(N) = \{\mathsf{stop}\} \neq \emptyset$, then $\Gamma \vdash \mathsf{stop}$.

- $N \equiv N_1 \,|\, N_2$. Since, by hypothesis, $\mathsf{tag}_\Gamma(N) \neq \emptyset$, then $\mathsf{tag}_\Gamma(N_1) \neq \emptyset$, and $\mathsf{tag}_\Gamma(N_2) \neq \emptyset$, because $\mathsf{tag}_\Gamma(N_1 \,|\, N_2) = \{M_1 \,|\, M_2 \text{ s.t. } M_i \in \mathsf{tag}_\Gamma(N_i)\}$. Therefore, by induction hypothesis, $\Gamma \vdash N_1$, and $\Gamma \vdash N_2$, so does $\Gamma \vdash N_1 \,|\, N_2$.

- Case $N \equiv s[P]$. By definition,

$$\mathsf{tag}_\Gamma(s[P]) = \{s[P]^t_\Delta, \text{ s.t. } \Gamma <: \Delta, \quad \Delta \vdash_s P : A, \quad \Delta \vdash s \;\mathsf{allows}\; t : A, \quad \Delta \vdash t \;\mathsf{canEnter}\; s\} \neq \emptyset.$$

  Then, the following derivation holds.

$$\dfrac{\dfrac{\Delta \vdash_s P : A \qquad \Gamma <: \Delta}{\Gamma \vdash_s P : A}}{\Gamma \vdash s[P]}$$

- Case $N \equiv (\boldsymbol{\nu} s : G)\; N_1$. By definition

$$\mathsf{tag}_\Gamma((\boldsymbol{\nu} s : G)\; N_1) = \{(\boldsymbol{\nu}_t\, s : G)\; M \text{ s.t. } M \in \mathsf{tag}_{\Gamma, \mathsf{s}:\, \mathsf{G}}(N_1)\} \neq \emptyset.$$

  Then, $\mathsf{tag}_{\Gamma, \mathsf{s}:\, \mathsf{G}}(N_1) \neq \emptyset$, and, by induction hypothesis, $\Gamma, s : G \vdash N_1$. Therefore, $\Gamma \vdash (\boldsymbol{\nu} s : G)\; N$.

  The cases for $N \equiv (\boldsymbol{\nu} g : (R, G))\; N_1$, and $N \equiv (\boldsymbol{\nu} a : C_{@s})\; N_1$ are proved in a similar way.

and, we conclude the proof. □

Lemma 20. *If $M \in \mathsf{tag}_\Gamma(N)$, then $\Gamma \Vdash M$.*

Proof. We prove by induction on the structure of N.

- Case $N \equiv \mathsf{stop}$. By definition, $\mathsf{tag}_\Gamma(\mathsf{stop}) = \{\mathsf{stop}\}$, and $\Gamma \Vdash \mathsf{stop}$, because any well-formed environment type the inaction network.

- Case $N \equiv s[P]$. By hypothesis, $M \in \mathsf{tag}_\Gamma(s[P]) = \{s[P]^t_\Delta \text{ s.t. } \Gamma <: \Delta,\ \Delta \vdash_s P : A,\ \Delta \vdash s \;\mathsf{allows}\; t : A,\ \Delta \vdash t \;\mathsf{canEnter}\; s\}$. Let $M \equiv_T s[P]^t_\Delta$. Then, the following derivation holds.

$$\dfrac{\dfrac{\Delta \vdash_s P : A}{\Delta \vdash s \;\mathsf{allows}\; t : A} \qquad \dfrac{\Gamma <: \Delta}{\Delta \vdash t \;\mathsf{canEnter}\; s}}{\Gamma \Vdash s[P]^t_\Delta}$$

- Case $N \equiv N_1 \mid N_2$. Then, $M \in \mathsf{tag}_\Gamma(N_1 \mid N_2) = \{M_1 \mid M_2 \text{ s.t. } M_i \in \mathsf{tag}\,\Gamma(N_i)\}$. By induction hypothesis, $\Gamma \Vdash M_1$, and $\Gamma \Vdash M_2$, and therefore, by rule T-Par, we conclude that $\Gamma \Vdash M_1 \mid M_2$.

- Case $N \equiv (\boldsymbol{\nu} a\colon C@s)\ N_1$. By hypothesis, $M \in \mathsf{tag}_\Gamma((\boldsymbol{\nu} a\colon C@s)\ N_1) = \{(\boldsymbol{\nu}_t\, a\colon C@s)\ N_1,\ \text{s.t. } N_1 \in \mathsf{tag}_{\Gamma, \mathsf{a}\colon \mathsf{C@s}}(N),\ \text{and } \Gamma \vdash s\ \mathsf{allows}\ t\colon\ \mathsf{createRes}\}$. By induction hypothesis, $\Gamma, a\colon C@s \Vdash N_1$, and therefore

$$\frac{\Gamma, a\colon C@s \Vdash N_1 \qquad \Gamma \vdash s\ \mathsf{allows}\ t\colon\ \mathsf{createRes}}{\Gamma \Vdash (\boldsymbol{\nu}_t\, a\colon C@s)\ N}$$

  The proofs for the creation of sites, and the creation of groups are handled similarly to the present case.

□

Lemma 21. *Let* $\Gamma \vdash N$.

(i) If $N \equiv N'$, *then* $\exists N_1 \in \mathsf{tag}_\Gamma(N)$ *s.t.* $N_1 \equiv_T N_1' \in \mathsf{tag}_\Gamma(N')$.

(ii) If $N_1 \in \mathsf{tag}_\Gamma(N) \equiv_T N_1'$, *then* $\exists N'$ *s.t.* $N \equiv N'$, *and* $N_1' \in \mathsf{tag}_\Gamma(N')$.

Proof. The proof of (i) is by induction on the definition of the structural congruence relation ($\equiv$). We proceed by analysing each reduction rule.

- Case $M \mid N \equiv N \mid M$. By hypothesis, $\Gamma \vdash N$, and $\Gamma \vdash M$; by lemma 19, we conclude that $\mathsf{tag}_\Gamma(N) \neq \emptyset$, and $\mathsf{tag}_\Gamma(M) \neq \emptyset$. Then, there exists $M_1 \mid N_1 \in \mathsf{tag}_\Gamma(M \mid N)$, which is congruent to $N_1 \mid M_1$ (by definition of $\equiv_T$).

  The cases for $M \mid (N \mid N') \equiv (M \mid N) \mid N'$, and $(N \mid \mathsf{stop}) \equiv N$ are handled in a similar way.

- Case $s[P] \mid s[Q] \equiv s[P \mid Q]$. So, the following derivation holds.

$$\frac{\Gamma \vdash_s P\colon A \qquad \Gamma \vdash_s Q\colon Q}{\Gamma \vdash s[P] \mid s[Q]}$$

  The network $s[P]^s_\Gamma \in \mathsf{tag}_\Gamma(s[P])$, since $\Gamma <: \Gamma$, $\Gamma \vdash s\ \mathsf{allows}\ s\colon A$, and $\Gamma \vdash s\ \mathsf{canEnter}\ s$, as well as, $s[Q]^s_\Gamma \in \mathsf{tag}_\Gamma(s[Q])$. Therefore, $s[P]^s_\Gamma \mid s[Q]^s_\Gamma \in \mathsf{tag}_\Gamma(s[P] \mid s[Q])$.

  By definition, $s[P]^s_\Gamma \mid s[Q]^s_\Gamma \equiv_T s[P \mid Q]^s_\Gamma$, which belongs to $\mathsf{tag}_\Gamma(s[P \mid Q])$, since $\Gamma \vdash_s P\colon A$, and $\Gamma \vdash_s Q\colon B$ implies $\Gamma \vdash_s P \mid Q\colon A \cup B$, $\Gamma <: \Gamma$, $\Gamma \vdash s\ \mathsf{allows}\ s\colon A \cup B$, and $\Gamma \vdash s\ \mathsf{canEnter}\ s$.

