Rôle-Based Access Control for a Distributed Calculus

Chiara Braghin
Dip. Informatica
Univ. "Ca' Foscari" di Venezia

Daniele Gorla
Dip. di Informatica
Univ. di Roma "La Sapienza"

Vladimiro Sassone
Dept. of Informatics
University of Sussex

Abstract

Rôle-based access control (RBAC) is increasingly attracting attention because it reduces the complexity and cost of security administration by interposing the notion of *rôle* in the assignment of permissions to users. In this paper, we present a formal framework relying on an extension of the π-calculus to study the behaviour of concurrent systems in a RBAC scenario. We define a type system ensuring that the specified policy is respected during computations, and a behavioural equivalence to equate systems. We then consider a more sophisticated feature that can be easily integrated in our framework, i.e., the possibility of automatically adding rôle activations and deactivations to processes to be run under a given policy (whenever possible). Finally, we show how the framework can be easily extended to express significant extensions of the core RBAC model, such as rôles hierarchies or constraints determining the acceptability of the system components.

Keywords: RBAC, Process Calculi, Type Systems, Behavioural Equivalences

Introduction

Rôle-based access control (RBAC) [10, 23] has recently emerged as an alternative to classical discretionary and mandatory access controls: a standard is currently under development by the National Institute of Standards and Technology (NIST) [11] and several commercial applications directly support some forms of RBAC, e.g., Oracle, Informix and Sybase in the field of commercial database management systems. Furthermore, the RBAC technology is finding applications in areas ranging from health-care to defence, in addition to the commerce systems for which it was originally designed.

RBAC is a flexible and policy-neutral access control technology: it regulates the access of users to information and system resources on the basis of activities they need to execute in the system. The essence of RBAC lies with the notions of *user*, *rôle* and *permission*: users are authorised to use only the permissions assigned to the rôles they belong to. More specifically, RBAC allows for a preliminary assignment of permissions to rôles (thus abstracting from which users will play the various rôles at run-time). A user may then establish multiple sessions, e.g., by signing on to the system, during which he activates a subset of rôles that he is a member of. This greatly simplifies

system management, as it reduces the cost of administering access control policies, as well as making the administration process less error-prone. In fact, by assigning to users predefined rôles that naturally express the organisation's structure, the administrative process of establishing permissions is streamlined, and management time for reviewing permissions assignment is reduced. Anyway, the complexity of the models (e.g., in large systems the number of rôles can exceed hundreds or thousands) demands a structured approach to the analysis and design of such systems.

This paper aims at developing a theory for reasoning about system behaviours in a RBAC scenario; to the best of our knowledge this is the first attempt in this direction. Our reference model is the so-called RBAC96 model, introduced by Sandhu et al. in their seminal paper [23]. More advanced RBAC models include rôle hierarchies and constraints such as rôle mutual exclusion, separation of duties, delegation of authority and negative permissions. Our starting point is the π-calculus [24], which provides well-established mathematical tools for expressing concurrent and possibly distributed systems. Essentially, our idea is to equip the π-calculus with the notion of *users*: we tag processes with a (not necessarily unique) name representing the user that activated them – this is very similar to the located threads of the Dπ [14]. Moreover, we add two new constructs, that enable processes to activate/deactivate rôles in the user session where they run, and we include a way to grant permissions to rôles. Thus, each process is associated with a name (representing the user owning it) and with a set ρ recording the rôles activated during the current session. Hence, the term $r\{\!|\, P \,|\!\}_\rho$ represents a session of the user r, running a process P with active rôles ρ. We model rôle's activation/deactivation by exploiting the following reductions:

$$r\{\!|\, \mathbf{role}\, R.P \,|\!\}_\rho \;\longmapsto\; r\{\!|\, P \,|\!\}_{\rho \cup \{R\}} \qquad\qquad r\{\!|\, \mathbf{yield}\, R.P \,|\!\}_\rho \;\longmapsto\; r\{\!|\, P \,|\!\}_{\rho \setminus \{R\}}$$

Intuitively, when a process activates a rôle R during a session, R must be added to the set of activated rôles ρ and the continuation P will be executed with the set ρ updated. Vice versa for the deactivation of R.

As an example, the following system

$$client\{\!|\, \mathbf{role}\, \texttt{auth_client}.port_80\langle index.html\rangle.P \,|\!\}_\rho \parallel server\{\!|\, port_80(x).Q \,|\!\}_{\rho'}$$

models the interaction between a client and an HTTP server. The system contains two users, *client* and *server*, running in parallel. It may evolve as follows. First, user *client* activates the rôle `auth_client` by exercising the **role** action, which in practice would involve to authenticate himself by means of a secure certificate. Then, he sends the request to the HTTP server along the usual port 80 by performing an output action along the channel *port_80*.

The introduction of named users immediately suggests the idea of a distributed system. In such systems, as e.g. the Internet, the notion of global, non-located channels as *port_80* is quite an abstraction over what is realistically achievable. We therefore use a notion of localised channels *à la* Dπ [14], where each channel is associated with a single user. Syntactically, we implement this feature by tagging output actions to specify the user (or location) where the exchange is supposed to take place. On the other hand, input actions are not tagged with any user name, as they are supposed to

take place locally. Thus, the example above may be rewritten as:

$$client\{\!\!|\ \textbf{role}\ \texttt{auth_client}.port_80^{server}\langle index.html\rangle.P\ |\!\!\}_\rho \parallel server\{\!\!|\ port_80(x).Q\ |\!\!\}_{\rho'}$$

We also allow user names to be exchanged during communications. This feature adds flexibility and realism to the language, since in distributed systems users have only a partial and evolving knowledge of their execution environment. For example, the client above can be generalised to leave the server identity unspecified and to dynamically retrieve it with an input from channel *choose_a_server*:

$$client\{\!\!|\ \textbf{role}\ \texttt{auth_client}.choose_a_server(x).port_80^{x}\langle index.html\rangle.P\ |\!\!\}_\rho$$

More details on the calculus, together with an illustrative example, will be given in Section 1.

The mapping among users, rôles and permissions, which controls the access of subjects to objects, is achieved by a pair of relations $(\mathcal{U}\,;\mathcal{P})$, called *RBAC schema*. In $(\mathcal{U}\,;\mathcal{P})$, the relation $\mathcal{U}$ is the rôles-to-users association, while $\mathcal{P}$ is the permissions-to-rôles association. As a first contribution of this paper, in Section 2 we define a type system which complements the dynamics of the calculus: it provides us with static guarantees that systems not respecting a given RBAC schema are rejected. In the client/server example above, a client not authenticated (i.e. interacting with the server without having previously performed a **role** `auth_client`) would be rejected, if the RBAC schema enabled only authorised users to perform HTTP requests.

As a second contribution of this paper, in Section 3 we study the behavioural semantics of the calculus via a standardly defined *(typed) barbed congruence*. The behavioural semantics allows us to study the behaviour of systems, concentrating on their functionalities while abstracting from their syntax. In particular, the barbed congruence allows us to prove some interesting algebraic laws that hold in our framework. As an example, we show how RBAC schemata may change the algebraic theory of the π-calculus. Consider the following system, adapted from the client/server example above:

$$(\nu\, port_80^{server}\!:R)(client\{\!\!|\ port_80^{server}\langle index.html\rangle.P\ |\!\!\}_\emptyset \parallel server\{\!\!|\ port_80(x).Q\ |\!\!\}_{\rho'})$$

where $(\nu\, port_80^{server}:R)$ is the standard restriction operator of a typed π-calculus (it declares $port_80^{server}$ at type R and limits the visibility of the channel to *client* and *server* only). By resuming the assumption that only authorised users can perform HTTP requests, the above system is blocked because the client has not been authenticated before performing the output. On the contrary, by removing the assumption that each action must be authorised by the activation of a proper rôle, the term above would have been equivalent to

$$(\nu\, port_80^{server}\!:R)(client\{\!\!|\ P\ |\!\!\}_\emptyset \parallel server\{\!\!|\ Q[^{index.html}/_{x}]\ |\!\!\}_{\rho'})$$

that is the term resulting from the client/server exchange ($Q[^{index.html}/_{x}]$ denotes the process Q where each occurrence of x has been replaced by the value *index.html*). By the way, this is exactly what would have happened in a similar term of the π-calculus, since the name *port_80* is restricted and no authorisation is needed to perform input/output actions.

As highlighted by the example above, the essence of our calculus resides in the assumption that each action can be performed only if a privilege enabling it is available in the user session where the action is executed. Since privileges are associated to rôles, it is fundamental to properly program rôle activations and deactivations within user sessions. To this aim, in Section 4 we describe an algorithm to automatically add rôle activations/deactivations within a system in such a way that the resulting system can be executed under a given schema, whenever possible.

In Section 5, we describe how our simple framework can be extended to express extensions of the core RBAC model. For example, rôles can be hierarchically ordered to reflect in a natural way the different levels of authority, responsibility and competency of the employees working in an enterprise. Moreover, the system administrator may want to enforce constraints limiting the set of rôles that can be activated during a session. Both extensions can be expressed in a uniform and scalable way by enriching the RBAC schema.

We conclude by comparing our approach with related work in Section 6. Appendix A contains the proofs of some results stated in the paper, while Appendix B provides a sound proof technique for barbed congruence in terms of a labelled transition system and a labelled bisimulation.

This paper is an extended and revised version of [4]; with respect to the extended abstract, in this paper we give all the technical details and proofs, we expand one of the possible examples of our framework, and we show how advanced RBAC features may be added in a modular way to the general picture.

1 The Language

In this section we formally introduce our calculus. First, we define its syntax and operational semantics; then, we formalise the RBAC schema to describe the rôles-to-users and permissions-to-rôles assignments.

1.1 Syntax

Since the calculus is an extension of the π-calculus [19, 24], we assume the reader to be somehow acquainted with its basic features. The syntax of the calculus is given in Table 1; we assume two countable and pairwise disjoint sets: $\mathcal{R}$ of *rôles*, ranged over by $R, S, \ldots$, and $\mathcal{N}$ of *names*. Names can serve three (logically) different purposes: they can be used as user names (in this case, we prefer letters $r, s, \ldots$), channel names (in this case, we prefer letters $a, b, \ldots$) or input variables (in this case, we prefer letters $x, y, ...$). As we discussed in the Introduction, channels are associated with users. Thus, the set of *values* of our calculus includes not only raw names but also pairs of names, written a^r; such pairs are called *channels* and include the name of the channel, a, and the user it is associated with, r.

