A Typed Model for Dynamic Authorizations

Silvia Ghilezan Svetlana Jakšić Jovanka Pantović
University of Novi Sad, Serbia

Jorge A. Pérez
University of Groningen, The Netherlands

Hugo Torres Vieira
IMT Institute for Advanced Studies Lucca, Italy

Security requirements in distributed software systems are inherently dynamic. In the case of authorization policies, resources are meant to be accessed only by authorized parties, but the authorization to access a resource may be dynamically granted/yielded. We describe ongoing work on a model for specifying communication and dynamic authorization handling. We build upon the π-calculus so as to enrich communication-based systems with authorization specification and delegation; here authorizations regard channel usage and delegation refers to the act of yielding an authorization to another party. Our model includes: (i) a novel scoping construct for authorization, which allows to specify authorization boundaries, and (ii) communication primitives for authorizations, which allow to pass around authorizations to act on a given channel. An authorization error may consist in, e.g., performing an action along a name which is not under an appropriate authorization scope. We introduce a typing discipline that ensures that processes never reduce to authorization errors, even when authorizations are dynamically delegated.

1 Introduction

Nowadays, computing systems operate in distributed environments, which may be highly heterogeneous, including at the level of trustworthiness. It is often the case that collaborating systems need to protect themselves from malicious entities by enforcing authorization policies that ensure actions are carried out by properly authorized parties. Such *authorizations* to act upon a resource may be statically prescribed — for instance, a determined party is known to have a determined authorization — but may also be dynamically established — for example, when a server delegates a task to a slave it may be sensible to pass along the appropriate authorization to carry out the delegated task.

As a motivating example, consider the message sequence chart given in Figure 1 describing a scenario where a client interacts with a bank portal in order to request a credit. After the client submits the request, the bank portal asks a teller to approve the request, allowing him/her to join the ongoing interaction. Apart from some rating that could be automatically calculated by the bank portal, it is the teller who ultimately approves/declines the request. It then seems reasonable that the teller *impersonates* the bank when informing the client about the outcome of the request. At this point we may ask: is the teller authorized to act on behalf of the bank portal in this structured interaction? Even if the teller gained access to the communication medium when joining the interaction, the authorization to act on behalf of the bank portal may not be necessarily granted; in such cases an explicit mechanism that dynamically grants such an authorization is required. To account for this kind of scenarios, in previous work [7] we explored the idea of *role authorizations*. It appears to us that the key notions underlying this idea can be well explored in a more general setting; here we aim at distilling such notions in a simple setting.

We distinguish an authorization from the resource itself: a system may already know the identity of the resource (say, an email address or a file name) but may not be authorized to act upon it (e.g., is not able to send an email on behalf of a given address or to write on a file). Also, it might be the case that

S. J. Gay and J. Alglave (Eds.): Programming Language Approaches to Concurrency- and Communication-Centric Software (PLACES 2015).
EPTCS 203, 2016, pp. 73–84, doi:10.4204/EPTCS.203.6

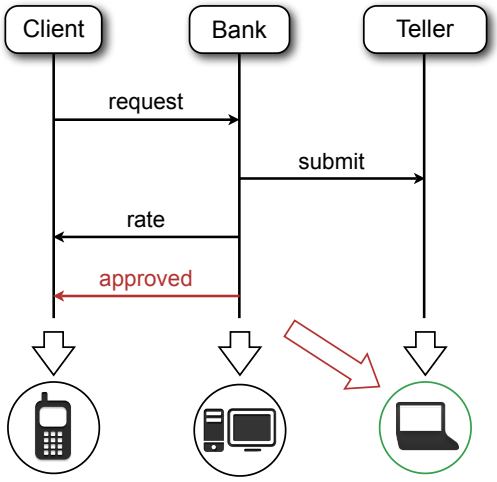

Figure 1: Credit request scenario.

the system acquires knowledge about the resource (for instance, by receiving an email address or a file) but not necessarily is immediately granted access to act upon it. We focus on communication-centered systems in which authorizations are a first-class notion modeled in a dedicated way, minimally extending the π-calculus [11] to capture dynamic authorization handling. As such, the resources that we consider are communication channels; authorizations concern the ability to communicate on channels.

Authorizations may be associated with a *spatial* connotation, as it seems fairly natural that a determined part of the system has access to a resource while the rest of the system does not. To this end, we introduce a *scoping operator* to specify delimited authorizations: we write $(a)P$ to specify that process P is authorized to act upon the resource a. For example, by $(a)a!b.Q$ we specify a process that is authorized on channel a and that is willing to use it to send b after which it behaves as Q.

Also, since we are interested in addressing systems in which authorizations are dynamically passed around, we model authorization communication in a distinguished way by means of dedicated communication actions: we write $a\langle b\rangle.P$ to specify the action of sending an authorization to act upon b (and proceeding as P) and $a(b).P$ to specify the action of receiving an authorization to act upon b (and proceeding as P), where in both cases channel a is used as the underlying communication medium. We remark that both in the authorization scoping $(a)P$ and in the authorization reception $b(a).P$ the name a is not bound so as to capture the notion that authorizations are handled at the level of known identities.

Given the sensitive nature of an authorization, we believe it is natural to enforce a *specialized* discipline regarding authorization manipulation. Namely, we consider that the act of passing along an authorization —*authorization delegation*— entails the yielding of the communicated authorization. That is, a party willing to communicate an authorization loses it after synchronization. Consider, for example, a process

$$S = (a)(a(b).P \mid (b)a\langle b\rangle.Q)$$

while the process on the left-hand side of the parallel composition ($\mid$) is awaiting an authorization to act on b (via a synchronization on channel a), the process on the right-hand side is willing to delegate authorization to act on b. In one reduction step, process S evolves to $(a)(((b)P) \mid Q)$, thus capturing the fact that the process on the right (Q) has now lost the authorization to act on b. Notice that this authorization transfer may have influence on the resources already known to the receiving party (i.e.,

P,Q	::=	0	(Inaction)	α	::=	$a!b$	(Output)
	$\mid$	$P \mid Q$	(Parallel)		$\mid$	$a?x$	(Input)
	$\mid$	$(\nu a)P$	(Restriction)		$\mid$	$a\langle b\rangle$	(Send authorization)
	$\mid$	$(a)P$	(Authorization)		$\mid$	$a(b)$	(Receive authorization)
	$\mid$	$\alpha.P$	(Prefix)				

Table 1: Syntax of processes.

process P may specify communications on channel b).