The remaining cases are simple to handle.

The proof of (ii) is by induction on the definition of the structural congruence relation ($\equiv_T$). We proceed by analysing each reduction rule.

- Case $M_1 \mid M_2 \equiv_T M_2 \mid M_1$. By hypothesis, $M_1 \mid M_2 \in \mathsf{tag}_\Gamma(N)$. Then N is of the form $N_1 \mid N_2$, and, by definition of $\equiv$, there exists $N_2 \mid N_1 \equiv N_1 \mid N_2$. Since $M_1 \in \mathsf{tag}_\Gamma(N_1)$, and $M_2 \in \mathsf{tag}_\Gamma(N_2)$ by definition of $\mathsf{tag}_\Gamma(N)$, we conclude that $M_2 \mid M_1 \in \mathsf{tag}_\Gamma(N_2 \mid N_1)$.

  The cases for $M_1 \mid (M_2 \mid M_3) \equiv_T (M_1 \mid M_2) \mid M_3$, and $(M \mid \mathsf{stop}) \equiv M$ are handled in a similar way.

- Case $s[P]^t_\Gamma \mid s[Q]^t_\Gamma \equiv_T s[P \mid Q]^t_\Gamma$. Since, $s[P]^t_\Gamma \mid s[Q]^t_\Gamma \in \mathsf{tag}_\Gamma(N)$, then N is of the form $s[P] \mid s[Q]$, which is congruent to $s[P \mid Q]$, and $\Gamma \vdash_s P\colon A$, $\Gamma \vdash_s Q\colon B$, $\Gamma \vdash s$ allows $t\colon A$, $\Gamma \vdash s$ allows $t\colon B$, and $\Gamma \vdash t$ $\mathsf{canEnter}$ s. Consequently, $s[P \mid Q]^t_\Gamma \in \mathsf{tag}_\Gamma(s[P \mid Q]$.

  The remaining cases are simple to handle.

□

Theorem 2. *Let* $\Gamma \vdash N$.

(i) If $N \to N'$, *then* $\exists M \in \mathsf{tag}_\Gamma(N)$ *s.t.* $M \mapsto M' \in \mathsf{tag}_\Gamma(N')$.

(ii) If $M \in \mathsf{tag}_\Gamma(N) \mapsto M'$, *then* $\exists N'$ *s.t.* $N \to N'$, *and* $M' \in \mathsf{tag}_\Gamma(N')$.

Proof. The proof of (i) is by induction on the definition of reduction ($\to$). We proceed by analysing each reduction rule.

- Case $s[a!\langle b @ r\rangle] \mid s[a?(x @ y)\ P] \to s[P\{r/y\}\{b/x\}]$. By hypothesis, $\Gamma \vdash s[a!\langle b @ r\rangle] \mid s[a?(x @ y)\ P]$. Then, the following derivation holds.

$$\dfrac{\dfrac{\dfrac{\cdots}{\Gamma \vdash_s a!\langle b @ r\rangle\colon \{\mathsf{useRes}\}}}{\Gamma \vdash s[a!\langle b @ r\rangle]} \qquad \dfrac{\dfrac{\cdots}{\Gamma \vdash_s a?(x @ y)\ P\colon A}}{\Gamma \vdash s[a?(x @ y)\ P]}}{\Gamma \vdash s[a!\langle b @ r\rangle] \mid s[a?(x @ y)\ P]}$$

  By definition of the tag function,

$$\mathsf{tag}_\Gamma(s[a!\langle b @ r\rangle]) = \{s[a!\langle b @ r\rangle]^{t_1}_{\Delta_1} \quad \text{s.t.} \quad \Delta_1 \vdash_s a!\langle b @ r\rangle\colon \{\mathsf{useRes}\},\ \Gamma <: \Delta_1,\quad \Delta_1 \vdash s\ \mathsf{allows}\ t_1\colon \{\mathsf{useRes}\},\quad \Delta_1 \vdash t_1\ \mathsf{canEnter}\ s\}$$

$$\mathsf{tag}_\Gamma(s[a?(x @ y)\ P]) = \{s[a?(x @ y)\ P]^{t_2}_{\Delta_2} \quad \text{s.t.} \quad \Delta_2 \vdash_s a?(x @ y)\ P\colon A,\ \Gamma <: \Delta_2,\quad \Delta_2 \vdash s\ \mathsf{allows}\ t_2\colon A,\quad \Delta_2 \vdash t_2\ \mathsf{canEnter}\ s\}$$

Therefore, $s[a!\langle b@r\rangle]^s_\Gamma \in \mathsf{tag}_\Gamma(s[a!\langle b@r\rangle])$, because $\Gamma \vdash_s a!\langle b@r\rangle$ by the above derivation; $\Gamma <: \Gamma$, $\Gamma \vdash s$ allows $s\colon \{\mathsf{useRes}\}$, and $\Gamma \vdash s$ canEnter s, since $<:$, allows, and canEnter are reflexive. Using similar arguments, we can conclude that $s[a?(x@y)\ P]^s_\Gamma \in \mathsf{tag}_\Gamma(s[a?(x@y)\ P])$ as well.

Hence, $s[a!\langle b@r\rangle]^s_\Gamma \mid s[a?(x@y)\ P]^s_\Gamma \in \mathsf{tag}_\Gamma(s[a!\langle b@r\rangle \mid a?(x@y)\ P])$.

By tagged reduction,

$$s[a!\langle b@r\rangle]^s_\Gamma \mid s[a?(x@y)\ P]^s_\Gamma \mapsto s[P\{b/x\}\{r/y\}]^s_\Gamma.$$

The network $s[P\{b/x\}\{r/y\}]^s_\Gamma \in \mathsf{tag}_\Gamma(s[P\{r/y\}\{b/x\}])$, because (1) the subject reduction theorem ensures that $\Gamma \vdash s[P\{b/x\}\{r/y\}]$, and thus, $\Gamma \vdash_s P\{b/x\}\{r/y\}\colon A$; and (2) $\Gamma <: \Gamma$, $\Gamma \vdash s$ allows $s\colon A$, and $\Gamma \vdash s$ canEnter s is assured by reflexivity of $<:$, allows, and canEnter relations.

Rules $\textsc{Comc}_2$, $\textsc{Comr}_1$, and $\textsc{Comr}_2$ are proved using the similar arguments to the just stated.

- Case $s[\mathsf{goto}\ r.P] \to r[P]$. The following derivation holds by hypothesis.

$$\frac{\dfrac{\Gamma \vdash_r P\colon A \quad \Gamma \vdash r \text{ allows } s\colon A \quad \Gamma \vdash s \text{ canEnter } r}{\Gamma \vdash_s \mathsf{goto}\ r.P\colon \{\mathsf{go}\}}}{\Gamma \vdash s[\mathsf{goto}\ r.P]}$$

By definition of the tag function,

$$\begin{aligned}\mathsf{tag}_\Gamma(s[\mathsf{goto}\ r.P]) = \{&s[\mathsf{goto}\ r.P]^t_\Delta, \quad \text{s.t.} \quad \Delta \vdash \mathsf{goto}\ r.P\colon \{\mathsf{go}\},\\ &\Gamma <: \Delta, \quad \Delta \vdash s \text{ allows } t\colon \{\mathsf{go}\}, \quad \Delta \vdash t \text{ canEnter } s\}\end{aligned}$$

The network $s[\mathsf{goto}\ r.P]^s_\Gamma \in \mathsf{tag}_\Gamma(s[\mathsf{goto}\ r.P])$, since, by the above derivation, $\Gamma \vdash_s \mathsf{goto}\ r.P\colon \{\mathsf{go}\}$, and $<:$, allows, and canEnter are reflexive.