Processes **nil**, $P \mid Q$, $!P$, $[m = n]P$, $(\nu a : R)P$, $a(x).P$, $m\langle n\rangle.P$ are derived from the corresponding π-like constructs. They represent, respectively, the inactive process, parallel composition of processes, replication (to model recursive processes), value matching, restriction of channel names and input/output actions over channels. Notice

Rôles	$R, S, ... \in \mathcal{R}$			
Names	$a, b, ..., r, s, ..., x, y, ... \in \mathcal{N}$			
Values	$m, n, ... \in \mathcal{N} \cup \mathcal{N} \times \mathcal{N}$			
Processes	P, Q	$::=$	$\mathbf{nil}$	*inactive process*
		$\mid$	$P \mid Q$	*parallel composition*
		$\mid$	$!P$	*replication*
		$\mid$	$(\nu a : R)P$	*name restriction*
		$\mid$	$[m = n]P$	*value matching*
		$\mid$	$a(x).P$	*input*
		$\mid$	$m\langle n\rangle.P$	*output*
		$\mid$	$\mathbf{role}\, R.P$	*rôle activation*
		$\mid$	$\mathbf{yield}\, R.P$	*rôle deactivation*
Systems	A, B	$::=$	$\mathbf{0}$	*empty system*
		$\mid$	$r\{\!\mid P \mid\!\}_\rho$	*user session*
		$\mid$	$A \parallel B$	*parallel composition*
		$\mid$	$(\nu a^r : R)A$	*channel restriction*

Table 1: Syntax of the Calculus

that input channels are not decorated with a user name: this is a syntactic means to localise them, as input channels implicitly belong to the user they appear in. The main novelty of the calculus resides in the actions **role** R and **yield** R that implement activations/deactivations of rôles in the user session they belong to, and modify the session rôles accordingly.

The syntax of processes we have just presented is too permissive, as it also contains meaningless terms. For example, when a name represents a channel, it cannot be transmitted as such, since it makes little sense without the indication of the user owning it. Similarly, output channels must indicate the name of the user containing the invoked channel. For example, a process like $a(x).b^x\langle n\rangle.P$ can be accepted but, in order to be executed, at run-time x must be assigned a user name r which owns an input channel b^r. One of the aims of the type system in Section 2 is to restrict the admissible language terms, thus rejecting terms that contain any kind of anomalies.

Systems consist of the parallel composition of user sessions that can share private channels (the latter ones are decorated with a rôle as described later, in Section 1.3). A user session $r\{\!\mid P \mid\!\}_\rho$ represents a process spawned by a user named r, with code P and with $\rho \subseteq \mathcal{R}$ recording the rôles activated so far. Observe that different sessions of the same user can run in parallel within a system A, either with the same or with different activated rôles: this is the usual notion of sessions in RBAC models.

The constructs $(\nu a : R)P$, $(\nu a^r : R)A$ and $a(x).P$ act as binders for a, a^r and x, respectively. Thus, we need to extend the standard notion of free and bound names of the π-calculus to encompass free and bound channels too. The formal definition of functions F (A) and B (A) is given in Table 2; it exploits the auxiliary functions F $_r(P)$ and B $_r(P)$ for processes of user r. Alpha-conversion, written $=_\alpha$, is then standardly defined and it allows the renaming of bound channels and names. Throughout the paper, we always assume that bound channels and names are pairwise distinct and dif-

System	F $(_)$	B $(_)$
$\mathbf{0}$	$\emptyset$	$\emptyset$
$r\{\!\| P \|\!\}_\rho$	$\{r\} \cup$ F $_r(P)$	B $_r(P)$
$A \parallel B$	F $(A) \cup$ F (B)	B $(A) \cup$ B (B)
$(\nu a^r : R)A$	F $(A) \setminus \{a^r\}$	B $(A) \cup \{a^r\}$

Process	F $_r(_)$	B $_r(_)$
nil	$\emptyset$	$\emptyset$
$a(x).P$	$\{a^r\} \cup$ F $_r(P)$	$\{x\} \cup$ B $_r(P)$
$m\langle n\rangle.P$	$\{m, n\} \cup$ F $_r(P)$	B $_r(P)$
role $R.P$	F $_r(P)$	B $_r(P)$
yield $R.P$	F $_r(P)$	B $_r(P)$
$!P$	F $_r(P)$	B $_r(P)$
$P\|Q$	F $_r(P) \cup$ F $_r(Q)$	B $_r(P) \cup$ B $_r(Q)$
$(\nu a : R)P$	F $_r(P) \setminus \{a^r\}$	B $_r(P) \cup \{a^r\}$
$[m = n]P$	$\{m, n\} \cup$ F $_r(P)$	B $_r(P)$

Table 2: Free and Bound Channels

ferent from the free ones; by using alpha-conversion, this requirement can be always satisfied.

Finally, observe that user names cannot be restricted, since the creation of a new user is a sensitive operation: it has to be performed only by the system administrator, as it may affect the RBAC policy underlying the entire system.

Notation. In this paper, we use '$_$' as a generic placeholder; thus, we denote with $\widetilde{_}$ a (possibly empty) tuple of entities of kind $_$. Moreover, we write $\widetilde{a^r} : \widetilde{R}$ to denote the tuple $a_1^r : R_1, \ldots, a_k^r : R_k$, for $k \geq 0$. Sometimes, we shall use tuples as sets (i.e., without considering the order of their elements) and we write, e.g., $b^s \in \widetilde{a^r}$ or $b^s : S \in \widetilde{a^r} : \widetilde{R}$. We use $\Pi_{i=1}^k P_i$ as a shorthand for $P_1 \mid \cdots \mid P_k$. Finally, as usual, we will omit trailing inactive processes.

1.2 Dynamic Semantics

The dynamics of the calculus is given in the form of a *reduction relation*. As customary, the reduction semantics is based on an auxiliary relation called *structural congruence* which allows to freely re-arrange systems in order to make reduction rules applicable. The key feature of the structural congruence is to equate terms that describe the same system; indeed, the syntax of the calculus provides a way to describe system behaviours, and the same behaviour can be described in different ways. For example, $A \parallel B$ describes a system of two parallel components that coincides with the system described by $B \parallel A$. The reason to split reductions and structural rules is to reserve reductions for actual computations, i.e., where the system actually performs some action, and keep them free of spurious term manipulation artifacts. In this way, reductions reflect at a glance the foundational building blocks of the computation, at the chosen abstraction level.

Axioms for Structural Congruence:

(S-A) $A \equiv B \qquad \text{if } A =_\alpha B$	(S-I) $A \parallel \mathbf{0} \equiv A$
(S-PC) $A \parallel B \equiv B \parallel A$	(S-A) $(A \parallel B) \parallel C \equiv A \parallel (B \parallel C)$
(S-E) $r\{\!\!\{ [n = n]P \}\!\!\}_\rho \equiv r\{\!\!\{ P \}\!\!\}_\rho$	(S-L C) $r\{\!\!\{ (\nu a : R)P \}\!\!\}_\rho \equiv (\nu a^r : R) r\{\!\!\{ P \}\!\!\}_\rho \qquad \text{if } a \neq r$
(S-R) $r\{\!\!\{ !P \}\!\!\}_\rho \equiv r\{\!\!\{ P \mid !P \}\!\!\}_\rho$	(S-RC) $(\nu a^r : R)(\nu b^s : S)A \equiv (\nu b^s : S)(\nu a^r : R)A$
(S-S) $r\{\!\!\{ P \mid Q \}\!\!\}_\rho \equiv r\{\!\!\{ P \}\!\!\}_\rho \parallel r\{\!\!\{ Q \}\!\!\}_\rho$	(S-E) $(\nu a^r : R)A \parallel B \equiv (\nu a^r : R)(A \parallel B) \qquad$ if $a^r \notin$ F (B)

Rules for Reduction Relation:

(R-R) $r\{\!\!\{ \mathbf{role}\, R.P \}\!\!\}_\rho \longmapsto r\{\!\!\{ P \}\!\!\}_{\rho \cup \{R\}}$	(R-C) $r\{\!\!\{ a(x).P \}\!\!\}_\rho \parallel s\{\!\!\{ a^r\langle n \rangle.Q \}\!\!\}_{\rho'} \longmapsto r\{\!\!\{ P[^n\!/x] \}\!\!\}_\rho \parallel s\{\!\!\{ Q \}\!\!\}_{\rho'}$
(R-Y) $r\{\!\!\{ \mathbf{yield}\, R.P \}\!\!\}_\rho \longmapsto r\{\!\!\{ P \}\!\!\}_{\rho \setminus \{R\}}$	(R-P) $\dfrac{A \longmapsto A'}{A \parallel B \longmapsto A' \parallel B}$
(R-R) $\dfrac{A \longmapsto A'}{(\nu a^r : R)A \longmapsto (\nu a^r : R)A'}$	(R-S) $\dfrac{A \equiv A' \quad A' \longmapsto B' \quad B' \equiv B}{A \longmapsto B}$

Table 3: Dynamic Semantics of the Calculus

The structural congruence relation, $\equiv$, is the least congruence on systems that is closed under the rules of the upper part of Table 3. Rule (S-A) equates alpha-convertible systems. Rules (S-I), (S-PC) and (S-A) state that '$\parallel$' is a commutative monoidal operator, with '**0**' as identity. Rules (S-L C), (S-RC) and (S-E) regulate the scope of restricted names. In particular, (S-L C) can be used to turn a restriction of a name inside a user into a restriction over the corresponding channel at the system level; (S-RC) allows to swap restrictions; (S-E) allows to extend the scope of a bound channel to include further parallel components, provided that this does not cause any name capture. Rule (S-E) states that a satisfied equality test does not affect the behaviour of the continuation process. Rule (S-R) allows to freely fold/unfold a replicated process. Finally, rule (S-S) states that a session of user r with rôles ρ hosting two parallel processes P and Q denotes the same system as two parallel sessions of r with rôles ρ hosting P and Q in isolation. Indeed, the key issues of a session

are the user owning it and the activated rôles. Rewriting one in the other represents no system computation, but only a different way of describing the same system, exactly like $A \parallel B$ and $B \parallel A$.