The fact that authorizations are yielded when communicated allows us to model a form of authorization accounting, in the sense that authorizations are viewed as a "countable" resource. As such, in general we would expect $(a)(a)P$ to differ from $(a)P$. However, since we intuitively interpret $(a)P$ as "the whole of P is authorized to interact on a", it does not seem sensible that part of P can completely yield the authorization. Consider, e.g., process $(a)(b\langle a\rangle.P \mid Q)$ where it does not seem reasonable that the authorization delegation expressed by prefix $b\langle a\rangle$ interferes with the authorization on a already held by Q which is (concurrently) active in the authorization scope. Hence, a system cannot create/discard valid authorizations (that are scoping active processes); authorizations can only float around. It is also reasonable to allow that a party that delegates an authorization may get it back via another synchronization step. This way, our model allows for reasoning about authorization ownership and lending. Finally, we envisage (in our untyped model) a possibility for sharing a given authorization scope with a specified number of parties. This may be represented by specifying multiple copies of the same authorization scope. For example, process $(a)(a)(b)b\langle a\rangle.P$ (or $(a)(b)b\langle a\rangle.(a)P$) will retain authorization scope for a and reduce to $(a)(b)P$, after communication with $(b)b(a).Q$

Some previous works have explored dedicated scoping operators with security motivations (see, e.g., [8, 13]). However, to our knowledge, the particular combination of a (non binding) scoping construct with name passing as in the π-calculus seems to be new. The syntactic elements of our process model, together with the dynamic nature of authorizations, pose challenges at the level of statically identifying processes that act only upon resources for which they are properly authorized. In this paper we start exploring a typing discipline for authorization manipulation that allows to statically ensure that processes never incur in authorization errors, essentially by accounting process authorization requirements. In the remaining, we formally present the language and type system, and state our results.

2 Process Model

We introduce our process calculus with authorization scoping and authorization delegation. Let $\mathcal{N}$ be a countable set of *names*, ranged over by a, b, $c,\ldots,x$, y, z. The syntax of processes is given in Table 1. Processes 0, $P \mid Q$, $(\nu a)P$, $a!b.P$ and $a?x.P$ comprise the usual π-calculus operators for specifying inaction, parallel composition, name restriction, and output and input communication actions, respectively. We introduce three novel operators, motivated earlier:

1. $a\langle b\rangle.P$ sends an authorization for the name b on a and proceeds as P;
2. $a(b).P$ receives an authorization for the name b on a and proceeds as P;
3. $(a)P$ authorizes all actions on the channel a in P.

We remark on the novel reasoning regarding scope authorization $(a)P$ in combination with π-calculus-like name passing, since all actions on channel a in process P are authorized, including actions originally specified for received names. For example, consider a process $P = (b)(a)b?x.x!c.0$ that interacts in a

$$P \mid 0 \equiv P \qquad P \mid Q \equiv Q \mid P \qquad (P \mid Q) \mid R \equiv P \mid (Q \mid R) \qquad (\nu a)0 \equiv 0$$
$$(\nu a)(\nu b)P \equiv (\nu b)(\nu a)P \qquad P \mid (\nu a)Q \equiv (\nu a)(P \mid Q) \quad \text{if } a \notin \mathsf{fn}(P) \qquad P \equiv_\alpha Q \implies P \equiv Q$$
$$(a)(b)P \equiv (b)(a)P \qquad (a)0 \equiv 0 \qquad (a)(P \mid Q) \equiv (a)P \mid (a)Q \qquad (a)(\nu b)P \equiv (\nu b)(a)P \quad \text{if } a \neq b$$

Table 2: Structural congruence.

context that sends name a on b. Then P may evolve to $P' = (b)(a)a!c.0$, which is authorization safe. Still, authorizations may be "revoked" via authorization delegations.

We introduce some auxiliary notions and abbreviations, useful for the remaining formal presentation. The set of free names of a process P, denoted $\mathsf{fn}(P)$, accounts for authorization constructs in the following way:

$$\begin{aligned} \mathsf{fn}((a)P) &\triangleq \{a\} \cup \mathsf{fn}(P) \\ \mathsf{fn}(a\langle b\rangle.P) = \mathsf{fn}(a(b).P) &\triangleq \{a,b\} \cup \mathsf{fn}(P) \end{aligned}$$

Given a name a, we use α_a to refer to either $a!b$, $a?x$, $a\langle b\rangle$, or $a(b)$. We abbreviate $(\nu a_1)(\nu a_2)\ldots(\nu a_k)P$ by $(\nu \vec{a})P$ and likewise $(a_1)(a_2)\ldots(a_k)P$ by $(\vec{a})P$.

Structural congruence expresses basic identities on the structure of processes; it is defined as the least equivalence relation between processes that satisfies the rules given in Table 2. Apart from the usual identities for the static fragment of the π-calculus (cf. first seven rules in Table 2), structural congruence gives basic principle for the novel authorization scope: (i) authorizations can be swapped around; (ii) authorizations can be discarded/created only for the inactive process; (iii) authorizations distributes over parallel composition; and (iv) authorizations and name restrictions can be swapped if the corresponding names differ. We remark that, differently from name restrictions, authorization scopes can neither be extruded/confined: since authorizations are specified for free names (that cannot be α-converted), extruding/confining authorizations actually changes the meaning of processes. For example, processes $(b)b?x.x!b.0$ and $(a)(b)b?x.x!b.0$ are not considered as structurally congruent, as the latter one authorizes the action on a in case it receives a through b, while the former one does not. Another distinctive property comes from the significance of multiplicity of authorization scopes. We do not adopt $(a)P \equiv (a)(a)P$ for $P \neq 0$, for the sake of authorization accounting. Before presenting the operational semantics of the language, we ensure that the rewriting supported by structural congruence is enough to isolate top level communication actions together with their respective authorization scopes.