By tagged reduction, $s[\mathsf{goto}\ r.P]^s_\Gamma \mapsto r[P]^s_\Gamma \in \mathsf{tag}_\Gamma(r[P])$, because, by subject reduction, $\Gamma \vdash r[P]$, and therefore $\Gamma \vdash_r P\colon A$. From the derivation of $\Gamma \vdash [s]\mathsf{goto}\ .rP$, presented above, it follows that $\Gamma \vdash r$ allows $s\colon A$, and $\Gamma \vdash s$ canEnter r.

- Case the last reduction rule applied is $\textsc{Res}$. By hypothesis, $\Gamma, a\colon C@s \vdash N$, since $\Gamma \vdash (\boldsymbol{\nu} a\colon C@s)\ N$. By induction hypothesis, and from $N \to N'$, there is a $N_1 \in \mathsf{tag}_{\Gamma,\mathsf{a}\colon \mathsf{C@s}}(N)$ s.t. $N_1 \mapsto N'_1 \in \mathsf{tag}_{\Gamma,\mathsf{a}\colon \mathsf{C@s}}(N')$. Therefore, $(\boldsymbol{\nu}_s\, a\colon C@s)\ N_1 \in \mathsf{tag}_\Gamma((\boldsymbol{\nu} a\colon C@s)\ N)$.

  By tag reduction, and by definition of allows function, $(\boldsymbol{\nu}_s\, a\colon C@s)\ N_1 \mapsto (\boldsymbol{\nu}_s\, a\colon C@s)\ N'_1 \in \mathsf{tag}_\Gamma((\boldsymbol{\nu} a\colon C@s)\ N')$.

- Case the last reduction rule applied is Par. The following derivation holds.

$$\frac{\Gamma \vdash N \qquad \Gamma \vdash M}{\Gamma \vdash N \mid M}$$

Since, $\Gamma \vdash N$, and, by hypothesis, $N \to N'$, then, by induction hypothesis, there exists $N_1 \in \mathsf{tag}_\Gamma(N)$ s.t. $N_1 \mapsto N_1'$, and $N_1' \in \mathsf{tag}_\Gamma(N')$.

By definition, $\mathsf{tag}_\Gamma(N \mid M) = \{N_1 \mid M_1 \text{ s.t. } N_1 \in \mathsf{tag}\,\Gamma(N), \text{ and } M_1 \in \mathsf{tag}\,\Gamma(M)\}$. Since $\Gamma \vdash M$, then, using lemma 19, we conclude that $\mathsf{tag}_\Gamma(M) \neq \emptyset$. Let $N_1 \mid M_1 \in \mathsf{tag}_\Gamma(N \mid M)$. Therefore, $N_1 \mapsto N_1'$, implies $N_1 \mid M_1 \mapsto N_1' \mid M_1$, having $N_1' \in \mathsf{tag}_\Gamma(N')$ and $M_1 \in \mathsf{tag}_\Gamma(M_1)$. Hence, $N_1' \mid M_1 \in \mathsf{tag}_\Gamma(N' \mid M)$.

- Case the last deduction rule applied is Str. The rule states that

$$\frac{N \equiv N' \qquad N' \to M' \qquad M' \equiv M}{N \to M}$$

Since $N \equiv N'$, and by lemma 21, there exists $N_1 \in tagf\Gamma(N)$, s.t. $N_1 \equiv_T N_1'$, and $N_1' \in \mathsf{tag}_\Gamma(N')$. By congruence lemma 16 $\Gamma \vdash N'$, and by induction hypothesis, from $\Gamma \vdash N'$ and $N' \to M'$, we conclude that there exists $N_1' \in \mathsf{tag}_\Gamma(N')$ s.t. $N_1' \mapsto M_1'$ and $M_1' \in \mathsf{tag}_\Gamma(M')$. By subject reduction, theorem 1, $\Gamma \vdash M'$. Again, by lemma 21, there exists $M_1' \in \mathsf{tag}_\Gamma(M')$ s.t. $M_1' \equiv M_1$, with $M_1 \in \mathsf{tag}_\Gamma(M)$.

This concludes the proof for (i).

The proof for (ii) is by induction on the definition of reduction $(\mapsto)$. We proceed by analysing each tagged reduction rule.

- Case $s[a!\langle\diamond\rangle]^t_\Sigma \mid s[a?(\diamond)\ P]^u_\Delta \mapsto s[P]^u_\Delta$. By hypothesis, $M \in \mathsf{tag}_\Gamma(N)$, which means that N is of the form $s[a!\langle\diamond\rangle] \mid s[a?(x_{@}y)\ P]$, and that, by lemma 20, $\Gamma \Vdash s[a!\langle\diamond\rangle]^t_\Sigma \mid s[a?(\diamond)\ P]^u_\Delta$. Consequently, $N \to s[P] \equiv N'$, and the following derivation holds.

$$\frac{\dfrac{\Sigma \vdash_s a!\langle\diamond\rangle : \{\mathsf{useRes}\} \quad \Gamma <: \Sigma}{\Sigma \vdash s\ \mathsf{allows}\ t : \{\mathsf{useRes}\}}}{\Sigma \vdash t\ \mathsf{canEnter}\ s} \qquad \frac{\dfrac{\dfrac{\Delta \vdash_s P : A \quad \Delta(a) <: \langle\diamond\rangle^{\mathsf{w}} s}{\Delta \vdash_s a?(\diamond)\ P : A \cup \{\mathsf{installRes}\}}}{\Gamma <: \Delta}}{\Delta \vdash s\ \mathsf{allows}\ u : A \cup \{\mathsf{installRes}\}}}{\Delta \vdash u\ \mathsf{canEnter}\ s}$$

$$\frac{\Gamma \Vdash s[a!\langle\diamond\rangle]^t_\Sigma \qquad \Gamma \Vdash s[a?(\diamond)\ P]^u_\Delta}{\Gamma \Vdash s[a!\langle\diamond\rangle]^t_\Sigma \mid s[a?(\diamond)\ P]^u_\Delta}$$

By definition, $\mathsf{tag}_\Gamma(s[P]) = \{s[P]^v_\Lambda \text{ s.t. } \Lambda \vdash_s P : A,\ \Gamma <: \Lambda,\ \Lambda \vdash s\ \mathsf{allows}\ b : A,\ \Lambda \vdash v\ \mathsf{canEnter}\ s\}$. Therefore, $s[P]^u_\Delta \in \mathsf{tag}_\Gamma(s[P])$ is assured by $\Gamma \Vdash s[a?(\diamond)\ P]^u_\Delta$.

Rule T-Comr$_2$ for replicated input of basic values is proved using the same reason as for the present case.

- Case $s[a!\langle b@r\rangle]^t_{\Sigma,r\colon G,b\colon T} \mid s[a?(x@y)\ P]^u_\Delta \mapsto s[P\{r/y\}\{b/x\}]^u_{\Delta\sqcap r\colon G\sqcap b\colon T}$. Since $M \in \mathsf{tag}_\Gamma(N)$, then N is of the form $s[a!\langle b@r\rangle] \mid s[a?(x@y)\ P]$, which reduces to $s[P\{r/y\}\{b/x\}]$, and, by lemma 20, $\Gamma \Vdash M$.