The reduction relation is defined by the axioms and rules in the lower part of Table 3. Rule (R-R) adds R to the rôles ρ activated in the current session, while (R-Y) removes R from ρ. Here and in what follows, '$\cup$' and '$\setminus$' denote the usual union and difference operations between sets; in particular, $\rho \setminus \{R\}$ is defined even if $R \notin \rho$. Rule (R-C) regulates the inter-process communication. It states that, whenever a process in s sends a message n along channel a^r and a process in r is waiting for a message on such a channel, an interaction occurs; as a result, n replaces each occurrence of the input variable x in the process P prefixed by the input action. Finally, rules (R-P) and (R-R) state that reductions are preserved by system contexts; rule (R-S) states that structurally equivalent systems have the same reductions.

Notice that, by exploiting rules (S-S), (R-R) and (R-Y), the user session $r\{\!|\, \mathbf{role}\, R.P \mid \mathbf{yield}\, S.Q \,|\!\}_\rho$ may evolve into $r\{\!|\, P \,|\!\}_{\rho\cup\{R\}} \parallel r\{\!|\, Q \,|\!\}_{\rho\setminus\{S\}}$, i.e., actions **role**/**yield** only affect the process thread executing them. Moreover, a single session with rôle R activated can later split in two distinct sessions; thus, a single rôle activation can be "closed" by several rôle deactivations.

1.3 RBAC Schema

So far, we discussed the way in which RBAC sessions can be modelled in our calculus. We now present a way to model in our framework the remaining features of the core RBAC96 model. To this aim, we need to define the *RBAC schema*, i.e., the rôles-to-users and permissions-to-rôles associations, where permissions enable the actions a user can perform within a system (in our framework, input and output actions only).

Managing rôles and their interrelationships is a difficult and sensitive task that is often centralised and delegated to a small team of security administrators. Traditionally, the RBAC schema consists of a pair of partial functions with finite domains $(\mathcal{U}_u\,;\mathcal{P})$, where $\mathcal{U}_u$ assigns rôles to users and $\mathcal{P}$ assigns permissions to rôles. Formally,

$$\mathcal{U}_u \;:\; \mathcal{N} \rightharpoonup_{\text{fin}} 2^{\mathcal{R}} \qquad\qquad \mathcal{P} \;:\; \mathcal{R} \rightharpoonup_{\text{fin}} 2^{\mathcal{A}}$$

where $\mathcal{A} \triangleq \mathcal{R} \times \{!, ?\}$ represents the set of performable actions. For notational convenience, we write the pairs $(R, !)$ and $(R, ?)$ as $R^!$ and $R^?$, respectively. Intuitively, permissions $R^?$ and $R^!$ determine the possibility of performing input and output actions over a channel of rôle R, respectively. Thus, input/output permissions are not defined in terms of channels, but of rôles. To this aim, we assume a partial function with finite domain, $\mathcal{U}_c$, assigning a rôle to a channel[1], i.e.

$$\mathcal{U}_c \;:\; \mathcal{N} \times \mathcal{N} \rightharpoonup_{\text{fin}} \mathcal{R}$$

In this way, we are flexible enough to model both the permission to communicate over a single channel (when the relation $\mathcal{U}_c$ maps only one channel to a rôle), and

[1] Since located channels can be considered as functionalities provided by users, it seems reasonable that each channel is associated with only one rôle.

the permission to communicate over the member of a group of channels (when the relation $\mathcal{U}_c$ maps more than one channel to the same rôle). This case may be useful in situations where more channels can handle the same kind of requests (cf. Example 1 for a possible situation). Observe that in $\mathcal{A}$ no permission represents actions **role** and **yield**. Indeed, we assume that a rôle can be activated if (and only if) it is assigned by $\mathcal{U}_u$ to the user willing to perform the action; a rôle can be deactivated if (and only if) it has been activated before.

For notational convenience, we merge $\mathcal{U}_u$ and $\mathcal{U}_c$ together and denote with $\mathcal{U}$ their union. Moreover, we also assume that the rôles associated with channels (grouped together in $\mathcal{R}_c$) are not included in the domain of $\mathcal{P}$, i.e., $\mathcal{P}$ matters only for rôles assigned to users (grouped together in $\mathcal{R}_u$). Clearly, $\mathcal{R}_c$ and $\mathcal{R}_u$ are assumed to be disjoint. To sum up, in our framework RBAC schemata are defined as follows.

Definition 1.1 (RBAC Schema). An *RBAC schema* $\mathcal{S}$ is a pair of partial functions with finite domains $(\mathcal{U}; \mathcal{P})$ such that

- $\mathcal{U} : (\mathcal{N} \cup \mathcal{N}\times\mathcal{N}) \rightharpoonup_{\text{fin}} (2^{\mathcal{R}} \cup \mathcal{R})$ such that, for any r and a^s in the domain of $\mathcal{U}$, it holds that $\mathcal{U}(r) \in 2^{\mathcal{R}}$ and $\mathcal{U}(a^s) \in \mathcal{R}$;
- $\mathcal{P} : \mathcal{R}_u \rightharpoonup_{\text{fin}} 2^{\mathcal{A}}$, where $\mathcal{A} \triangleq \mathcal{R}_c \times \{!, ?\}$, $\mathcal{R}_u \cup \mathcal{R}_c \subseteq \mathcal{R}$ and $\mathcal{R}_u \cap \mathcal{R}_c = \emptyset$.

To conclude the presentation of our language, we now give an example using the features introduced so far.

Example 1. Let us now formalise a scenario where a bank client is waiting to be served by one of the branch cashiers available. There are two users, r and s, representing respectively the client and the bank branch, while cashiers are modelled as channels named $c_1, \dots, c_n$ located at user s. The rôles available are `client` and `cashier`. The relation $\mathcal{U}$ assigns rôle `client` to user r and `cashier` to channels c_i, while $\mathcal{P}$ assigns to `client` the permission to communicate with any of the cashiers, i.e., $\mathtt{cashier}^! \in \mathcal{P}(\mathtt{client})$. In this way, r can indistinctly interact with any of the cashier available. The overall system can be described as follows:

$$
\begin{array}{l}
r\{\!|\, \mathbf{role}\ \mathtt{client}.\mathit{signal}^s\langle r\rangle.\mathit{served}(z).z\langle \mathit{req}_1\rangle.\cdots.z\langle \mathit{req}_k\rangle.z\langle \mathit{stop}\rangle.\mathbf{yield}\ \mathtt{client}\,|\!\}_{\emptyset} \quad \| \\
s\{\!|\, (\nu \mathit{free} : \mathtt{scheduling})(!\mathit{signal}(x).\mathit{free}(y).\mathit{served}^x\langle y\rangle \quad | \quad \Pi_{i=1}^{n} \mathit{free}^s\langle c_i^s\rangle \quad | \\
\qquad \Pi_{i=1}^{n}\ !c_i(x).(\ [x = \mathit{withdrw_req}] \ \text{<handle withdraw request>} \\
\qquad\qquad\qquad |\ [x = \mathit{dep_req}] \ \text{<handle deposit request>} \\
\qquad\qquad\qquad |\ [x = \mathit{stop}]\mathit{free}^s\langle c_i^s\rangle)\ )\,|\!\}_{\rho'}
\end{array}
$$

Once the client enters the bank (i.e., he activates rôle `client`), he signals his presence to the bank and waits to be served. When one of the cashiers becomes available (information maintained internally by the bank via the reserved channel *free* used for cashiers' scheduling), the client is notified and can make requests along the received channel z, referring to the available cashier. Then, cashiers repeatedly receive and handle requests. For simplicity, we only assume functionalities to handle money withdraw and deposit. Moreover, we do not consider the order in which clients arrive; a system of queues can however be added routinely [24]. ◇

2 Static Semantics

We now describe a type system that provides us with static guarantees that the set of actions performed by any user during the computation respects the RBAC schema. Moreover, as already discussed when presenting the syntax of the calculus, it is also used to reject ill-formed terms.

The syntax of types is defined by the following productions:

$$\begin{array}{llll} \textit{Value Types} & T & ::= & \rho[\widetilde{a} : \widetilde{C}] \mid C \\ \textit{Channel Types} & C & ::= & R(T) \end{array}$$

Type $\rho[a_1 : R_1(T_1), \dots, a_n : R_n(T_n)]$ can be assigned to a user r belonging to rôles in ρ and owning channels $\widetilde{a^r}$ ordinately of type $\widetilde{R(T)}$. Type $R(T)$ can be assigned to channels of rôle R along which values of type T can be exchanged. Notice that the base case of the recursive definition of types is when the set $\widetilde{a} : \widetilde{C}$ in a type $\rho[\widetilde{a} : \widetilde{C}]$ is empty.

Notation. Here and in the rest of the paper, $\mathcal{P}(\rho)$ denotes the set $\bigcup_{R \in \rho} \mathcal{P}(R)$. Moreover, we denote with $\uplus$ the union of partial functions with disjoint domains.

A *typing environment* Γ is a partial mapping with finite domain from $\mathcal{N}$ into types and it can be extended as follows:

$$\Gamma, x : T \triangleq \Gamma \uplus \{x : T\}$$

$$\Gamma, a^r : C \triangleq \Gamma', \text{ for } \Gamma'(s) = \begin{cases} \Gamma(s) & \text{if } s \neq r \\ \rho[a : C, \widetilde{b} : \widetilde{C}] & \text{if } s = r \text{ and } \Gamma(r) = \rho[\widetilde{b} : \widetilde{C}] \text{ and } a \notin \widetilde{b} \end{cases}$$

Remarkably, the extension of a typing environment could be undefined (e.g., if $x \in dom(\Gamma)$ in the first case or if $a \in \widetilde{b}$ in the second case). A typing environment Γ can be used to type a system under a schema $(\mathcal{U}; \mathcal{P})$ only if the rôle information in Γ 'respects' the associations in $\mathcal{U}$. This intuition is formalised by the following definition.

Definition 2.1. Given a RBAC schema $(\mathcal{U}; \mathcal{P})$ and a typing environment Γ, we say that Γ *respects* $\mathcal{U}$ if, for all $r \in dom(\Gamma)$, it holds that $\Gamma(r) = \rho[a_1 : R_1(T_1), \dots, a_n : R_n(T_n)]$ with $\mathcal{U}(r) = \rho$ and $\mathcal{U}(a_i^r) = R_i$, for all $i = 1, \dots, n$.