Proposition 1 (Normal Form) *For any process Q we have that there are $P_1,\ldots,P_k$, $\alpha_1,\ldots,\alpha_k$, $\vec{c}$, and $\vec{a}_1,\ldots,\vec{a}_k$, where $(\nu\vec{c})$ and $(\vec{a}_i)$ for $i \in 1,\ldots,k$ can be empty sequences, such that*

$$Q \equiv (\nu\vec{c})((\vec{a}_1)\alpha_1.P_1 \mid (\vec{a}_2)\alpha_2.P_2 \mid \ldots \mid (\vec{a}_k)\alpha_k.P_k) \tag{1}$$

Proof (by induction on the structure of Q)
$Q \equiv 0$: It is in the form (1).
$Q \equiv Q' \mid Q''$: By induction hypothesis, we have that

$$Q' \equiv (\nu\vec{c})((\vec{a}_1)\alpha_1.P_1 \mid \ldots \mid (\vec{a}_k)\alpha_k.P_k) \qquad Q'' \equiv (\nu\vec{d})((\vec{b}_1)\beta_1.Q_1 \mid \ldots \mid (\vec{b}_l)\beta_l.Q_l)$$

and we can assume, by application of α-conversion, that $\vec{d} \cap \mathsf{fn}(Q') = \emptyset$. Therefore,

$$Q \equiv (\nu\vec{c})(\nu\vec{d})((\vec{a}_1)\alpha_1.P_1 \mid \ldots \mid (\vec{a}_k)\alpha_k.P_k \mid (\vec{b}_1)\beta_1.Q_1 \mid \ldots \mid (\vec{b}_l)\beta_l.Q_l).$$

$$\frac{P \equiv P' \to Q' \equiv Q}{P \to Q}\ (\text{STRU}) \qquad \frac{P \to Q}{P \mid R \to Q \mid R}\ (\text{PARC}) \qquad \frac{P \to Q}{(\nu a)P \to (\nu a)Q}\ (\text{NEWC}) \qquad \frac{P \to Q}{(a)P \to (a)Q}\ (\text{AUTC})$$

$$(\text{COMM})\quad (\vec{a}_1)(b)b!c.P \mid (\vec{a}_2)(b)b?x.Q \to (\vec{a}_1)(b)P \mid (\vec{a}_2)(b)Q\{c/x\}$$

$$(\text{AUTH})\quad (\vec{a}_1)(b)(c)b\langle c\rangle.P \mid (\vec{a}_2)(b)b(c).Q \to (\vec{a}_1)(b)P \mid (\vec{a}_2)(b)(c)Q$$

Table 3: Reduction rules.

$Q \equiv (\nu a)P$: Applying the induction hypothesis on P, we have that

$$Q \equiv (\nu a)(\nu \vec{c})((\vec{a}_1)\alpha_1.P_1 \mid (\vec{a}_2)\alpha_2.P_2 \mid \ldots \mid (\vec{a}_k)\alpha_k.P_k).$$

$Q \equiv (a)P$: By induction hypothesis and α-conversion, we have that

$$Q \equiv (a)(\nu \vec{c})((\vec{a}_1)\alpha_1.P_1 \mid (\vec{a}_2)\alpha_2.P_2 \mid \ldots \mid (\vec{a}_k)\alpha_k.P_k),$$

where $a \notin \{c_1, \ldots, c_k\}$. Hence,

$$Q \equiv (\nu \vec{c})((a)(\vec{a}_1)\alpha_1.P_1 \mid \ldots \mid (a)(\vec{a}_k)\alpha_k.P_k).$$

$Q \equiv \alpha.P$: It is in the form (1).

We may then characterize the evolution of systems via a reduction relation, denoted by $\to$, defined as the least relation that satisfies the rules given in Table 3, focusing on the representative cases for synchronization and closing the relation under structural congruence (STRU) and static contexts (PARC), (NEWC), and (AUTC). Rule (COMM) formalizes communication of names, stating that it can be performed only via authorized channel names — notice we single out authorization scopes on channel b both for output and input. Authorization delegation is formalized by rule (AUTH). It meets the following requirements: synchronization is realized via an authorized channel and the emitting process must have the authorization in order to delegate it away (names b and c in the rule, respectively); after sending an authorization for a name the emitting process proceeds (P) falling outside of authorization scope of that name (losing authorization), and after receiving an authorization for a name the receiving process proceeds (Q) under the scope of the received authorization (acquiring authorization). Notice rules (COMM) and (AUTH) address action prefixes up to the relevant authorizations (cf. Proposition 1). We denote by $\to^\star$ the reflexive and transitive closure of $\to$.

We introduce some auxiliary notions in order to syntactically characterize *authorization errors* in our setting. First of all, we define the usual notion of *active contexts* for our calculus:

Definition 1 (Active Context) $\mathscr{C}[\cdot] ::= \cdot \quad | \quad P \mid \mathscr{C}[\cdot] \quad | \quad (\nu a)\mathscr{C}[\cdot] \quad | \quad (a)\mathscr{C}[\cdot]$

Active contexts allow us to talk about any active communication prefixes of a process. Also, we may introduce a predicate that states that an active context authorizes actions on a given channel. More precisely, for a given context and a channel, when the hole of the context is filled with an action on the channel, it is authorised for that action.