  We show that $s[P\{r/y\}\{b/x\}]^u_{\Delta\sqcap r\colon G\sqcap b\colon T} \in \mathsf{tag}_\Gamma(s[P\{r/y\}\{b/x\}])$. By definition of tagging,

$$\mathsf{tag}_\Gamma(s[P\{r/y\}\{b/x\}]) = \{s[P\{r/y\}\{b/x\}]^r_\Lambda \text{ s.t. } \Gamma <\colon \Lambda,\ \Lambda \vdash_s P\{r/y\}\ \{b/x\}\colon A,\ \Lambda \vdash s\ \mathsf{allows}\ t\colon A,\ \Lambda \vdash t\ \mathsf{canEnter}\ s\}$$

  Since $\Gamma \Vdash M$, and $M \mapsto M'$, then, by subject reduction, theorem 4, $\Gamma \Vdash M'$, and the following derivation holds.

$$\frac{\begin{array}{cc}\Delta\sqcap r\colon G\sqcap b\colon T \vdash_s P\{r/y\}\{b/x\}\colon A & \Gamma <\colon \Delta\sqcap r\colon G\sqcap b\\ \Delta\sqcap r\colon G\sqcap b\vdash s\ \mathsf{allows}\ u\colon A & \Delta\sqcap r\colon G\sqcap b\vdash u\ \mathsf{canEnter}\ t\end{array}}{\Gamma \Vdash s[P\{r/y\}\{r/x\}]^u_{\Delta\sqcap r\colon G\sqcap b\colon T}}$$

  which guarantees that $M' \in \mathsf{tag}_\Gamma(N')$. Rule T-Comr$_1$ for replicated input of channels is proved using the reason as for the present case.

- Case $s[\mathsf{goto}\ r.P]^t_\Delta \mapsto r[P]^s_\Delta$. By hypothesis, $M \in \mathsf{tag}_\Gamma(N)$, then N is of the form $s[\mathsf{goto}\ r.P]$, which reduces to $r[P]$. So, we need to show that $r[P]^s_\Delta \in \mathsf{tag}_\Gamma(r[P]) = \{r[P]^t_\Sigma \text{ s.t. } \Gamma <\colon \Sigma,\ \Sigma \vdash_r P\colon A,\ \Sigma \vdash r\ \mathsf{allows}\ s\colon A,\ \Sigma \vdash s\ \mathsf{canEnter}\ r\}$. By lemma 20, $\Gamma \Vdash s[\mathsf{goto}\ r.P]^t_\Delta$, and by subject reduction, theorem 4, we have that $\Gamma \Vdash r[P]^s_\Gamma$. Hence, the following derivation

$$\frac{\begin{array}{cc}\Delta \vdash_r P\colon A & \Gamma <\colon \Delta\\ \Delta \vdash r\ \mathsf{allows}\ s\colon A & \Delta \vdash s\ \mathsf{canEnter}\ r\end{array}}{\Gamma \Vdash r[P]^s_\Delta}$$

  concludes the case.

- Case $\dfrac{M_1 \mapsto M_2}{(\boldsymbol{\nu}_t\, n\colon T)\ M_1 \mapsto (\boldsymbol{\nu}_t\, n\colon T)\ M_2}$.

  By hypothesis, $(\boldsymbol{\nu}_t\, n\colon T)\ M_1 \in \mathsf{tag}_\Gamma(N)$. Then, N is of the form $(\boldsymbol{\nu} n\colon T)\ N_1$, and by definition of $\mathsf{tag}_\Gamma(N)$, we conclude that $M_1 \in \mathsf{tag}_\Gamma(N_1)$. By induction hypothesis, considering that $M_1 \in \mathsf{tag}_\Gamma(N_1)$, and that $M_1 \mapsto M_2$, we infer that there exists N_2 such that $N_1 \to N_2$, and $M_2 \in \mathsf{tag}_\Gamma(N_2)$. With the conditions assured by the fact that $(\boldsymbol{\nu}_t\, n\colon T)\ M_1 \in \mathsf{tag}_\Gamma(N)$, and $M_2 \in \mathsf{tag}_\Gamma(N_2)$, we conclude that $(\boldsymbol{\nu}_t\, n\colon T)\ M_2 \in \mathsf{tag}_\Gamma((\boldsymbol{\nu} n\colon T)\ N_2)$.

- Case $\dfrac{M_1 \mapsto M_1'}{M_1 \mid M_2 \Vdash M_1' \mid M_2}$.

  By hypothesis, $M_1 \mid M_2 \in \mathsf{tag}_\Gamma(N)$, then N is of the form $N_1 \mid N_2$. By definition of $\mathsf{tag}_\Gamma(N_1 \mid N_2)$, we conclude that $M_1 \in \mathsf{tag}_\Gamma(N_1)$, and $M_2 \in \mathsf{tag}_\Gamma(N_2)$. From $M_1 \in \mathsf{tag}_\Gamma(N_1)$, and from $M_1 \mapsto M_1'$, we can apply the induction hypothesis, and infer that there exists N_1 s.t. $N_1 \rightarrow N_1'$, and $M_1' \in \mathsf{tag}_\Gamma(N_1')$. Therefore, $N_1 \rightarrow N_1'$, leads to $N_1 \mid N_2 \rightarrow N_1' \mid N_2$. Hence, $M_1' \mid M_2 \in \mathsf{tag}_\Gamma(N_1' \mid N_2)$, because $M_1' \in \mathsf{tag}_\Gamma(N_1')$, $M_2 \in \mathsf{tag}_\Gamma(N_2)$.

- Case $\dfrac{M_1 \equiv_T M_1' \quad M_1' \mapsto M_2' \quad M_2' \equiv_T M_2}{M_1 \mapsto M_2}$

  By hypothesis, $M_1 \in \mathsf{tag}_\Gamma(N)$, and $M_1 \equiv_T M_1'$. Then, by lemma 21, there exists N_1', such that, $N \equiv N_1'$, and $M_1' \in \mathsf{tag}_\Gamma(N_1')$. Using the induction hypothesis, from $M_1' \in \mathsf{tag}_\Gamma(N_1')$, and $M_1' \Vdash M_2'$, we deduce that there exists N_2' such that $N_1' \rightarrow N_2'$, and $M_2' \in \mathsf{tag}_\Gamma(N'2)$. Finally, from $M_2' \in \mathsf{tag}_\Gamma(N_2')$, and $M_2' \equiv_T M_2$, by lemma 21, there exists N_2 such that $N_2' \equiv N_2$, and $M_2 \in \mathsf{tag}_\Gamma(N_2)$, as we want to show.

This concludes the proof of the theorem. □

A.2.2 Proofs from subsection 4.2

This section presents the proofs for the preservations of types by the tagging function (theorem 3), and for the preservations of types during reduction (theorem 4). The subject reduction proof is based on a similar result for the tagged congruence relation (lemma 22), and on several auxiliary results introduced in the previous section.

Theorem 3 (Tagging preserves types). *If* $\Gamma \vdash N$, *and* $M \in \mathsf{tag}_\Gamma(N)$, *then* $\Gamma \Vdash M$.

Proof. By induction on the structure of N.

- Case $N \equiv \mathsf{stop}$. By definition $\mathsf{tag}_\Gamma(\mathsf{stop}) = \{\mathsf{stop}\}$, and $\Gamma \Vdash \mathsf{stop}$, since the stop network is typed by any well-formed environment.