The primary judgements of the type system are of the form $\Gamma \vdash^{\mathcal{S}} A$. Such a judgement states that A is well-formed with respect to Γ and $\mathcal{S}$; this implies that A respects the RBAC schema $\mathcal{S}$. To infer the main judgement, we rely on two auxiliary judgements, one for values and one for processes. Judgement $\Gamma \vdash n : T$ states that the value n has type T in Γ; judgement $\Gamma; \rho \vdash_r^{\mathcal{S}} P$ states that P respects Γ and $\mathcal{S}$ when it is run in a session of r with rôles ρ activated.

The typing rules are collected in Table 4. Most of them are self-explanatory; we only comment below the most significant ones, i.e. those related to the actions in our calculus. The idea beyond these rules is that an action can be executed only if the current session has activated a rôle enabling the action. Rule (T-I) states that, for typing $a(x).P$ in a session of r where rôles ρ are activated, we need to establish that a^r has type $R(T)$ in Γ, that inputs over a channel of group R can be performed when playing rôles ρ and that P is typeable once assumed that x has type T. Rule (T-O)

Typing Values:

(T-I $_1$)
$$\frac{\Gamma(r) = \rho[\widetilde{a} : \widetilde{C}]}{\Gamma \vdash r : \rho[\widetilde{a} : \widetilde{C}]}$$

(T-I $_2$)
$$\frac{\Gamma(r) = \rho[\widetilde{b} : \widetilde{C}, a : C, \widetilde{b'} : \widetilde{C'}]}{\Gamma \vdash a^r : C}$$

Typing Processes:

(T-N)
$$\frac{}{\Gamma; \rho \vdash_r^{\mathcal{S}} \mathbf{nil}}$$

(T-I)
$$\frac{\Gamma \vdash a^r : R(T) \quad R^? \in \mathcal{P}(\rho) \quad \Gamma, x : T; \rho \vdash_r^{(\mathcal{U};\mathcal{P})} P}{\Gamma; \rho \vdash_r^{(\mathcal{U};\mathcal{P})} a(x).P}$$

(T-P)
$$\frac{\Gamma; \rho \vdash_r^{\mathcal{S}} P \quad \Gamma; \rho \vdash_r^{\mathcal{S}} Q}{\Gamma; \rho \vdash_r^{\mathcal{S}} P \mid Q}$$

(T-O)
$$\frac{\Gamma \vdash m : R(T) \quad \Gamma \vdash n : T \quad R^! \in \mathcal{P}(\rho) \quad \Gamma; \rho \vdash_r^{(\mathcal{U};\mathcal{P})} P}{\Gamma; \rho \vdash_r^{(\mathcal{U};\mathcal{P})} m\langle n \rangle.P}$$

(T-R)
$$\frac{\Gamma; \rho \vdash_r^{\mathcal{S}} P}{\Gamma; \rho \vdash_r^{\mathcal{S}} !P}$$

(T-R^)
$$\frac{\Gamma \vdash r : \rho'[\widetilde{a} : \widetilde{C}] \quad R \in \rho' \quad \Gamma; \rho \cup \{R\} \vdash_r^{\mathcal{S}} P}{\Gamma; \rho \vdash_r^{\mathcal{S}} \mathbf{role}\, R.P}$$

(T-R)
$$\frac{\Gamma, a^r : R(T); \rho \vdash_r^{\mathcal{S}} P}{\Gamma; \rho \vdash_r^{\mathcal{S}} (\nu a : R)P}$$

(T-Y)
$$\frac{R \in \rho \qquad \Gamma; \rho \setminus \{R\} \vdash_r^{\mathcal{S}} P}{\Gamma; \rho \vdash_r^{\mathcal{S}} \mathbf{yield}\, R.P}$$

(T-M)
$$\frac{\Gamma; \rho \vdash_r^{\mathcal{S}} P}{\Gamma; \rho \vdash_r^{\mathcal{S}} [m = n]P}$$

Typing Systems:

(T-E)
$$\frac{}{\Gamma \vdash^{\mathcal{S}} \mathbf{0}}$$

(T-S)
$$\frac{\Gamma \vdash r : \rho'[\widetilde{a} : \widetilde{C}] \quad \rho \subseteq \rho' \quad \Gamma; \rho \vdash_r^{\mathcal{S}} P}{\Gamma \vdash^{\mathcal{S}} r\{\!\!\{ P \}\!\!\}_\rho}$$

(T-S P)
$$\frac{\Gamma \vdash^{\mathcal{S}} A \qquad \Gamma \vdash^{\mathcal{S}} B}{\Gamma \vdash^{\mathcal{S}} A \parallel B}$$

(T-S R)
$$\frac{\Gamma, a^r : R(T) \vdash^{\mathcal{S}} A}{\Gamma \vdash^{\mathcal{S}} (\nu a^r : R)A}$$

Table 4: Typing Rules

is similar: it checks that an output over a channel of group R is allowed when the rôles in ρ are activated; moreover, it also requires that the transmitted value n is assigned type T in Γ. Rule (T-R^) states that, for typing process **role** $R.P$ in a session of r where rôles ρ are activated, we need to check that r can assume rôle R and that P is typeable for r having activated $\rho \cup \{R\}$. Rule (T-Y) states that process **yield** $R.P$ is legal for r only when R has been previously activated and if P is typeable for r when R is off.

Finally, notice that in rules (T-R) and (T-S R) the type of the restricted channel is not tracked in the restriction construct. Indeed, for type checking purposes, it suffices to ensure that the new channel is used coherently by all the processes accessing it. To this aim, we only need to invent a suitable T when applying the rules and verify that all the accesses to the channel conform to T.

Definition 2.2 (Well-typedness). Given a RBAC schema $\mathcal{S} = (\mathcal{U}; \mathcal{P})$ and a system A, we say that A is *well-typed in* $\mathcal{S}$ if there exists a typing environment Γ respecting $\mathcal{U}$ such that $\Gamma \vdash^{\mathcal{S}} A$.

Example 2. Let us consider the banking scenario described in Example 1. To illustrate the type system introduced above, let us give a suitable typing for the system. Let

$$T_{csh} \triangleq \mathtt{cashier}(\{\mathtt{request}\}[])$$

be the type of the cashiers, i.e., channels belonging to rôle `cashier` and exchanging values of type $\{\mathtt{request}\}[]$. Type $\{\mathtt{request}\}[]$ represents the possible requests of clients; values of this type are names belonging to rôle `request` which do not provide any channel. Moreover, we let

$$T_{cl} \triangleq \{\mathtt{client}\}[served : \mathtt{cashier_get}(T_{csh})]$$

be the type of user r. This represents users belonging to rôle `client` and owning a channel named *served* of type $\mathtt{cashier_get}(T_{csh})$. Then, a suitable typing environment Γ is

$$\begin{array}{rcl} r & \mapsto & T_{cl} \\ s & \mapsto & \rho'[signal : \mathtt{cashier_req}(T_{cl}), c_1 : T_{csh}, \ldots, c_n : T_{csh}] \\ withdrw_req & \mapsto & \{\mathtt{request}\}[] \\ dep_req & \mapsto & \{\mathtt{request}\}[] \\ stop & \mapsto & \{\mathtt{request}\}[]. \end{array}$$

The system of Example 1 is then well-typed in any schema $(\mathcal{U}, \mathcal{P})$ such that Γ respects $\mathcal{U}$ and $\mathcal{P}$ is such that

$$\begin{array}{rcl} \{\mathtt{cashier_req}^!, \mathtt{cashier_get}^?, \mathtt{cashier}^!\} & \subseteq & \mathcal{P}(\mathtt{client}) \\ A \cup \{\mathtt{cashier_req}^?, \mathtt{cashier_get}^!, \mathtt{cashier}^?, & & \\ \mathtt{scheduling}^?, \mathtt{scheduling}^!\} & \subseteq & \mathcal{P}(\rho') \end{array}$$

where $A \subseteq \mathcal{A}$ is a set of action permissions that allow the handling of client's requests. ⋄

Example 3. In the real world, it is unrealistic to allow any bank client to ask for any kind of bank operation. For instance, when a client applies for a credit card, he is always asked for some credentials. To model this finer scenario, we let each available operation to be modelled as a specific process, which can be activated through

a specific channel (e.g., channel *wdrw* handles withdraw requests and activates process P_{wdrw}, *opn* handles open account requests, *cc* handles credit card requests). The communication along different channels requires different rôles and, thus, it is a way to control the credentials of the client. In this setting, the cashier c_i of Example 1 is implemented by the following process (the remaining behaviour of the bank is implemented as in Example 1):

$$c_i(x).(\ [x = withdrw_req]\, wdrw(y).P_{wdrw} \quad | \quad [x = open_req]\, opn(y).P_{opn} \quad | \\ [x = creditcard_req]\, cc(y).P_{cc} \quad | \quad [x = stop]\, free^s\langle c_i^s\rangle\ )$$

Let $\mathcal{U}$ assign to channel *wdrw* (resp., *opn* and *cc*) the group $\mathtt{wdrw}$ (resp., $\mathtt{opn}$ and $\mathtt{cc}$), and let $\mathcal{P}$ be such that $\mathtt{cc}^! \in \mathcal{P}(\mathtt{rich_client})$, $\mathtt{wdrw}^! \in \mathcal{P}(\mathtt{client})$ and $\mathtt{opn}^! \in \mathcal{P}(\mathtt{user})$. Under this schema, the following client is not well-typed, as he has not activated the correct rôle to perform credit card requests:

$$r\{\!|\ \textbf{role}\ \mathtt{client}.signal^s\langle r\rangle.served(z).z\langle creditcard_req\rangle.cc^s\langle signature\rangle.z\langle stop\rangle\ |\!\}_\emptyset$$

Indeed, the type checking fails when applying the rule (T-O) to action $cc^s\langle signature\rangle$ because $\mathtt{cc}^! \notin \mathcal{P}(\{\mathtt{client}\})$. On the contrary, assuming that $\mathtt{rich_client} \in \mathcal{U}(r_1)$, $\mathtt{client} \in \mathcal{U}(r_2)$ and $\mathtt{user} \in \mathcal{U}(r_3)$, and that $\{\mathtt{cashier_req}^!, \mathtt{cashier_get}^?, \mathtt{cashier}^!\} \subseteq \mathcal{P}(\mathtt{user}) \cap \mathcal{P}(\mathtt{client}) \cap \mathcal{P}(\mathtt{rich_client})$, the following clients are well-typed:

$$r_1\{\!|\ \textbf{role}\ \mathtt{rich_client}.signal^s\langle r\rangle.served(z).z\langle creditcard_req\rangle.cc^s\langle signature\rangle.z\langle stop\rangle\ |\!\}_\emptyset \\ r_2\{\!|\ \textbf{role}\ \mathtt{client}.signal^s\langle r\rangle.served(z).z\langle withdrw_req\rangle.wdrw^s\langle sum\rangle.z\langle stop\rangle\ |\!\}_\emptyset \\ r_3\{\!|\ \textbf{role}\ \mathtt{user}.signal^s\langle r\rangle.served(z).z\langle open_req\rangle.opn^s\langle personal_data\rangle.z\langle stop\rangle\ |\!\}_\emptyset$$

Indeed, actions $cc^s\langle signature\rangle$, $wdrw^s\langle sum\rangle$ and $opn^s\langle personal_data\rangle$ are all enabled by the rôles previously activated by users, viz. $\mathtt{rich_client}$, $\mathtt{client}$ and $\mathtt{user}$, respectively. $\diamond$

We now establish the soundness of our type system in the standard way, i.e. by proving subject reduction and type safety. The first result states that well-typedness is preserved along reductions; the second result ensures that only systems abiding by the RBAC schema are allowed (i.e., users only perform actions permitted by their duly activated rôles). The proofs are in Appendix A.1.