Definition 2 (Context Authorization) *For an active context $\mathscr{C}[\cdot]$ and a channel a, the context authorization predicate, denoted $auth(\mathscr{C}[\cdot],a)$, is defined inductively on the structure of $\mathscr{C}[\cdot]$ as*

$$auth(\mathscr{C}[\cdot],a) \triangleq \begin{cases} false & \text{if } \mathscr{C}[\cdot] = \cdot \\ true & \text{if } \mathscr{C}[\cdot] = (a)\mathscr{C}'[\cdot] \\ auth(\mathscr{C}'[\cdot],a) & \text{if } \mathscr{C}[\cdot] = (b)\mathscr{C}'[\cdot] \text{ and } a \neq b \\ auth(\mathscr{C}'[\cdot],a) & \text{if } \mathscr{C}[\cdot] = P \mid \mathscr{C}'[\cdot] \\ auth(\mathscr{C}'[\cdot],a) & \text{if } \mathscr{C}[\cdot] = (\nu b)\mathscr{C}'[\cdot] \end{cases}$$

We may then use active contexts and context authorization to precisely characterize errors in our model, since active contexts allow us to talk about any communication prefix in the process and the context authorization predicate allows to account for the authorizations granted by the context: processes that have active communication prefixes which are not authorized are errors.

Definition 3 (Error) *We say process P is an error if $P \equiv \mathscr{C}[\alpha_a.Q]$ where*

1. *$auth(\mathscr{C}[\cdot],a) = false$ or*
2. *$\alpha_a = a\langle b\rangle$ and $auth(\mathscr{C}[\cdot],b) = false$.*

Notice that by α_a we refer to any communication action on channel a, so intuitively communication actions that may cause stuck configurations according to our semantics (where synchronizations only occur when processes hold the proper authorizations) are seen as errors.

3 Type System

In order to statically single out the processes that can never evolve into authorization errors, we introduce a suitable type system that accounts for the authorizations required by the processes.

Typing Judgment and Typing Rules. Let ρ denote a set of names. The typing judgment $\rho \vdash P$ states that process P is typed if the context provides authorizations for names ρ; hence the process performs actions in the (unauthorized) names ρ (i.e., actions along names not under respective authorization scopes). Thus, $\emptyset \vdash P$ says that all communication actions prescribed by process P are authorized, i.e., occur within the scope of the appropriate authorizations. We say that process P is *well typed* if $\emptyset \vdash P$.

Typing rules are given in Table 4. The inactive process contains no actions along unauthorized channel names (TSTOP). If two processes act along unauthorized channel names ρ_1 and ρ_2, their parallel composition performs actions along the union $\rho_1 \cup \rho_2$ of unauthorized names (TPAR). If a typed process P does not perform actions along a channel a, the process where name a is restricted is typed under the same set of names as P (TNEW). If P acts under a set of unauthorized names ρ, then $(a)P$ authorizes a in P and thus performs actions under the set of unauthorized names $\rho \setminus \{a\}$ (TAUTH). Sending a name along a channel a extends the set of unauthorized names with the name a (TSEND). Receiving a name x along a channel a extends the set of unauthorized names with the name a and it is typed only if there is no unauthorized actions along x within P (TRECV). Sending authorization for a name b along a channel a extends the set of unauthorized names with both names a and b (TDELEG). Receiving authorization for a name b along a name a is typed under the set of unauthorized names that is extended with a and does not contain b (since the reception authorizes b in P).

$$\text{(TSTOP)}\ \emptyset \vdash 0 \qquad \text{(TPAR)}\ \frac{\rho_1 \vdash P \quad \rho_2 \vdash Q}{\rho_1 \cup \rho_2 \vdash P \mid Q} \qquad \text{(TNEW)}\ \frac{\rho \vdash P \quad a \notin \rho}{\rho \vdash (\nu a)P} \qquad \text{(TAUTH)}\ \frac{\rho \vdash P}{\rho \setminus \{a\} \vdash (a)P}$$

$$\text{(TSEND)}\ \frac{\rho \vdash P}{\rho \cup \{a\} \vdash a!b.P} \qquad \text{(TRECV)}\ \frac{\rho \vdash P \quad x \notin \rho}{\rho \cup \{a\} \vdash a?x.P} \qquad \text{(TDELEG)}\ \frac{\rho \vdash P \quad b \notin \rho}{\rho \cup \{a,b\} \vdash a\langle b\rangle.P} \qquad \text{(TRECP)}\ \frac{\rho \vdash P}{(\rho \setminus \{b\}) \cup \{a\} \vdash a(b).P}$$

Table 4: Typing rules.

Main Results. Our main result is *type safety*: well-typed processes never evolve into an (authorization) error, in the sense of Definition 3. This is stated as Corollary 1; before giving the main statement we show its supporting results. We first state a basic property of typing derivations: unauthorized names must be included in the free names of the process.