- Case $N \equiv N_1 \mid N_2$. By hypothesis,

$$\frac{\Gamma \vdash N_1 \quad \Gamma \vdash N_2}{\Gamma \vdash N_1 \mid N_2,},$$

and by definition $\mathsf{tag}_\Gamma(N_1 \mid N_2) = \{M_1 \mid M_2 \text{ s.t. } M_i \in \mathsf{tag}_\Gamma(N_i)\}$. Therefore, $\Gamma \vdash N_i$, and $M_i \in \mathsf{tag}_\Gamma(N_i)$, so, by induction hypothesis, $\Gamma \Vdash M_i$. Hence, the following derivation holds.

$$\frac{\Gamma \Vdash M_1 \qquad \Gamma \Vdash M_2}{\Gamma \Vdash M}$$

- Case $N \equiv s[P]$. By hypothesis, $M \in \mathsf{tag}_\Gamma(s[P])$. So $M \equiv_T s[P]^t_\Delta$, with $\Delta \vdash_s P\colon A$, $\Gamma <\colon \Delta$, $\Delta \vdash s$ allows $t\colon A$, and $\Delta \vdash t$ canEnter s. Therefore, we conclude that $\Gamma \Vdash s[P]^t_\Delta$, using rule T-Site.

$$\frac{\Delta \vdash_s P\colon A \qquad \Gamma <\colon \Delta \qquad \Delta \vdash s \text{ allows } t\colon A \qquad \Delta \vdash t \text{ canEnter } s}{\Gamma \Vdash s[P]^t_\Delta}$$

- Case $N \equiv (\boldsymbol{\nu} a\colon C@s)\ N_1$. By hypothesis, $M \in \mathsf{tag}_\Gamma((\boldsymbol{\nu} a\colon C@s)\ N_1)$. Thus, $M \equiv_T (\boldsymbol{\nu}_t\, a\colon C@s)\ M_1$, with $M_1 \in \mathsf{tag}_{\Gamma,\mathsf{a}\colon \mathsf{C@s}}(N_1)$, and $\Gamma \vdash s$ allows $t\colon \{\mathsf{go}\}$. Consequently, by induction hypothesis,

$$\frac{\Gamma, a\colon C@s \Vdash M_1 \qquad \Gamma \vdash s \text{ allows } t\colon \{\mathsf{go}\}}{\Gamma \Vdash (\boldsymbol{\nu}_t\, a\colon C@s)\ M_1}$$

- Case $N \equiv (\boldsymbol{\nu} s\colon G)\ N_1$. By hypothesis, $M \in \mathsf{tag}_\Gamma((\boldsymbol{\nu} s\colon G)\ N_1) = \{(\boldsymbol{\nu}_t\, s\colon G)\ M_1 \text{ s.t. } M_1 \in \mathsf{tag}_{\Gamma,\mathsf{s}\colon \mathsf{G@s}}(N_1)\}$. Since, $\Gamma \vdash (\boldsymbol{\nu} s\colon G)\ N_1$, then $\Gamma, s\colon G@s \vdash N_1$, and, by induction hypothesis, $\Gamma, s\colon G@s \Vdash M_1$. The judgement $\Gamma \Vdash (\boldsymbol{\nu} s\colon G)\ M$ follows by rule T-Ress.

- Case $N \equiv (\boldsymbol{\nu} g\colon (R, G))\ N_1$. The proof for this case is handled as the case above.

□

Lemma 22 (Congruence lemma – Tagged language). *If* $\Gamma \Vdash N, N \equiv_T M$*, then* $\Gamma \Vdash M$*.*

Proof. By induction in the typing of $\Gamma \Vdash N$. We analyse only the two congruence rules that differ significantly from the congruence relation for the untagged language.

- Case $(\boldsymbol{\nu}_t\, n\colon L@s)\ s[P]^t_{\Delta \sqcap n\colon T@s} \equiv s[(\boldsymbol{\nu} n\colon L)\ P]^t_\Delta$, for $n \notin \mathrm{dom}(\Delta) \cup \{s\}$ We analyse only the case of channel creation, $(\boldsymbol{\nu}_t\, a\colon C@s)\ s[P]^t_{\Delta \sqcap a\colon C@s} \equiv s[(\boldsymbol{\nu} a\colon L)\ P]^t_\Delta$, since the proof for site and group creation is similar to the proof for the untagged version.

By hypothesis,

$$\dfrac{\dfrac{\begin{array}{c}\Delta \vdash t\ \mathsf{canEnter}\ s \\ \Delta \sqcap a\colon C@s \vdash_s P\colon A \\ \Gamma, a\colon C@s <\colon \Delta \sqcap a\colon C@s \\ \Delta \sqcap a\colon C@s \vdash s\ \mathsf{allows}\ t\colon A\end{array}}{\Gamma, a\colon C@s \Vdash s[P]^t_{\Delta \sqcap a\colon C@s}} \qquad \Gamma \vdash s\ \mathsf{allows}\ t\colon \{\mathsf{createRes}\}}{\Gamma \Vdash (\boldsymbol{\nu}_t\, a\colon C@s)\ s[P]^t_{\Delta \sqcap a\colon C@s}}$$

Since a $\notin \mathrm{dom}(\Delta) \cup \{s\}$, then $\Delta \sqcap a\colon C@s = \Delta, a\colon C@s$. Hence,

$$\dfrac{\Delta, a\colon C@s \vdash_s P\colon A}{\Delta \vdash_s (\boldsymbol{\nu} a\colon C)\ P\colon A \cup \{\mathsf{useRes}\}}.$$

If $\Delta \sqcap a\colon C@s \vdash s\ \mathsf{allows}\ t\colon A$, then $\Delta \vdash s\ \mathsf{allows}\ t\colon A$, since $a\colon C@s$ does not change the security policies defined in Δ. From $\Delta \vdash s\ \mathsf{allows}\ t\colon A$, $\Delta \vdash s\ \mathsf{allows}\ t\colon \{\mathsf{useRes}\}$, and $\Gamma <\colon \Delta$, we conclude that $\Delta \vdash s\ \mathsf{allows}\ t\colon A \cup \{\mathsf{useRes}\}$, because the security for the groups referred in Δ and Γ are the same (there is no subtyping on group types). Therefore, we can conclude

$$\dfrac{\begin{array}{cc}\Delta \vdash_s (\boldsymbol{\nu} a\colon C)\ P\colon A \cup \{\mathsf{useRes}\} & \Gamma <\colon \Delta \\ \Delta \vdash s\ \mathsf{allows}\ t\colon A \cup \{\mathsf{useRes}\} & \Delta \vdash t\ \mathsf{canEnter}\ s\end{array}}{\Gamma \Vdash s[(\boldsymbol{\nu} a\colon C)\ P]^t_\Delta}$$

- Case $s[P]^t_\Delta \mid s[Q]^t_\Delta \equiv s[P \mid Q]^t_\Delta$
  Therefore we can deduce

$$\dfrac{\Gamma \Vdash s[P]^t_\Delta \qquad \Gamma \Vdash s[Q]^t_\Delta}{\Gamma \Vdash s[P]^t_\Delta \mid s[Q]^t_\Delta}$$

from

$$\dfrac{\begin{array}{cc}\Delta \vdash_s P\colon A & \Gamma <\colon \Delta \\ \Delta \vdash s\ \mathsf{allows}\ t\colon A & \Delta \vdash t\ \mathsf{canEnter}\ s\end{array}}{\Gamma \Vdash s[P]^t_\Delta},$$

and from

$$\dfrac{\begin{array}{cc}\Delta \vdash_s Q\colon B & \Gamma <\colon \Delta \\ \Delta \vdash s\ \mathsf{allows}\ t\colon B & \Delta \vdash t\ \mathsf{canEnter}\ s\end{array}}{\Gamma \Vdash s[Q]^t_\Delta}.$$

Hence,

$$\frac{\Delta \vdash_s P : A \qquad \Delta \vdash_s Q : B}{\Delta \vdash_s P \mid Q : A \cup B},$$

and, we conclude with,

$$\frac{\begin{array}{cc}\Delta \vdash_s P \mid Q : A \cup B & \Gamma <: \Delta \\ \Delta \vdash s \text{ allows } t : A \cup B & \Delta \vdash t \text{ canEnter } s\end{array}}{\Gamma \Vdash s[P \mid Q]^t_\Delta}$$

which concludes the proof. □

Theorem 4 (Subject reduction – Tagged language). *If* $\Gamma \Vdash N$*, and* $N \mapsto M$*, then* $\Gamma \Vdash M$*.*

Proof. By induction on the typing of $\Gamma \Vdash N$. We analyse the definition of the tagged reduction relation, and establish induction on the last rule applied.