Theorem 2.1 (Subject Reduction). *If* $\Gamma \vdash^{\mathcal{S}} A$ *and* $A \longmapsto A'$, *then* $\Gamma \vdash^{\mathcal{S}} A'$.

To state type safety, we first need to formally define what situations our type system wants to avoid. Thus, in Table 5, we introduce the notion of *run-time errors* and prove that they never arise in any well-typed system. Intuitively, run-time errors are generated in three possible ways: whenever a session is equipped with rôles not assigned to the user owning that session (see law (E-S)); whenever a rôle is activated (resp., deactivated) by a user (resp., by a session) not owning such a rôle (see laws (E-Rˆ) and (E-Y), resp.); whenever an input/output action is performed in a user session where no privilege enabling such an action is provided by the rôles active in that session (see laws (E-I) and (E-O)).

(E-S) $\dfrac{\rho \not\subseteq \mathcal{U}(r)}{r\{\!\mid P \mid\!\}_\rho \rightsquigarrow_{(\mathcal{U};\mathcal{P})}}$	(E-Rˆ) $\dfrac{R \notin \mathcal{U}(r)}{r\{\!\mid \textbf{role}\, R.P \mid\!\}_\rho \rightsquigarrow_{(\mathcal{U};\mathcal{P})}}$	(E-Y) $\dfrac{R \notin \rho}{r\{\!\mid \textbf{yield}\, R.P \mid\!\}_\rho \rightsquigarrow_{\mathcal{S}}}$
(E-I) $\dfrac{\mathcal{U}(b^r) = S \qquad S^{\,?} \notin \mathcal{P}(\rho)}{r\{\!\mid b(x).P \mid\!\}_\rho \rightsquigarrow_{(\mathcal{U};\mathcal{P})}}$	(E-O) $\dfrac{\mathcal{U}(b^s) = S \qquad S^{\,!} \notin \mathcal{P}(\rho)}{r\{\!\mid b^s\langle n\rangle.P \mid\!\}_\rho \rightsquigarrow_{(\mathcal{U};\mathcal{P})}}$	
(E-P) $\dfrac{A \rightsquigarrow_{\mathcal{S}}}{A \parallel B \rightsquigarrow_{\mathcal{S}}}$	(E-R) $\dfrac{A \rightsquigarrow_{(\mathcal{U}\uplus\{a^r:R\};\mathcal{P})}}{(\nu a^r\!:\!R)A \rightsquigarrow_{(\mathcal{U};\mathcal{P})}}$	(E-S) $\dfrac{A \equiv B \quad B \rightsquigarrow_{\mathcal{S}}}{A \rightsquigarrow_{\mathcal{S}}}$

Table 5: Run-time Errors

Theorem 2.2 (Type Safety). *If A is well-typed in $\mathcal{S}$, then $A \rightsquigarrow_{\mathcal{S}}$ cannot hold.*

We conclude this section remarking that our type system is not powerful enough to type all legal systems. For example, the system $r\{\!\mid a^r\langle r\rangle \mid\!\}_\rho$ is untypeable. Similarly, we have no notion of subtyping; thus, a channel must always carry values exactly of the same type. We now sketch how these deficiencies could be remedied, by following standard techniques; full details are omitted, as they are completely well understood and orthogonal w.r.t. the new ideas of our work.

Recursive Types In order to type $r\{\!\mid a^r\langle r\rangle \mid\!\}_\rho$, we would need a typing environment Γ assigning to r a type T such that $T = \rho'[a^r : R(T)]$, for some $\rho' \supseteq \rho$ and R. Clearly, such a type T is not expressible with our type syntax, as it would require an infinite nesting of type constructors, i.e. $\rho'[\ a^r : R(\rho'[a^r : R(\cdots)])\ ]$. By following [8, 20, 24], this problem can be solved by using equations between type expressions whose solutions are infinite types, like T above. To this aim, we assume a set of type variables Ξ, ranged over by ξ, and extend the syntax of value types as follows:

$$T \ ::= \ \rho[\widetilde{a} : \widetilde{C}] \ \mid \ C \ \mid \ \xi \ \mid \ \mu\xi.T$$

Intuitively, $\mu\xi.T$ stands for the solution of the (recursive) equation $\xi = T$. However, to avoid nonsensical expressions like $\mu\xi.\xi$, we impose the constraint that in $\mu\xi.T$ the variable ξ occurs *guarded* in T, i.e. it occurs underneath at least one of the other type constructors. Moreover, we only consider *closed* type expressions, i.e. expressions in which each occurrence of a type variable ξ is underneath a $\mu\xi._$ construct. By exploiting these two assumptions, it can be standardly proved that the set of types is a c.p.o. and, thus, the solution of an equation $\mu\xi.T$ is unique and can be obtained with a least fixed-point construction.

If we now represent (the unfolding of) a (possibly recursive) type as its (possibly infinite) parse tree, we can deem two types *equivalent*, written $T_1 \asymp T_2$, if and only if they represent the same tree. A lot of literature on type systems is devoted to compute

$\asymp$ algorithmically (see, e.g., [2, 20]); for our purposes, it suffices to remember that $\asymp$ is a congruence on types such that $\mu\xi.T \asymp T[^{\mu\xi.T}/_{\xi}]$.

If we now add the rule

$$\text{(T-I -R)}\ \frac{\Gamma \vdash n : T \qquad T \asymp T'}{\Gamma \vdash n : T'}$$

to the type system of Table 4 and modify rule (T-I $_2$) to be

$$\frac{\Gamma(r) \asymp \rho[\widetilde{b} : \widetilde{C}, a : C, \widetilde{b'} : \widetilde{C'}]}{\Gamma \vdash a^r : C}$$

it is easy to see that $\Gamma \vdash^{(\mathcal{U};\mathcal{P})} r\{\!|\, a^r\langle r\rangle \,|\!\}_\rho$, whenever $\Gamma(r) \triangleq \mu\xi.\rho[a^r : R(\xi)]$ and $R^! \in \mathcal{P}(\rho)$.

Subtyping Subtyping is a preorder on types that can be thought of as inclusion between the set of the values of the types. If T' is subtype of T, then a value of type T' is also of type T; thus, any expression of type T' can always replace an expression of type T, without compromising well-typedness. In traditional programming languages, this feature is used to reduce the size of a program, as the same function can be invoked on parameters of different types, without writing a 'copy' of the same function for each subtype. For example, assume, as usual, that `int` is a subtype of `real` and that there is a function to multiply two reals. Then, the same function can also be used to multiply two integers.

In our setting, we can define an ordering on types to allow the passage of values of different (yet somehow related) types along the same channel. We sketch here a very basic form of subtyping inspired by [14]; for more elaborated settings see, e.g., [21, 24]. First, we need to define the *subtyping relation*, $\sqsubseteq$, that is the least preorder on types satisfying the following rules:

$$\frac{\rho \subseteq \rho' \qquad h \leq k \qquad \forall\, i = 1, ..., h \,.\, C_i \sqsubseteq C'_i}{\rho[a_1 : C_1, \ldots, a_h : C_h] \;\sqsubseteq\; \rho'[a_1 : C'_1, \ldots, a_k : C'_k]} \qquad\qquad \frac{R = R' \qquad T \sqsubseteq T'}{R(T) \;\sqsubseteq\; R'(T')}$$

Then, we need to update rule (T-O) to become

$$\frac{\Gamma \vdash m : R(T) \quad \Gamma \vdash n : T' \quad T \sqsubseteq T' \quad R^! \in \mathcal{P}(\rho) \quad \Gamma; \rho \vdash_r^{(\mathcal{U};\mathcal{P})} P}{\Gamma; \rho \vdash_r^{(\mathcal{U};\mathcal{P})} m\langle n\rangle.P}$$

In this framework, we can type the system

$$r\{\!|\, a(x).b^x\langle n\rangle \,|\!\}_\rho \;\;\|\;\; s\{\!|\, a^r\langle r_1\rangle \,|\!\}_{\rho_1} \;\;\|\;\; t\{\!|\, a^r\langle r_2\rangle \,|\!\}_{\rho_2} \;\;\|\;\; r_1\{\!|\, b(y).c(z) \,|\!\}_{\rho'} \;\;\|\;\; r_2\{\!|\, b(y).c'(z) \,|\!\}_{\rho'}$$

Indeed, it suffices to take Γ such that $\Gamma(n) = T$, $\Gamma(r_1) = \rho'[b : R(T), c : C]$, $\Gamma(r_2) = \rho'[b : R(T), c' : C']$ and $\Gamma(r) = \rho[\, a \,:\, S\,(\rho'[b : R(T)])\,]$, for some R and S.

3 Behavioural Semantics

One of the main advantages of process calculi is the possibility of developing over them behavioural equivalences, that abstract a term from its syntax and concentrate

on its functionalities. To this aim, we consider a standardly defined typed behavioural congruence, viz. *reduction barbed congruence* [15]. This is a touchstone equivalence defined in terms of the reduction relation and of a notion of observation, and then closed under all possible system contexts. The reason to consider a typed congruence is that only well-typed contexts guarantee a reduction behaviour abiding by the RBAC policy.