Proposition 2 *If $\rho \vdash P$ then $\rho \subseteq \mathsf{fn}(P)$.*

Proof (by induction on the depth of the derivation of $\rho \vdash P$)
If $\emptyset \vdash 0$ then $\mathsf{fn}(0) = \emptyset$.
The following cases follow by definition of free names and the induction hypothesis.
Case $\rho_1 \cup \rho_2 \vdash P \mid Q$ is derived from $\rho_1 \vdash P$ and $\rho_2 \vdash Q$: $\mathsf{fn}(P \mid Q) = \mathsf{fn}(P) \cup \mathsf{fn}(Q) \supseteq \rho_1 \cup \rho_2$.
Case $\rho \vdash (\nu a)P$ is derived from $\rho \vdash P$ and $a \notin \rho$: $\mathsf{fn}((\nu a)P) = \mathsf{fn}(P) \setminus \{a\} \supseteq \rho \setminus \{a\} = \rho$.
Case $\rho \setminus \{a\} \vdash (a)P$ is derived from $\rho \vdash P$: $\mathsf{fn}((a)P) = \mathsf{fn}(P) \cup \{a\} \supseteq \rho \cup \{a\} \supseteq \rho \setminus \{a\}$.
Case $\rho \cup \{a\} \vdash a!b.P$ is derived from $\rho \vdash P$: $\mathsf{fn}(a!b.P) = \mathsf{fn}(P) \cup \{a\} \supseteq \rho \cup \{a\}$.
Case $\rho \cup \{a\} \vdash a?x.P$ is derived from $\rho \vdash P$ and $x \notin \rho$: $\mathsf{fn}(a?x.P) = (\mathsf{fn}(P) \setminus \{x\}) \cup \{a\} \supseteq (\rho \setminus \{x\}) \cup \{a\} = \rho \cup \{a\}$.
Case $\rho \cup \{a,b\} \vdash a\langle b\rangle.P$ is derived from $\rho \vdash P$ and $b \notin \rho$: $\mathsf{fn}(a\langle b\rangle.P) = \mathsf{fn}(P) \cup \{a,b\} \supseteq \rho \cup \{a,b\}$.
Case $(\rho \setminus \{b\}) \cup \{a\} \vdash a(b).P$ is derived from $\rho \vdash P$: $\mathsf{fn}(a(b).P) = \mathsf{fn}(P) \cup \{a,b\} \supseteq \rho \cup \{a,b\} \supseteq (\rho \setminus \{b\}) \cup \{a\}$.

We now state results used to prove that typing is preserved under system evolution, namely that (i) typing is preserved under structural congruence, as reduction is closed under structural congruence, and that (ii) typing is preserved under name substitution, since channel passing involves name substitution.

Lemma 1 (Inversion Lemma) *1. If $\rho \vdash 0$ then $\rho = \emptyset$.*

2. If $\rho \vdash P \mid Q$ then there are ρ_1 and ρ_2 such that $\rho = \rho_1 \cup \rho_2$ and $\rho_1 \vdash P$ and $\rho_2 \vdash Q$.

3. If $\rho \vdash (\nu a)P$ then $\rho \vdash P$ and $a \notin \rho$.

4. If $\rho \vdash (a)P$ then there is ρ' such that $\rho = \rho' \setminus \{a\}$ and $\rho' \vdash P$.

5. If $\rho \vdash a!b.P$ then there is ρ' such that $\rho = \rho' \cup \{a\}$ and $\rho' \vdash P$.

6. If $\rho \vdash a?x.P$ then there is ρ' such that $\rho = \rho' \cup \{a\}$ and $x \notin \rho'$ and $\rho' \vdash P$.

7. If $\rho \vdash a\langle b\rangle.P$ then there is ρ' such that $\rho = \rho' \cup \{a,b\}$ and $b \notin \rho'$ and $\rho' \vdash P$.

8. If $\rho \vdash a(b).P$ then there is ρ' such that $\rho = (\rho' \setminus \{b\}) \cup \{a\}$ and $\rho' \vdash P$.

Lemma 2 (Subject Congruence) *If $\rho \vdash P$ and $P \equiv Q$ then $\rho \vdash Q$.*

Proof (by induction on the depth of the derivation of $P \equiv Q$)

We only write the following two interesting cases, and other cases can be obtained by similar reasoning.

Case $P \mid (\nu a)Q \equiv (\nu a)(P \mid Q)$ and $a \notin \mathsf{fn}(P)$:

From $\rho \vdash P \mid (\nu a)Q$, by Lemma 1. 2, there are ρ_1 and ρ_2 such that $\rho_1 \vdash P$ and $\rho_2 \vdash (\nu a)Q$. Therefore, by Lemma 1. 3, $\rho_2 \vdash Q$ and $a \notin \rho_2$. Since $a \notin \mathsf{fn}(P)$ we conclude by Proposition 2 that $a \notin \rho_1$. By the rule (TPAR) we get that $\rho_1 \cup \rho_2 \vdash P \mid Q$, and from $a \notin \rho_1 \cup \rho_2$, by (TNEW) we derive $\rho \vdash (\nu a)(P \mid Q)$.

Case $(a)(\nu b)P \equiv (\nu b)(a)P$ and $a \neq b$:

If $\rho \vdash (a)(\nu b)P$ then by Lemma 1. 4 there is ρ' such that $\rho' \vdash (\nu b)P$ and $\rho = \rho' \setminus \{a\}$. By Lemma 1. 3, $\rho' \vdash P$ and $b \notin \rho'$ (and so $b \notin \rho' \setminus \{a\}$). Hence, by (TAUTH) and (TNEW), we get $\rho \vdash (\nu b)(a)P$.

Lemma 3 (Substitution) *If $\rho \vdash P$ then $\rho\{a/b\} \vdash P\{a/b\}$.*

Proof (by induction on the depth of the derivation of $\rho \vdash P$)

We give only one interesting case. If $\rho \cup \{b,c\} \vdash c\langle b\rangle.P$ is derived from $\rho \vdash P$ and $b \notin \rho$. It holds that $(\rho \cup \{b,c\})\{a/b\} = \rho \cup \{a,c\}$ and $(c\langle b\rangle.P)\{a/b\} = c\langle a\rangle.P\{a/b\}$. By induction hypothesis, $\rho\{a/b\} \vdash P\{a/b\}$, and by the rule (TDELEG), $\rho \cup \{a,c\} \vdash c\langle a\rangle.P\{a/b\}$.

We may now state our soundness result which ensures typing is preserved under reduction.