1. Case $s[\mathsf{goto}\ r.P]^t_\Delta \mapsto r[P]^s_\Delta$.
   By hypothesis, the following derivation holds.

$$\frac{\begin{array}{cc}\Delta \vdash_s \mathsf{goto}\ r.P : \{\mathsf{go}\} & \Delta \vdash s \text{ allows } t : \{\mathsf{go}\} \\ \Gamma <: \Delta & \Delta \vdash t \text{ canEnter } s\end{array}}{\Gamma \Vdash s[\mathsf{goto}\ r.P]^t_\Delta}$$

   provided that

$$\frac{\Delta \vdash_r P : A \qquad \Delta \vdash r \text{ allows } s : A \qquad \Delta \vdash s \text{ canEnter } r}{\Delta \vdash_s \mathsf{goto}\ r.P : \{\mathsf{go}\}}$$

   Therefore, we can conclude that

$$\frac{\begin{array}{cc}\Delta \vdash_r P : A & \Gamma <: \Delta \\ \Delta \vdash r \text{ allows } s : A & \Delta \vdash s \text{ canEnter } r\end{array}}{\Gamma \Vdash r[P]^s_\Delta}$$

2. Case $s[a!\langle\diamond\rangle]^t_\Sigma \mid s[a?(\diamond)\ P]^u_\Delta \mapsto s[P]^u_\Delta$
   Since the left-hand side of the tagged reduction relation is typed by hypothesis, we get

$$\frac{\Gamma \Vdash s[a!\langle\diamond\rangle]^t_\Sigma \qquad \Gamma \Vdash s[a?(\diamond)\ P]^u_\Delta}{\Gamma \Vdash s[a!\langle\diamond\rangle]^t_\Sigma \mid s[a?(\diamond)\ P]^u_\Delta}$$

from

$$\dfrac{\dfrac{\Gamma \vdash \mathrm{env} \quad \Gamma(a) \vdash \langle \mathsf{unit} \rangle^{\mathsf{w}}{}_{@} s}{\Sigma \vdash_s a!\langle \diamond \rangle \colon \{\mathsf{useRes}\}} \quad \Gamma <: \Sigma \quad \Sigma \vdash s \ \mathsf{allows}\ t \colon \{\mathsf{useRes}\} \quad \Sigma \vdash t \ \mathsf{canEnter}\ s}{\Gamma \Vdash s[a!\langle \diamond \rangle]^t_\Sigma}$$

and

$$\dfrac{\dfrac{\Delta \vdash_s P \colon A \quad \Delta(a) <: \langle \mathsf{unit} \rangle^{\mathsf{r}}{}_{@} s}{\Delta \vdash_s a?(\diamond)\ P \colon A \cup \{\mathsf{installRes}\}} \quad \Gamma <: \Delta \quad \Delta \vdash s \ \mathsf{allows}\ u \colon A \cup \{\mathsf{installRes}\} \quad \Delta \vdash u \ \mathsf{canEnter}\ s}{\Gamma \Vdash s[a?(\diamond)\ P]^u_\Delta}$$

Hence, using lemma 7, we conclude that $\Delta \vdash s \ \mathsf{allows}\ u \colon A$, thus

$$\dfrac{\Delta \vdash_s P \colon A \quad \Gamma <: \Delta \quad \Delta \vdash s \ \mathsf{allows}\ u \colon A \quad \Delta \vdash u \ \mathsf{canEnter}\ s}{\Gamma \Vdash s[P]^u_\Delta}$$

3. Case $s[a!\langle b_{@} r \rangle]^t_\Sigma \mid s[a?(x_{@} y)\ P]^u_\Delta \mapsto s[P\{r/y\}\{b/x\}]^u_{\Delta \sqcap r \colon \Sigma(r) \sqcap b \colon \Sigma(b)}$
By hypothesis, the following derivation holds.

$$\dfrac{\Gamma \Vdash s[a!\langle b_{@} r \rangle]^t_\Sigma \quad \Gamma \Vdash s[a?(x_{@} y)\ P]^u_\Delta}{\Gamma \Vdash s[a!\langle b_{@} r \rangle]^t_\Sigma \mid s[a?(x_{@} y)\ P]^u_\Delta}$$

from

$$\dfrac{\dfrac{\Sigma \vdash \mathrm{env} \quad \Gamma(a) <: \langle C_{@} H \rangle^{\mathsf{w}}{}_{@} s \quad \Sigma(r) = H \quad \Sigma(b) = C_{@} r}{\Sigma \vdash_s a!\langle b_{@} r \rangle \colon \{\mathsf{useRes}\}} \quad \Gamma <: \Sigma \quad \Sigma \vdash s \ \mathsf{allows}\ t \colon \{\mathsf{useRes}\} \quad \Sigma \vdash t \ \mathsf{canEnter}\ s}{\Gamma \Vdash s[a!\langle b_{@} r \rangle]^t_\Sigma}$$

and

$$\dfrac{\dfrac{\Delta, x \colon C''_{@} y, y \colon H'' \vdash_s P \colon A \quad \Delta(a) <: \langle C''_{@} H'' \rangle^{\mathsf{r}}{}_{@} s}{\Delta \vdash_s a?(x_{@} y)\ P \colon A \cup \{\mathsf{installRes}\}} \quad \Gamma <: \Delta \quad \Delta \vdash s \ \mathsf{allows}\ u \colon A \cup \{\mathsf{installRes}\} \quad \Delta \vdash u \ \mathsf{canEnter}\ s}{\Gamma \Vdash s[a?(x_{@} y)\ P]^u_\Delta}$$

We want to prove that

$$\Gamma \Vdash s[P\{r/y\}\{b/x\}]^{u}_{\Delta \sqcap r\colon \Sigma(r) \sqcap b\colon \Sigma(b)}$$

which means that

$$\frac{\begin{array}{c}\Gamma <\colon \Delta \sqcap r\colon \Sigma(r) \sqcap b\colon \Sigma(b) \\ \Delta \sqcap r\colon \Sigma(r) \sqcap b\colon \Sigma(b) \vdash s \ \mathsf{allows}\ u\colon A \\ \Delta \sqcap r\colon \Sigma(r) \sqcap b\colon \Sigma(b) \vdash u \ \mathsf{canEnter}\ s \\ \Delta \sqcap r\colon \Sigma(r) \sqcap b\colon \Sigma(b) \vdash_s P\{r/y\}\{b/x\}\colon A\end{array}}{\Gamma \Vdash s[P\{r/y\}\{b/x\}]^{u}_{\Delta \sqcap r\colon \Sigma(r) \sqcap b\colon \Sigma(b)}}$$

First, we substitute r for y. By hypothesis, $\Delta, x\colon C''{}_{@}y, y\colon H'' \vdash_s P\colon A$. We analyse two subcases:

- Case $r \in \mathrm{dom}(\Delta)$
  Thus, $\Gamma(r) <\colon \Sigma(r)$, and $\Gamma(r) <\colon \Delta(r)$; then there exists $\Sigma(r) \sqcap \Delta(r)$ such that $\Gamma(r) <\colon \Sigma(r) \sqcap \Delta(r) <\colon \Delta(r)$. Let $\Delta'(z) = \Delta(z)$, for $z \neq r$, and $\Delta'(r) = \Delta(r) \sqcap \Sigma(r)$; using lemma 17, $\Delta', x\colon C''{}_{@}y, y\colon H'' \vdash_s P\colon A$. From $\Gamma(a) <\colon \Sigma(a) <\colon \langle C{}_{@}H\rangle^{\mathsf{w}}{}_{@}s$, $\Gamma(a) <\colon \Delta'(a) <\colon \langle C''{}_{@}H''\rangle^{\mathsf{r}}{}_{@}s$, and $\Delta'(r) <\colon \Sigma(r)$, we conclude that $\Delta'(r) \subseteq H \subseteq H''$.
- Case $r \notin \mathrm{dom}(\Delta)$
  So, $(\Delta, x\colon C''{}_{@}y, y\colon H'') \sqcap r\colon \Sigma(r)$ is well defined (by definition of $\sqcap$ operator). Let $\Delta' = \Delta, r\colon \Sigma(r)$; hence, by lemma 14, $\Delta', x\colon C''{}_{@}y, y\colon H'' \vdash_s P\colon A$. The inclusion $\Delta'(r) \subseteq H''$ holds, because $\Delta'(r) = \Sigma(r) = H$, and $H \subseteq H''$, according to the facts that $\Gamma(a) <\colon \Sigma(a) <\colon \langle C{}_{@}H\rangle^{\mathsf{w}}{}_{@}s$, and $\Gamma(a) <\colon \Delta'(a) <\colon \langle C''{}_{@}H''\rangle^{\mathsf{r}}{}_{@}s$.