In its typed version, barbed congruence is tagged with an environment Γ and RBAC schema $\mathcal{S}$, to signify that it equates terms that are typeable under Γ and $\mathcal{S}$. Moreover, only contexts typeable under Γ and $\mathcal{S}$ are considered in the definition of the congruence. Thus, following the style of [13], we write $\Gamma \models^{\mathcal{S}} A_1 \cong A_2$ to mean that $\Gamma \vdash^{\mathcal{S}} A_i$, for $i = 1, 2$, and that A_1 and A_2 exhibit the same behaviour in all environments 'compatible' with Γ and $\mathcal{S}$.

We now formally define barbed congruence. As usual, we denote with $\longmapsto^*$ the reflexive and transitive closure of the reduction relation $\longmapsto$.

Definition 3.1 (Barbs). The *observation predicate* $A \downarrow \eta$ holds if

- either $\eta = a^r$ and $A \equiv (\nu\, \widetilde{b^s} \colon \widetilde{R})(A' \parallel r\{\!|\, a(x).P \,|\!\}_\rho)$ for $a^r \notin \widetilde{b^s}$,
- or $\eta = \overline{a^r}$ and $A \equiv (\nu\, \widetilde{b^s} \colon \widetilde{R})(A' \parallel s'\{\!|\, a^r\langle n\rangle.P \,|\!\}_\rho)$ for $a^r \notin \widetilde{b^s}$.

The predicate $A \Downarrow \eta$ holds if there exists A' such that $A \longmapsto^* A'$ and $A' \downarrow \eta$.

We remark that the chosen barbs only express the ability to interact over channels. Indeed, observing rôle activations/deactivations is not reasonable, as no context can determine whether a user performs a **role/yield**: these operations only affect the thread performing them.

Definition 3.2 (Reduction Barbed Congruence). *Reduction barbed congruence* is the largest binary and symmetric typed relation over systems such that, whenever $\Gamma \models^{\mathcal{S}} A_1 \cong A_2$, it holds that

1. (*Barb preservation*) if $A_1 \downarrow \eta$, then $A_2 \Downarrow \eta$
2. (*Reduction closure*) if $A_1 \longmapsto A_1'$, then there exists a system A_2' such that $A_2 \longmapsto^* A_2'$ and $\Gamma \models^{\mathcal{S}} A_1' \cong A_2'$
3. (*Contextuality*) let $\mathcal{S}$ be $(\mathcal{U}; \mathcal{P})$; then,
   (a) for all $\mathcal{P}'$ and $\widetilde{n} \colon \widetilde{T}$ such that $\widetilde{n} \cap dom(\mathcal{U}) = \emptyset$, it holds that $\Gamma, \widetilde{n} \colon \widetilde{T} \models^{(\mathcal{U} \cup \widetilde{n}\colon \widetilde{T}; \mathcal{P} \cup \mathcal{P}')} A_1 \cong A_2$
   (b) for all systems B such that $\Gamma \vdash^{\mathcal{S}} B$, it holds that $\Gamma \models^{\mathcal{S}} A_1 \parallel B \cong A_2 \parallel B$
   (c) for all $a^r : R(T)$ such that $\Gamma = \Gamma', a^r : R(T)$ and $\mathcal{U} = \mathcal{U}' \uplus \{a^r : R\}$, it holds that $\Gamma' \models^{(\mathcal{U}'; \mathcal{P})} (\nu a^r \colon R)A_1 \cong (\nu a^r \colon R)A_2$.

The less intuitive condition of the above definition is contextuality. Essentially, it requires that the equated systems A_1 and A_2 must be equivalent in any execution context. An execution context can affect the behaviour of such systems in three ways: it can extend the RBAC schema, thus enabling more functionalities of A_1 and A_2; it can provide more parallel components that, by interacting with A_1 and A_2, could change their behaviours; it can hide channels and, hence, delete observable behaviours of A_1 and A_2. These aspects are handled by the sub-conditions (a), (b) and (c), respectively.

The problem with this definition of barbed congruence is that it must be proved by analysing all the system contexts, which makes it hardly tractable. In Appendix B, we provide a more tractable proof-technique for $\cong$. Since this task requires several technicalities taken from the field of process calculi, we leave it to the interested reader; the other readers should only know that tools for establishing barbed congruence in a simpler way do exist.

To conclude, we now list a few algebraic laws that illustrate some key features of our framework. In what follows, we fix an RBAC schema $\mathcal{S}$ and a suitable typing environment Γ. The first equation states that a terminated session of a user does not affect the evolution of a system. Indeed, it holds that

$$\Gamma \models^{\mathcal{S}} r\{\!\!\{\, \mathbf{nil} \,\}\!\!\}_{\rho} \cong \mathbf{0}.$$

This is different from some distributed calculi, like e.g. the Ambient calculus [7] or K [9], where the presence of a place for computations is relevant. Moreover, differently from several distributed languages, the user performing an *output* action is irrelevant; the only relevant aspect is the set of permissions activated when performing the action. This is summarised in the following law:

$$\Gamma \models^{\mathcal{S}} r\{\!\!\{\, b^{s}\langle n\rangle \,\}\!\!\}_{\rho} \cong t\{\!\!\{\, b^{s}\langle n\rangle \,\}\!\!\}_{\rho}.$$

A similar law holds for the **yield** action. Notice that only for these two actions the identity of the user performing them is irrelevant. For example, relocating an input action breaks the equivalence between processes, as input channels implicitly refer the user owning them. Indeed, we have that

$$\Gamma \models^{\mathcal{S}} r\{\!\!\{\, a(x).P \,\}\!\!\}_{\rho} \not\cong t\{\!\!\{\, a(x).P \,\}\!\!\}_{\rho}.$$

Similarly, it is possible to move a **role** R prefix between two users only when R is assigned to both of them.

By exploiting these observations, we develop in the following example a relocation procedure to establish whether a process can be moved from a user to another. This procedure can be exploited to reduce the number of users in a system, while maintaining the overall system behaviour.

Example 4. We now give simple procedure to infer judgements of the form $\Gamma \models^{\mathcal{S}} r\{\!\!\{\, P \,\}\!\!\}_{\rho} \cong s\{\!\!\{\, P \,\}\!\!\}_{\rho}$. This judgement states that process P can be indifferently executed by r and s without altering its observable functionalities. Thus, session $r\{\!\!\{\, P \,\}\!\!\}_{\rho}$ can be replaced by $s\{\!\!\{\, P \,\}\!\!\}_{\rho}$. If no other session of r is left in the system, then r itself has been removed.

The procedure is very simple. Try to infer both $\Gamma;\rho \vdash_{r}^{\mathcal{S}} P$ and $\Gamma;\rho \vdash_{s}^{\mathcal{S}} P$ without using rules (T-I) and (T-R). If you succeed, then $\Gamma \models^{\mathcal{S}} r\{\!\!\{\, P \,\}\!\!\}_{\rho} \cong s\{\!\!\{\, P \,\}\!\!\}_{\rho}$, otherwise $\Gamma \models^{\mathcal{S}} r\{\!\!\{\, P \,\}\!\!\}_{\rho} \not\cong s\{\!\!\{\, P \,\}\!\!\}_{\rho}$. We cannot use rule (T-I) because, as we have already discussed, we cannot relocate processes containing input prefixes. A similar problem arises also for restriction (thus, we cannot use rule (T-R)). Indeed, the interplay between user names, restricted channel names and restricted channels is subtle. For example, consider the system $P \triangleq (\nu a : R)a^{r}\langle a^{s}\rangle$, try to run it in users r, s

and t, and put the resulting session in an arbitrary system context. In the first case, P cannot be engaged in any reduction, as it emits a value on a channel known only by P itself; in the second case, P sends a bound value; in the third case, P sends a free value. Thus, relocating processes with restrictions breaks equivalences, in general.

It can now be easily proved that this procedure is a sound and complete proof-technique for judgement $\Gamma \models^{\mathcal{S}} r\{\!| P |\!\}_\rho \;\cong\; s\{\!| P |\!\}_\rho$, whenever P does not contain restrictions and input prefixes. ⋄

4 Adding Rôle Activations and Deactivations

In this section, we show how our framework can be adapted to encompass more advanced features. Usually, the task of properly putting **role/yield** operations within a system is tedious and error-prone; moreover, it assumes a full knowledge of the RBAC schema at programming time. We now describe a way to add rôle activations/deactivations within a system in such a way that the resulting system can be executed under any given schema $(\mathcal{U}; \mathcal{P})$, whenever possible, i.e., when users are allowed to activate the rôles required by the actions they are willing to perform.

A first technique rewrites a system A without actions **role/yield** by activating at the beginning of each session of a generic user r all the rôles in $\mathcal{U}(r)$. Intuitively, the refined system contains all the legal behaviours of A with respect to the RBAC schema given. However, the fact that all the rôles assigned to a user are always activated violates a basilar RBAC design principle: a rôle should be active only when needed.

A second naïve algorithm replaces each input/output prefix $\alpha.Q$ occurring in each session of a generic user r with $\mathbf{role}\,R.\alpha.\mathbf{yield}\,R.Q$, where rôle R belongs to $\mathcal{U}(r)$ and enables action α. The algorithm is very simple but it presents several drawbacks: it always adds a pair of auxiliary actions **role/yield** for every prefix α occurring in the process, although it could be that rôle R enables also the prefixes following in Q. Furthermore, when rewriting $\alpha.Q$, it is possible that several rôles enable α: in this case, a thorough choice of which rôle to activate may minimise the number of **role/yield** actions.

We now present an algorithm that adds a smaller number of actions **role/yield**. The algorithm works on a tree-representation of the process in each session of a generic user r: first, the tree is partitioned into subtrees by collecting together nodes (i.e., actions) which require the same rôle in order to be executed correctly; then a $\mathbf{role}\,R$ auxiliary action is added before the root of each subtree requiring rôle R, and dually a $\mathbf{yield}\,R$ action is added after the leaf of each such subtree.