Theorem 1 (Subject reduction) *If $\rho \vdash P$ and $P \to Q$ then $\rho \vdash Q$.*

Proof (by induction on the depth of the derivation of $P \to Q$)

Base case 1: Assume that $\rho \vdash (\vec{a}_1)(b)b!c.P \mid (\vec{a}_2)(b)b?x.Q$ and

$$\text{(COMM)} \qquad (\vec{a}_1)(b)b!c.P \mid (\vec{a}_2)(b)b?x.Q \to (\vec{a}_1)(b)P \mid (\vec{a}_2)(b)Q\{c/x\}.$$

By Lemma 1. 2, there are ρ_1 and ρ_2 such that $\rho = \rho_1 \cup \rho_2$ and

$$\rho_1 \vdash (\vec{a}_1)(b)b!c.P \quad \text{and} \quad \rho_2 \vdash (\vec{a}_2)(b)b?x.Q.$$

By consecutive application of Lemma 1. 4, there are ρ_1' and ρ_2' such that $\rho_1 = \rho_1' \setminus \{\vec{a}_1, b\}$ and $\rho_2 = \rho_2' \setminus \{\vec{a}_2, b\}$ and

$$\rho_1' \vdash b!c.P \quad \text{and} \quad \rho_2' \vdash b?x.Q.$$

By Lemma 1. 5-6, there are ρ_1'' and ρ_2'' such that $\rho_1' = \rho_1'' \cup \{b\}$ and $\rho_2' = \rho_2'' \cup \{b\}$ and $x \notin \rho_2''$ and

$$\rho_1'' \vdash P \quad \text{and} \quad \rho_2'' \vdash Q.$$

One should notice that $\rho_1 = (\rho_1'' \cup \{b\}) \setminus \{\vec{a}_1, b\} = \rho_1'' \setminus \{\vec{a}_1, b\}$ and $\rho_2 = (\rho_2'' \cup \{b\}) \setminus \{\vec{a}_2, b\} = \rho_2'' \setminus \{\vec{a}_2, b\}$ and $\rho_2''\{c/x\} = \rho_2''$ (since $x \notin \rho_2''$). By Lemma 3 and consecutive application of the typing rules (TAUTH) we get

$$\rho_1 \vdash (\vec{a}_1)(b)P \quad \text{and} \quad \rho_2 \vdash (\vec{a}_2)(b).Q\{c/x\}.$$

and finally, by (TPAR), we have $\rho \vdash (\vec{a}_1)(b)P \mid (\vec{a}_2)(b).Q\{c/x\}$.

Base case 2: Assume that $\rho \vdash (\vec{a}_1)(b)(c)b\langle c\rangle.P \mid (\vec{a}_2)(b)b(c).Q$ and

$$\text{(AUTH)} \qquad (\vec{a}_1)(b)(c)b\langle c\rangle.P \mid (\vec{a}_2)(b)b(c).Q \to (\vec{a}_1)(b)P \mid (\vec{a}_2)(b)(c)Q$$

By Lemma 1. 2, there are ρ_1 and ρ_2 such that $\rho = \rho_1 \cup \rho_2$ and

$$\rho_1 \vdash (\vec{a}_1)(b)(c)b\langle c\rangle.P \quad \text{and} \quad \rho_2 \vdash (\vec{a}_2)(b)b(c).Q$$

By consecutive application of Lemma 1. 4, there are ρ_1' and ρ_2' such that $\rho_1 = \rho_1' \setminus \{\vec{a}_1, b, c\}$ and $\rho_2 = \rho_2' \setminus \{\vec{a}_2, b\}$ and

$$\rho_1' \vdash b\langle c\rangle.P \quad \text{and} \quad \rho_2' \vdash b(c).Q$$

By Lemma 1. 7-8, there are ρ_1'' and ρ_2'' such that $\rho_1' = \rho_1'' \cup \{b, c\}$ and $c \notin \rho_1''$ and $\rho_2' = (\rho_2'' \setminus \{c\}) \cup \{b\}$ and

$$\rho_1'' \vdash P \quad \text{and} \quad \rho_2'' \vdash Q.$$

We conclude that $\rho_1 = (\rho_1'' \cup \{b, c\}) \setminus \{\vec{a}_1, b, c\} = \rho_1'' \setminus \{\vec{a}_1, b\}$ (since $c \notin \rho_1''$) and $\rho_2 = \rho_2'' \setminus \{\vec{a}_2, b, c\}$. By consecutive application of the typing rule (TAUTH), we get

$$\rho_1 \vdash (\vec{a}_1)(b)P \quad \text{and} \quad \rho_2 \vdash (\vec{a}_2)(b)(c)Q$$

and by (TPAR)

$$\rho \vdash (\vec{a}_1)(b)P \mid (\vec{a}_2)(b)(c)Q.$$

Case 3: Assume that $\rho \vdash P \mid R$ and $P \mid R \to Q \mid R$ is derived from $P \to Q$. By Lemma 1. 2, there are ρ_1 and ρ_2 such that $\rho = \rho_1 \cup \rho_2$ and

$$\rho_1 \vdash P \quad \text{and} \quad \rho_2 \vdash R.$$

By induction hypothesis, it holds that $\rho_1 \vdash Q$ and therefore we have, by (TPAR), that $\rho \vdash Q \mid R$.

Case 4: Assume that $\rho \vdash (\nu a)P$ and $(\nu a)P \to (\nu a)Q$ is derived from $P \to Q$. By Lemma 1. 3, it holds that $a \notin \rho$ and $\rho \vdash P$. By induction hypothesis, it holds that $\rho \vdash Q$ and therefore, by (TNEW), we get $\rho \vdash (\nu a)Q$.

Case 5: Assume that $\rho \vdash (a)P$ and $(a)P \to (a)Q$ is derived from $P \to Q$. By Lemma 1. 4, it holds that there is ρ' such that $\rho = \rho' \setminus \{a\}$ and $\rho' \vdash P$. By induction hypothesis, it holds that $\rho' \vdash Q$ and therefore, by (TAUTH), we get $\rho \vdash (a)Q$.

Case 6: Assume that $\rho \vdash P$ and $P \to Q$ is derived from $P' \to Q'$, where $P \equiv P'$ and $Q' \equiv Q$. We conclude by Lemma 2 that $\rho \vdash P'$. Than, by induction hypothesis, we get $\rho \vdash Q'$, and applying again Lemma 2, we have that $\rho \vdash Q$.