Therefore, for both subcases, $\Delta'(r) \subseteq H''$, $\Delta', x\colon C''{}_{@}y, y\colon H'' \vdash_s P\colon A$, $y \notin \mathrm{dom}(\Delta')$, and $y \neq s$; hence, by lemma 10, one can prove that $\Delta', x\colon C''{}_{@}r \vdash_s P\{r/y\}\colon A$.

In what follows, we consider the substitution of b for x. We will proceed using the approach taken above for r, and consider also two subcases.

- Case $b \in \mathrm{dom}(\Delta')$
  In the case $\Gamma(b) <\colon \Sigma(b)$ and $\Gamma(b) <\colon \Delta'(b)$, so $\Sigma(b) \sqcap \Delta'(b)$ is defined. Let $\Delta''(z) = \Delta'(z)$, for $z \neq b$, and $\Delta''(b) = \Sigma(b) \sqcap \Delta'(b)$. Therefore, $\Delta', x\colon C''{}_{@}r \vdash_s P\{r/y\}\colon A$, and $(\Delta', x\colon C''{}_{@}r) \sqcap b\colon \Sigma(b)$ is well defined. Hence, by lemma 18, $\Delta'', C''{}_{@}r \vdash_s P\{r/y\}\colon A$. From $\Gamma(a) <\colon \Sigma(a) <\colon \langle C{}_{@}H\rangle^{\mathsf{w}}{}_{@}s$, and from $\Gamma(a) <\colon \Delta''(a) <\colon$

$\langle C''_{@}H''\rangle^{\mathsf{r}}{}_{@}s$, we conclude that $C_{@}r <: C''_{@}r$. Since $\Sigma(b) = C_{@}r$, and $\Delta''(b) <: \Sigma(b)$ (by definition of $\sqcap$ operator) we get that $\Delta''(b) <: C''_{@}r$.

- Case $b \neq \mathrm{dom}(\Delta')$
  Consequently, $(\Delta', x\colon C''_{@}y) \sqcap b\colon \Sigma(b)$ is well defined (by definition of $\sqcap$ operator). Lemma 18 assures that $(\Delta', x\colon C''_{@}t) \sqcap b\colon \Sigma(b) \vdash_s P\{r/y\}\colon A$. Let $\Delta'' = \Delta', b\colon \Sigma(b)$; then $\Delta''(b) <: C''_{@}r$ because $C_{@}r <: C''_{@}r$ (just consider that $\Gamma(a) <: \Sigma(a) <: \langle C_{@}H\rangle^{\mathsf{w}}{}_{@}s$ and $\Gamma(a) <: \Delta''(a) <: \langle C''_{@}H''\rangle^{\mathsf{r}}{}_{@}s$), and $\Sigma(b) = C_{@}r = \Delta''(b)$.

Therefore, since $\Delta''(b) <: C''_{@}r$, and $\Delta'', r\colon C''_{@}r \vdash_s P\{r/y\}\colon A$ hold for both cases, by lemma 12, we conclude that $\Delta'' \vdash_s P\{r/y\}\{b/x\}\colon A$, which concludes the proof for this case.

We have shown that $\Delta \sqcap r\colon \Sigma(r) \sqcap b\colon \Sigma(b) \vdash_s P\{r/y\}\{b/x\}\colon A$.

Next, we prove that $\Gamma <: \Delta''$. Indeed, Δ'' only differ from Δ with respect to r and b. To prove that $\Gamma <: \Delta''$ it suffices to show that $\Gamma(r) <: \Delta''(r)$, and $\Gamma(b) <: \Delta''(b)$, which is true from the definition of $\sqcap$. Thus, $\Gamma <: \Delta \sqcap r\colon \Sigma(r) \sqcap b\colon \Sigma(b)$.

To prove that $\Delta \sqcap r\colon \Sigma(r) \sqcap b\colon \Sigma(b) \vdash s$ allows $u\colon A$ holds, consider the following subcases.

- $r \neq s$ and $r \neq u$
  Since the type of r does not interferes with the types of s, and of u, by lemma 7, $\Delta \sqcap r\colon \Sigma(r) \sqcap b\colon \Sigma(b) \vdash s$ allows $u\colon A$ holds.
- $r = s$ and $s = u$
  The judgement $\Delta \sqcap r\colon \Sigma(r) \sqcap b\colon \Sigma(b) \vdash s$ allows $u\colon A$ holds because $\Gamma \vdash r$ allows $r\colon A$ is an axiom of the allows relation.
- $r = s$ and $r \neq u$
  By hypothesis, $\Delta \vdash r$ allows $u\colon A \cup \{\mathsf{installRes}\}$. Since $\Delta(r) \sqcap \Sigma(r)$ is well defined, $\Delta(r) \sqcap \Sigma(r) \subseteq \Delta(r)$, and by lemma 7, $\Delta \sqcap r\colon \Sigma(r) \sqcap b\colon \Sigma(b) \vdash r$ allows $u\colon A$.
- $r \neq s$ and $r = u$
  Then, $\Delta \sqcap r\colon \Sigma(r) \sqcap b\colon \Sigma(b) \vdash s$ allows $r\colon A \cup \{\mathsf{installRes}\}$ using arguments similar to the ones stated for the previous case.

At last, we need to prove also that $\Delta \sqcap r\colon \Sigma(r) \sqcap b\colon \Sigma(b) \vdash u$ canEnter s holds. The proof follows a similar pattern to the one for allows, replacing canEnter for allows and using lemma 8 instead of lemma 7.

Let $\Delta'' = \Delta \sqcap r\colon \Sigma(r) \sqcap b\colon \Sigma(b)$. Now, we are in position to state that:

$$\frac{\Delta'' \vdash_s P\{r/y\}\{b/x\}\colon A \qquad \Gamma <\colon \Delta'' \qquad \Delta'' \vdash s \;\mathsf{allows}\; u\colon A \qquad \Delta'' \vdash u \;\mathsf{canEnter}\; s}{\Gamma \Vdash s[P\{r/y\}\{b/x\}]^u_{\Delta \sqcap r\colon \Sigma(r) \sqcap b\colon \Sigma(b)}}$$

that concludes the case.

4. Case $s[a!\langle\diamond\rangle]^t_\Sigma \mid s[a?\ast(\diamond)\ P]^u_\Delta \to s[P]^u_\Delta \mid s[a?\ast(\diamond)\ P]^u_\Delta$
   By hypothesis, $\Gamma \vdash s[a!\langle\diamond\rangle]^t_\Sigma \mid s[a?\ast(\diamond)\ P]^u_\Delta$, therefore, from the type derivation, we conclude that, (1) $\Delta \vdash_s P\colon A$, (2) $\Delta \vdash s \;\mathsf{allows}\; u\colon A \cup \{\mathsf{installRes}\}$, (3) $\Delta \vdash u \;\mathsf{canEnter}\; s$, (4) $\Gamma <\colon \Delta$, and (5) $\Gamma \vdash s[a?\ast(\diamond)\ P]^u_\Delta$. Using lemma 7 on (2), we can proof (2') $\Delta \vdash s \;\mathsf{allows}\; u\colon A$. From (1), (2'), (3), and (4), we derive (6) $\Gamma \vdash s[P]^u_\Delta$. Using (6), and (5), we conclude the case.