More specifically, a process P running inside a session of user r is translated into an annotated binary tree, where nodes represent process operators and each node is annotated with the set of rôles whose permissions enable the action associated with the node (if the node is not associated with an input/output action, any rôle available for user r will enable it). The tree is expressed in terms of a tuple $t = (V, E, rt, \phi)$, where V is a finite set of *nodes*, $E \subseteq V \times V$ is the set of the *edges* (i.e., $(v, v') \in E$ iff there is an edge from v to v'), $rt \in V$ is the *root* of the tree, and $\phi : V \longrightarrow 2^{\mathcal{R}}$ is the assignment of rôles to nodes used to annotate each node. The annotated tree associated with a finite process P (without **role/yield** actions) running inside a session of user r may be

generated by the function $\mathrm{T}^{r}_{\Gamma;\mathcal{P}}(P)$ described below, with $(\mathcal{U};\mathcal{P})$ a RBAC schema and Γ a typing environment respecting $\mathcal{U}$. In the definition, we write $\Gamma\{r\}$ for the set ρ such that $\Gamma \vdash r : \rho[\widetilde{a} : \widetilde{C}]$; thus, $R \in \Gamma\{r\}$ is a shortcut for $\Gamma \vdash r : \rho[\widetilde{a} : \widetilde{C}] \ \wedge \ R \in \rho$.

$$
\begin{array}{rcl}
\mathrm{T}^{r}_{\Gamma;\mathcal{P}}(\mathbf{nil}) & \triangleq & (\{v\}, \emptyset, v, \{v \mapsto \Gamma\{r\}\}) \\
\mathrm{T}^{r}_{\Gamma;\mathcal{P}}(a(x).P) & \triangleq & (V \cup \{v\}, E \cup \{(v, rt)\}, v, \phi \cup \{v \mapsto \rho\}) \\
 & & \text{where } \mathrm{T}^{r}_{\Gamma,x:T;\mathcal{P}}(P) = (V, E, rt, \phi), v \notin V, \Gamma \vdash a^r : S(T) \\
 & & \text{and } \rho = \{R \in \Gamma\{r\} \ : \ S^{?} \in \mathcal{P}(R)\} \\
\mathrm{T}^{r}_{\Gamma;\mathcal{P}}(m\langle n\rangle.P) & \triangleq & (V \cup \{v\}, E \cup \{(v, rt)\}, v, \phi \cup \{v \mapsto \rho\}) \\
 & & \text{where } \mathrm{T}^{r}_{\Gamma;\mathcal{P}}(P) = (V, E, rt, \phi), v \notin V, \ \Gamma \vdash a^r : S(T), \\
 & & \Gamma \vdash n : T \text{ and } \rho = \{R \in \Gamma\{r\} \ : \ S^{!} \in \mathcal{P}(R)\} \\
\mathrm{T}^{r}_{\Gamma;\mathcal{P}}(P_1 \mid P_2) & \triangleq & (V_1 \cup V_2 \cup \{v\}, E_1 \cup E_2 \cup \{(v, rt_1), (v, rt_2)\}, v, \\
 & & \ \phi_1 \cup \phi_2 \cup \{v \mapsto \Gamma\{r\}\}) \\
 & & \text{where } \mathrm{T}^{r}_{\Gamma;\mathcal{P}}(P_i) = (V_i, E_i, rt_i, \phi_i) \text{ for } i \in \{1, 2\}, \\
 & & V_1 \cap V_2 = \emptyset \text{ and } v \notin V_1 \cup V_2 \\
\mathrm{T}^{r}_{\Gamma;\mathcal{P}}((\nu a : R)P) & \triangleq & (V \cup \{v\}, E \cup \{(v, rt)\}, v, \phi \cup \{v \mapsto \Gamma\{r\}\}) \\
 & & \text{where } \mathrm{T}^{r}_{\Gamma,a^r:R(T);\mathcal{P}}(P) = (V, E, rt, \phi), v \notin V \\
 & & \text{and } \Gamma, a^r : R(T); \Gamma\{r\} \vdash^{(\mathcal{U};\mathcal{P})}_{r} P \\
\mathrm{T}^{r}_{\Gamma;\mathcal{P}}([m = n]P) & \triangleq & (V \cup \{v\}, E \cup \{(v, rt)\}, v, \phi \cup \{v \mapsto \Gamma\{r\}\}) \\
 & & \text{where } \mathrm{T}^{r}_{\Gamma;\mathcal{P}}(P) = (V, E, rt, \phi) \text{ and } v \notin V
\end{array}
$$

Example 5. Let us consider a system consisting of a single user r running the following process

$$P = a(x).([x = b^r]a^r\langle x\rangle \mid [x = s](\nu c : S)(a^r\langle x\rangle \mid a^s\langle c^r\rangle))$$

and the RBAC schema $(\mathcal{U};\mathcal{P})$ defined as

$$
\begin{array}{ccccc}
\mathcal{U} & : & r & \mapsto & \{R_1, R_2\} \\
 & & a^r & \mapsto & R \\
 & & a^s & \mapsto & S
\end{array}
\qquad\qquad
\begin{array}{ccccc}
\mathcal{P} & : & R_1 & \mapsto & \{R^!, R^?\} \\
 & & R_2 & \mapsto & \{S^!\}
\end{array}
$$

A pictorial representation of $\mathrm{T}^{r}_{\mathcal{U};\mathcal{P}}(P)$ is given in Figure 1, where each node is explicitly named (the name is shown on the left-hand side of the node), the process operator associated with the node is written within the node, and the annotation (i.e., the set of rôles associated with the node) is depicted on the right-hand side of the node. Notice that in this case it suffices to parameterise $\mathrm{T}^{r}_{\mathcal{U};\mathcal{P}}$ only with a rôles-to-users assignment $\mathcal{U}$ instead of a (more complex) typing environment Γ (respecting $\mathcal{U}$) because no received value is used as an output channel. ⋄

Once process P has been translated into an annotated binary tree, the problem of finding a minimal refinement of P (in terms of the number of actions **role/yield** added) for user r under the schema $(\mathcal{U};\mathcal{P})$ can be reformulated as the problem of finding a partition of nodes such that:

- the partition generates the minimum number of blocks;

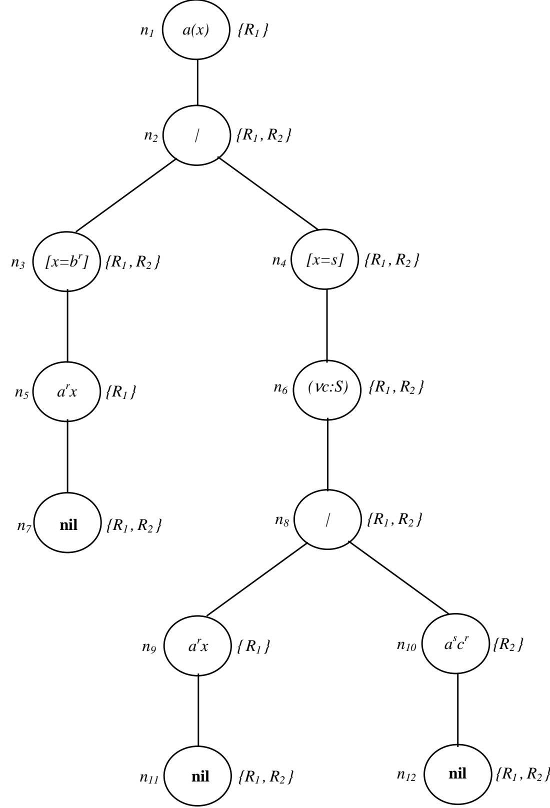

Figure 1: The annotated binary tree for process P from Example 5

- each block is a subtree[2];
- all the nodes v belonging to the same block have in common one of their annotating rôles, i.e., there exists $R \in \mathcal{R}$ such that $R \in \phi(v)$ for all v in the block.

We call such a problem the *minimal partition problem*.

We now describe a way to find a minimal partition of t and assign to each node v a label taken from $\phi(v)$. To this aim, for every node v and rôle R, the number $m[v, R]$ denotes the minimum number of blocks that can be obtained in the subtree rooted in v when v is labelled with R; we let $m[v, R] = \infty$ if $R \notin \phi(v)$. An algorithm for calculating the quantity $m[v, R]$ is given in Table 6. Intuitively, we work in a bottom-up fashion on the tree. When we consider a leaf v and a rôle R that authorises it, we can use R to generate a singleton tree (hence, $m[v, R] = 1$); if, on the other hand, R cannot authorise the (action associated with) the node, then it can be ignored because it cannot induce any block in the partition (hence, $m[v, R] = \infty$). When we consider an internal node v with just one child v' and a rôle R suitable for v, we can either try to include v in the

[2]Here, we use the term *subtree* to refer any connected subgraph of the given tree.

Visit t in postorder

When visiting the node v do

- if v is a leaf then

$$m[v,R] \ := \begin{cases} \infty & \text{if } R \notin \phi(v) \\ 1 & \text{otherwise} \end{cases}$$

- if v has only one child (and let it be v') then

$$m[v,R] \ := \begin{cases} \infty & \text{if } R \notin \phi(v) \\ \\ min\{\ m[v',R]\ , \\ \quad min_{S \neq R}\{m[v',S]\} + 1\ \} & \text{otherwise} \end{cases}$$

- if v has children v_1 and v_2 then

$$m[v,R] \ := \begin{cases} \infty & \text{if } R \notin \phi(v) \\ \\ min\{\ m[v_1,R] + m[v_2,R] - 1\ , \\ \quad min_{S \neq R}\{m[v_1,S]\} + min_{S \neq R}\{m[v_2,S]\} + 1 \\ \quad m[v_1,R] + min_{S \neq R}\{m[v_2,S]\}\ , \\ \quad min_{S \neq R}\{m[v_1,S]\} + m[v_2,R]\ \ \} & \text{otherwise} \end{cases}$$

Table 6: Computing function $m[v,R]$

subtree of v' induced by R (hence, $m[v,R] = m[v',R]$), or we can put it in a new subtree (that can possibly grow up when analysing v's ancestors). In the latter case, the new subtree is induced by R, while the subtree for v' can be induced by any other rôle S; thus, $m[v,R] = 1 + min_{S \neq R}\{m[v',S]\}$. Finally, the case for a node v with two children is similar, but it requires to examine four possible situations (according to whether v is included in both, in none, or in just one of the subtrees induced by R for v_1 and v_2).

Now, we can compute, for every node v, a rôle (v) which represents the rôle common to all the nodes in the block which v belongs to. To this aim, we assume a standard function (v) returning the father of node v in t (if any).

Visit t in preorder
When visiting the node v do
    $m_v \ := \ \{R \ : \ m[v,R] = min_{S \in \phi(v)}\{\ m[v,S]\ \}\ \}$
    if $v = rt$ or $(\quad (v)) \notin m_v$
        then $(v) := R$, where $R \in m_v$
        else $(v) := \quad (\quad (v))$

Notice that the choice of $R \in m_v$ (in the 'then' branch) is totally arbitrary: any such R can be chosen, since m_v only contains rôles that minimise $m[v,_]$. We can now formulate the soundness of the algorithm presented so far; a sketch of the proof is in Appendix A.2.