Theorem 1 ensures, considering $\rho = \emptyset$, that well-typed processes always reduce to well-typed processes. We now express the basic property for well-typed systems, namely that they do not expose any authorization errors up to the ones granted by pending authorizations ρ.

Proposition 3 (Error Free) *If $\rho \vdash \mathscr{C}[\alpha_a.Q]$ and $a \notin \rho$ then $auth(\mathscr{C}[\cdot], a)$.*

Proof (by induction on the structure of $\mathscr{C}[\cdot]$)

Case $\mathscr{C}[\cdot] = [\cdot]$: If $\rho \vdash \alpha_a.Q$ we conclude that $a \in \rho$, by Lemma 1.
Case $\mathscr{C}[\cdot] = P \mid \mathscr{C}'[\cdot]$: If $\rho \vdash P \mid \mathscr{C}'[\alpha_a.Q]$, by Lemma 1. 2, there are ρ_1 and ρ_2 such that $\rho = \rho_1 \cup \rho_2$ and $\rho_1 \vdash P$ and $\rho_2 \vdash \mathscr{C}'[\alpha_a.Q]$. By induction hypothesis, $auth(\mathscr{C}'[\cdot], a)$, while by definition $auth(P \mid \mathscr{C}'[\cdot], a) = auth(\mathscr{C}'[\cdot], a)$.
Case $\mathscr{C}[\cdot] = (\nu b)\mathscr{C}'[\cdot]$: If $a \notin \rho$ and $\rho \vdash (\nu b)\mathscr{C}'[\alpha_a.Q]$, by Lemma 1. 3, $\rho \vdash \mathscr{C}'[\alpha_a.Q]$ and $b \notin \rho$. By induction hypothesis, $auth(\mathscr{C}'[\cdot], a)$. By definition, $auth((\nu b)\mathscr{C}'[\cdot], a) = auth(\mathscr{C}'[\cdot], a)$.
Case $\mathscr{C}[\cdot] = (b)\mathscr{C}'[\cdot]$ and $a \neq b$: If $a \notin \rho$ and $\rho \vdash (b)\mathscr{C}'[\alpha_a.Q]$, by Lemma 1. 4, there is ρ' such that $\rho = \rho' \setminus \{b\}$ and $\rho' \vdash \mathscr{C}'[\alpha_a.Q]$. If $a \neq b$ and $a \notin \rho$ then $a \notin \rho'$. By induction hypothesis, $auth(\mathscr{C}'[\cdot], a)$. By definition, $auth((b)\mathscr{C}'[\cdot], a) = auth(\mathscr{C}'[\cdot], a)$.
Case $\mathscr{C}[\cdot] = (a)\mathscr{C}'[\cdot]$: By definition $auth((a)\mathscr{C}'[\cdot], a) = true$.

Proposition 3 thus ensures that active communication prefixes that do not involve a pending authorization (outside of ρ) are not errors. Considering $\rho = \emptyset$ we thus have that well-typed processes do not have any unauthorized prefixes and thus are not errors. Along with Theorem 1 we may then state our safety result which says well-typed processes never evolve into an error.

Corollary 1 (Type Safety) *If $\emptyset \vdash P$ and $P \rightarrow^\star Q$ then Q is not an error.*

Proof Immediate from Theorem 1 and Proposition 3.

Corollary 1 attests that well-typed systems never reduce to authorization errors, including when authorizations are dynamically delegated. The presented type system allows for a streamlined analysis on process authorization requirements, which we intend to exploit as the building block for richer analysis.

4 Concluding Remarks

The work presented here builds on our previous work [7], in which we explored authorization passing in the context of communication-centered systems. In [7], the analysis addressed not only authorization passing but also role-based protocol specification, building on the conversation type analysis presented in [2]. Here we focussed exclusively on the authorization problem, obtaining a simple model which paves the way for further investigation, since the challenges involved may now be highlighted in a crisper way. As usual, there are non typable processes that are authorized for all the actions and reduce to 0. This is unsurprising, given the simplicity of the analysis. An example is

$$(a)(b)(b?x.x!b.0 \mid b!a.a?x.0).$$

For the same reason, even though our untyped model enables to keep existing copies of delegated authorization scopes, the type system restricts the usage of their scopes. For example, the current discipline can not type the process

$$(b)(a)(a)b\langle a\rangle a!b.0 \mid (b)b(a)a?x.0$$

even though it safely reduces to 0. We believe it would be interesting to enrich the typing analysis so that it encompasses the contextual information (authorizations already held by the process) so as to address name reception and authorization delegation in a different way. We also believe it would be

interesting to discipline authorization usage so as to ensure absence of double authorizations for the sake of authorization accountability, so as to ensure only the strictly necessary authorizations are specified.

Naturally, it would also be interesting to integrate the analysis presented here in richer settings, for instance (i) considering the need to ensure protocol fidelity using session types [10], or (ii) ensuring liveness properties so that security critical events are guaranteed to take place, or (iii) exploring an ontology on names so that authorizations to act upon higher ranked names automatically yield authorization for lower-level ones. While relevant, these extensions appear as orthogonal developments to the analysis presented here and therefore should be studied in depth in a dedicated way.

We briefly review some related works. Scoping operators have been widely used for the purpose of modeling security aspects (e.g., [8]) but typically they use bound names (e.g., to model secrets). With the aim of representing secrecy and confidentiality requirements in process specifications, an alternative scoping operator called *hide* is investigated in [8]. The hide operator is embedded in the so-called secrecy π-calculus, tailored to program secrecy in communications. The expressiveness of the hide operator is investigated in the context of a behavioral theory, by means of an Spy agent. In contrast, our (free name) scoping operator focuses on authorization, a different security concern. In [13], a scoping operator (called *filter*) is proposed for dynamic channel screening. In a different setting (higher-order communication) and with similar properties, the filter operator blocks all the actions that are not contained in the corresponding filter (which contains polarised channel names). Contrary to the authorization scope, filters are statically assigned to processes, while the authorization scope assigned to a process may be dynamically changed. To the best of our knowledge, the authorization scoping proposed here has not been explored before for the specification of communication-centered systems.