5. Case $s[a!\langle b @ r\rangle]^t_{\Sigma, r\colon G, b\colon C@r} \mid s[a?\ast(x @ y)\ P]^u_\Delta \to$
   $s[P\{r/y\}\{b/x\}]^u_{\Delta \sqcap r\colon G \sqcap b\colon C@r} \mid s[a?\ast(x @ y)\ P]^u_\Delta$
   From the typing derivation of $\Gamma \Vdash s[a!\langle b @ r\rangle]^t_{\Sigma, r\colon G, b\colon C@r} \mid s[a?\ast(x @ y)\ P]^u_\Delta$, we conclude that (a) $\Gamma \Vdash s[a!\langle b @ r\rangle]^t_{\Sigma, r\colon G, b\colon C@r}$, (b) $\Gamma \Vdash s[a?\ast(x @ y)\ P]^u_\Delta$, and (c) $\Delta \vdash_s a?(x @ y)\ P\colon A \cup \{\mathsf{installRes}\}$. From (a), and (c), using similar arguments as for case 3 above, we deduce that (d) $\Gamma \Vdash s[P\{r/y\}\{b/x\}]^u_{\Delta \sqcap r\colon G \sqcap b\colon C@r}$. Using (b), and (d), we end the case.

6. Case $\dfrac{N \mapsto M}{(\boldsymbol{\nu} n\colon L @ s)\ N \mapsto (\boldsymbol{\nu} n\colon L @ s)\ M}$ and $\dfrac{N \mapsto N'}{N \mid M \mapsto N' \mid M}$

   The proof is equal to case 6 of the proof of theorem 1.

7. Case $\dfrac{N \equiv N' \qquad N' \mapsto M' \qquad M' \equiv M}{N \mapsto M}$

   The proof follows a similar pattern to case 7 of the proof of theorem 1, but uses lemma 22 instead of lemma 16.

This concludes the proof. □

A.2.3 Proofs from subsection 4.3

In this subsection we prove our type safety result.

Theorem 5. *If* $\Gamma \Vdash N$, *then* $M \not\xrightarrow{err}$.

Proof. We prove the contrapositive result, namely, that $\mathsf{tag}_\Gamma(N') \overset{err}{\longmapsto}$ implies that there is no Γ s.t. $\Gamma \Vdash N$. We proceed by induction on the definition of $\overset{err}{\longmapsto}$ relation.

- Case $s[a!\langle\diamond\rangle]^r_\Gamma \overset{err}{\longmapsto}$, if $\Gamma \not\Vdash s$ allows $r\colon \{\mathsf{useRes}\}$, or $\Gamma(a) \not<: \langle \mathsf{unit}\rangle^{\mathsf{w}}{}_{@}s$. We assume that $\Delta \Vdash s[a!\langle\diamond\rangle]^r_\Gamma$, and show that from this premise we may conclude that $\Gamma \vdash s$ allows $r\colon$ useRes, and that $\Gamma(a) <: \langle \mathsf{unit}\rangle^{\mathsf{w}}{}_{@}s$. In fact, to conclude $\Delta \Vdash s[a!\langle\diamond\rangle]^r_\Gamma$, we may only apply T-Site,

$$\frac{\begin{array}{cc}\Gamma \vdash_s a!\langle\diamond\rangle\colon \{\mathsf{useRes}\} & \Delta <: \Gamma \\ \Gamma \vdash s \text{ allows } r\colon \{\mathsf{useRes}\} & \Gamma \vdash r \text{ canEnter } s\end{array}}{\Delta \Vdash s[a!\langle\diamond\rangle]^r_\Gamma}$$

  concluding that $\Gamma \vdash s$ allows $r\colon \{\mathsf{useRes}\}$. From $\Gamma \vdash_s a!\langle\diamond\rangle\colon \{\mathsf{useRes}\}$, we may derive

$$\frac{\Gamma \vdash \mathrm{env} \qquad \Gamma(a) <: \langle \mathsf{unit}\rangle^{\mathsf{w}}{}_{@}s}{\Gamma \vdash_s a!\langle\diamond\rangle\colon \{\mathsf{useRes}\}}$$

  where $\Gamma(a) <: \langle \mathsf{unit}\rangle^{\mathsf{w}}{}_{@}s$, leading to a contradiction. The proofs for the output of channels, as well as for R-Inp rule, follow a similar scheme to the one shown above.

- Case $s[\mathsf{goto}\ t.P]^r_\Gamma \overset{err}{\longmapsto}$, if $\Gamma \not\Vdash s$ allows $r\colon$ go, or $\Gamma \not\Vdash s$ canEnter t. By way of contradiction, assume that $\Delta \Vdash s[\mathsf{goto}\ t.P]^r_\Gamma$. The following derivation holds.

$$\frac{\begin{array}{cc}\Gamma \vdash_s \mathsf{goto}\ t.P\colon \{\mathsf{go}\} & \Delta <: \Gamma \\ \Gamma \vdash s \text{ allows } t\colon \{\mathsf{go}\} & \Gamma \vdash t \text{ canEnter } s\end{array}}{\Delta \Vdash s[\mathsf{goto}\ t.P]^r_\Gamma}$$

  where we conclude that $\Gamma \vdash s$ allows $t\colon \{\mathsf{go}\}$, and that $\Gamma \vdash t$ canEnter s, leading to a contradiction.

- Case $(\boldsymbol{\nu}_r\, a\colon T)\ s[P]^r_\Gamma \overset{err}{\longmapsto}$, if $\Gamma \not\Vdash s$ allows $r\colon$ createRes. Assuming by way of contradiction that $\Delta \Vdash (\boldsymbol{\nu}_r\, a\colon T)\ s[P]^r_\Gamma$, we show that $\Gamma \vdash s$ allows $r\colon$ createRes is derivable. In fact, it is easy to conclude from rule T-Resc

$$\frac{\Gamma, a\colon C_{@}s \Vdash N \qquad \Gamma \vdash s \text{ allows } t\colon \mathsf{createRes}}{\Gamma \Vdash (\boldsymbol{\nu}_t\, a\colon C_{@}s)\ N}$$

  that we reach a contradiction. The proof for rule R-Res$_1$ is constructed along the same lines.

- Case $N \overset{err}{\longmapsto}$ implies $N \,|\, M \overset{err}{\longmapsto}$. Assuming by way of contradiction that $\Gamma \Vdash N \,|\, M$, then $\Gamma \Vdash N$, and $\Gamma \Vdash M$, and, by induction hypothesis, $N \overset{err}{\not\longmapsto}$, which leads to a contradiction. The proof for R-Res is performed using similar arguments.

- Case $N \equiv_T M$, and $N \overset{err}{\longmapsto}$, implies $M \overset{err}{\longmapsto}$. Assuming by way of contradiction that $\Gamma \Vdash M$, then, by the congruence lemma, (lemma 22), we conclude that $\Gamma \Vdash N$, which, by induction hypothesis, implies that $N \overset{err}{\not\longmapsto}$, leading to a contradiction.

□

Corollary 6 (Type safety). *If* $\Gamma \vdash N$*, and* $N \to^\star N'$*, then* $\mathsf{tag}_\Gamma(N') \overset{err}{\not\longmapsto}$.

Proof. By hypothesis $\Gamma \vdash N$, then, since types are preserved during reduction, theorem 1, $\Gamma \vdash N'$. Let $M \in \mathsf{tag}_\Gamma(N')$, then, since types are preserved by the tagging function, theorem 3, $\Gamma \Vdash M$. By the type safety theorem, theorem 5, we conclude that $M \overset{err}{\not\longmapsto}$. □