One key idea explored here is the separation between resource and the respective authorization which in particular allows us to distinguish the communication of the resource handler from the resource authorization. Consider, e.g., process $a?x.b!x.0$, where a forwarding process receives a channel and forwards it without necessarily being authorized to interact in it. This allows to model scenarios where resource handlers may be passed around in unverified contexts, since their unauthorized use is excluded. This example in particular distinguishes the type-based authorization handling presented in [9], since authorizations directly flow in the communications (via the types) and cannot be received afterwards — we leave to future work a comparison with more *refined* typing notions such as [6, 12] where we conceive that dependencies between received values can address such a separation, albeit in a more indirect way.

To further remark on the particularities of our linguistic constructors consider process

$$a?x.(x)P$$

where P is authorized to act on channel x, regardless of the identity of the channel actually received, which somewhat hints on the particular combination between the (non-binding) scoping operator and name passing. While we do not claim that our constructs cannot be encoded in other models, we do believe they provide an adequate abstraction level to reason on authorization handling. Still, it would be interesting to see how to express authorization scopes and authorization communication (including delegation) using π-calculus like models (such as, e.g., [1, 3, 5]).

There are high-level similarities between our work and the concept of *ownership types*, as well studied for object-oriented languages [4]. Although in principle ownership types focus on static ownership structures, assessing their use for disciplining dynamic authorizations is interesting future work.

Acknowledgments. We thank the anonymous referees for their insightful and useful remarks. This work was supported by COST Action IC1201: Behavioural Types for Reliable Large-Scale Software

Systems (BETTY) via Short-Term Scientific Mission grants (to Pantović and Vieira). Pérez is also affiliated to the NOVA Laboratory for Computer Science and Informatics (NOVA LINCS), Universidade Nova de Lisboa, Portugal.

References

[1] Martín Abadi & Cédric Fournet (2001): *Mobile values, new names, and secure communication*. In: *Conference Record of POPL 2001: The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ACM, pp. 104–115, doi:10.1145/360204.360213.

[2] Pedro Baltazar, Luís Caires, Vasco T. Vasconcelos & Hugo Torres Vieira (2012): *A Type System for Flexible Role Assignment in Multiparty Communicating Systems*. In: *Trustworthy Global Computing - 7th International Symposium, TGC 2012, Revised Selected Papers*, *LNCS* 8191, Springer, pp. 82–96, doi:10.1007/978-3-642-41157-1_6.

[3] Marco Carbone & Sergio Maffeis (2002): *On the Expressive Power of Polyadic Synchronisation in pi-calculus*. *Electr. Notes Theor. Comput. Sci.* 68(2), pp. 15–32, doi:10.1016/S1571-0661(05)80361-5.

[4] David G. Clarke, John Potter & James Noble (1998): *Ownership Types for Flexible Alias Protection*. In: *Proceedings of the 1998 ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages & Applications, OOPSLA 1998*, ACM, pp. 48–64, doi:10.1145/286936.286947.

[5] Silvano Dal-Zilio & Andrew D. Gordon (2002): *Region analysis and a pi-calculus with groups*. *J. Funct. Program.* 12(3), pp. 229–292, doi:10.1017/S0956796801004270.

[6] Juliana Franco & Vasco Thudichum Vasconcelos (2013): *A Concurrent Programming Language with Refined Session Types*. In: *Software Engineering and Formal Methods - SEFM 2013 Collocated Workshops: BEAT2, WS-FMDS, FM-RAIL-Bok, MoKMaSD, and OpenCert, Revised Selected Papers*, *LNCS* 8368, Springer, pp. 15–28, doi:10.1007/978-3-319-05032-4_2.

[7] Silvia Ghilezan, Svetlana Jaksic, Jovanka Pantovic, Jorge A. Pérez & Hugo Torres Vieira (2014): *Dynamic Role Authorization in Multiparty Conversations*. In: *Proceedings Third Workshop on Behavioural Types, BEAT 2014*, *EPTCS* 162, pp. 1–8, doi:10.4204/EPTCS.162.1.

[8] Marco Giunti, Catuscia Palamidessi & Frank D. Valencia (2012): *Hide and New in the Pi-Calculus*. In: *Proceedings of the Combined 19th International Workshop on Expressiveness in Concurrency and 9th Workshop on Structured Operational Semantics, EXPRESS/SOS 2012*, *EPTCS* 89, pp. 65–79, doi:10.4204/EPTCS.89.6.

[9] Daniele Gorla & Rosario Pugliese (2009): *Dynamic management of capabilities in a network aware coordination language*. *J. Log. Algebr. Program.* 78(8), pp. 665–689, doi:10.1016/j.jlap.2008.12.001.

[10] Kohei Honda, Vasco Thudichum Vasconcelos & Makoto Kubo (1998): *Language Primitives and Type Discipline for Structured Communication-Based Programming*. In: *Programming Languages and Systems, 7th European Symposium on Programming, ESOP 1998, Proceedings*, *LNCS* 1381, Springer, pp. 122–138, doi:10.1007/BFb0053567.

[11] Davide Sangiorgi & David Walker (2001): *The Pi-Calculus - a theory of mobile processes*. Cambridge University Press.

[12] Nikhil Swamy, Juan Chen & Ravi Chugh (2010): *Enforcing Stateful Authorization and Information Flow Policies in Fine*. In: *Programming Languages and Systems, 19th European Symposium on Programming, ESOP 2010, Proceedings*, *LNCS* 6012, Springer, pp. 529–549, doi:10.1007/978-3-642-11957-6_28.

[13] José-Luis Vivas & Nobuko Yoshida (2002): *Dynamic Channel Screening in the Higher Order pi-Calculus*. *Electr. Notes Theor. Comput. Sci.* 66(3), pp. 170–184, doi:10.1016/S1571-0661(04)80421-3.