Proposition 4.1. *Function can be used to induce a partition of t's nodes in subtrees satisfying the requirements of the minimal partition. Moreover, the overall*

procedure takes $O(|V| \times K^2)$*, where K is the size of the largest set annotating a node of the tree.*

A solution of the original problem of properly putting actions **role/yield** in a process P can be then extracted easily from $t = \mathrm{T}\ ^{r}_{\Gamma;\mathcal{P}}(P)$ and from the associated function . Each block of the partition induced by represents the set of process operators in P that are under the influence of the rôle labelling the block. If the tree consists of a single node, then P must be **nil** and no auxiliary action is needed. Otherwise, the annotated (and partitioned) tree can be visited in preorder: depending on the value of (v) a pair of **role/yield** auxiliary actions are either added or not. In particular, the operator corresponding to the root is prefixed in P with **role** R, where $R = \ (rt)$. Then, no other actions are added until $(v) \neq \ (\ (v))$. In this case, the operator associated with node v is prefixed with **yield** R.**role** S, where $R = \ (\ (v))$ and $S = \ (v)$. We denote the process resulting from this procedure as $\mathrm{R}\ ^{r}_{\Gamma;\mathcal{P}}(P)$.

Example 6. Consider again Example 5 and the process P running inside user r:

$$P = a(x).([x = b^r]a^r\langle x\rangle \mid [x = s](\nu c : S)(a^r\langle x\rangle \mid a^s\langle c^r\rangle))$$

In this case, we have that function $m[v, R]$ is

	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}	n_{11}	n_{12}
R_1	2	2	1	2	1	2	1	2	1	∞	1	1
R_2	∞	3	2	2	∞	2	1	2	∞	1	1	1

Thus, a minimal partition of $\mathrm{T}\ ^{r}_{\mathcal{U};\mathcal{P}}(P)$ is given by the following two blocks

$$b_1 = \{n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8, n_9, n_{11}\}\ ,\ b_2 = \{n_{10}, n_{12}\},$$

from which we can extract the refined process

$$\begin{aligned}\mathrm{R}\ ^{r}_{\Gamma;\mathcal{P}}(P) = \mathbf{role}\,R_1.a(x).(\,[x = b^r]a^r\langle x\rangle\ \mid \\ [x = s](\nu c : S)(a^r\langle x\rangle \mid \mathbf{yield}\,R_1.\mathbf{role}\,R_2.a^s\langle c^r\rangle)\,)\end{aligned}$$

$\diamond$

The correctness of the approach can be stated as follows; some details on the proof are in Appendix A.2. Recall from the definition of the type system that r can run P if $\Gamma; \mathcal{U}(r) \vdash_r^{(\mathcal{U};\mathcal{P})} P$.

Proposition 4.2. *Let* $\mathcal{S} = (\mathcal{U}; \mathcal{P})$ *be a RBAC schema and P be a finite process without* **role/yield** *to be run by user r. Then*

1. $\Gamma; \mathcal{U}(r) \vdash_r^{\mathcal{S}} P$ *implies that* $\mathrm{R}\ ^{r}_{\Gamma;\mathcal{P}}(P)$ *is defined;*
2. *symmetrically,* Γ *respects* $\mathcal{U}$ *and* $\mathrm{R}\ ^{r}_{\Gamma;\mathcal{P}}(P) = P'$ *imply that* $\Gamma; \emptyset \vdash_r^{\mathcal{S}} P'$.

Intuitively, the first implication ensures that every process P that can be run by r under the schema $\mathcal{S}$ can be properly annotated with actions **role** and **yield**. The second implication states that the result of the annotation procedure we have just presented is a well-typed process for r in the schema given. Notice that $R^{r}_{\Gamma;\mathcal{P}}(P)$ is the minimal typeable process obtained from P by adding actions **role**/**yield**: this is an easy corollary of Propositions 4.1 and 4.2(2).

Finally, by Proposition 4.1, the overall procedure is linear in the size of P (i.e., in the number of its operators). Indeed, $|V|$ is proportional to the size of P and $\mathcal{U}(r)$ is an upper bound to the sets annotating the nodes of the tree (usually, $|\mathcal{U}(r)|$ is a small constant). This is the best asymptotic performance we could aim at, since we at least have to parse all P to properly add actions **role**/**yield**.

Least Privilege. Example 5 can be easily adapted to enforce the *least privilege* property [22, 26]. This is a well-known property requiring that every program and every user of the system operate using the least set of privileges necessary to complete their job.[3] Primarily, this principle limits the damage that can result from an accident or error. It also reduces the number of potential interactions among privileged programs to the minimum for correct operation, so that unintentional, unwanted, or improper uses of privileges are less likely to occur.

In our setting, we can say that a user r satisfies this property w.r.t. a schema $(\mathcal{U};\mathcal{P})$ while running in A if, whenever $A \longmapsto^* (\nu \widetilde{a^r} : \widetilde{R})(A' \parallel r\{\!\!\{ \alpha.P \}\!\!\}_\rho)$, it holds that ρ is a minimal (w.r.t. $|\mathcal{P}(\cdot)|$, i.e. the cardinality of the set of privileges associated to $\cdot$) set of rôles assignable to r that enables α. The approach presented in this section can be easily adapted to encompass the least privilege. The only thing we need to modify in the algorithm given above is the definition of function ϕ when building the tree for a process prefixed by action α. Let v be the node associated to α. By letting $enable(v) = \{R \in \Gamma\{r\} : \mathcal{P}(R) \text{ enable action } \alpha\}$, we let $\phi(v) = \{R \in enable(v) : |\mathcal{P}(R)| = min_{S \in enable(v)}\{\, |\mathcal{P}(S)|\,\}\}$.

Example 7. Consider a user r that connects to a mail server to read his e-mail and changes his password before quitting. Suppose that $(\mathcal{U};\mathcal{P})$ is such that $\{\mathtt{user}, \mathtt{admin}\} \subseteq \mathcal{U}(r)$, $\mathcal{U}(login^{e-server}) = \mathtt{login}$, $\mathcal{U}(read_mail^r) = \mathtt{read_mail}$, $\mathcal{U}(change_pwd^{e-server}) = \mathtt{change_pwd}$, $\{\mathtt{login}^!, \mathtt{read_mail}^?\} \subseteq \mathcal{P}(\mathtt{user})$ and $\mathcal{P}(\mathtt{admin}) = \mathcal{P}(\mathtt{user}) \cup \{\mathtt{change_pwd}^!\}$. Remarkably, the rôle admin gives r the permission to change his password. The following two systems

$$
\begin{aligned}
A_1 &= r\{\!\!\{ login^{e-server}\langle pwd\rangle.read_mail(x).change_pwd^{e-server}\langle pwd'\rangle \}\!\!\}_{\{\mathtt{admin}\}} \\
A_2 &= r\{\!\!\{ login^{e-server}\langle pwd\rangle.read_mail(x).\mathbf{role}\ \mathtt{admin}. \\
&\qquad change_pwd^{e-server}\langle pwd'\rangle.\mathbf{yield}\ \mathtt{admin} \}\!\!\}_{\{\mathtt{user}\}}
\end{aligned}
$$

are both well-typed in $(\mathcal{U},\mathcal{P})$. However, they differ in the auxiliary actions used: system A_2 satisfies the least-privilege requirement, since at each execution step user r owns

[3] To be precise, one should use the term *minimal privilege* instead of *least privilege*. Indeed, imagine that you have only two rôles, R_1 and R_2, such that R_1 enables actions $\{R^?, S^!\}$ and R_2 enables actions $\{R^?, S^?\}$; then, it is unclear which one would be the rôle giving the least privilege to execute an input from a channel of rôle R. Nevertheless, the current terminology in computer security uses the word "least" instead of "minimal"; we adhere to this trend.

a minimal set of permissions required to execute the action, while A_1 does not, since it activated the rôle `admin` also to login and read mails. $\diamond$

5 Possible Extensions of the Core RBAC Model

There is a wide spectrum of RBAC models differing on the operations supported. For example, [10, 23] propose various extensions of the core RBAC96 model we have used up to now, that in *loc. cit.* is referred to as $RBAC_0$. In particular, two variants are proposed: $RBAC_1$, adding rôle hierarchies, and $RBAC_2$, introducing constraints to permissions a user can exploit. In this section, we describe how these extensions can be easily expressed also in our framework.

5.1 Hierarchical RBAC

Hierarchies are a natural means for structuring rôles to reflect the organisational structure of an enterprise. A hierarchy is a partial order defining a seniority relation between rôles, whereby senior rôles acquire the permissions of their juniors, and junior rôles acquire the users authorised for their senior rôles. For example, in a health-care scenario, a rôle `cardiologist` is hierarchically superior to the rôle `doctor`, thus the cardiologist should have all the permissions of the doctor as well, and all the users that are authorised for the `cardiologist` rôle should be authorised also for the `doctor` rôle. This approach can increase the administrative efficiency of the enterprise: rather than specifying all the permissions of the junior rôle for the senior rôle, the junior rôle is specified as a permission of the senior rôle.

Our framework can be easily extended to express rôle hierarchies by adding a third component $\leq$, a partial order on $\mathcal{R}$, to the RBAC schema which becomes a triplet $(\mathcal{U};\mathcal{P};\leq)$. More specifically, when $R \leq S$, R is a *junior rôle* of S or, similarly, S is a *senior rôle* of R. Once a hierarchical RBAC schema $(\mathcal{U};\mathcal{P};\leq)$ has been fixed, we can define the set of junior rôles with respect to a given rôle R, or to a given set of rôles ρ, as $jnr(R) \triangleq \{S \ : \ S \leq R\}$ and $jnr(\rho) \triangleq \bigcup_{R \in \rho} jnr(R)$. Then, we may re-define $\mathcal{P}(\rho)$ as $\mathcal{P}(\rho) \triangleq \bigcup_{R \in jnr(\rho)} \mathcal{P}(R)$ and adapt both the type system and the barbed congruence take into account the hierarchy relation. In particular, in Definition 3.3.3(a) we can also extend the partial order $\leq$ with all $\leq'$ such that $\leq \cup \leq'$ is still a partial order. These modifications suffice to let the theory presented in Sections 2 and 3 properly work.

Example 8. Consider the health-care scenario again. In a hospital there is often a strict hierarchy establishing which operations are permitted depending on the position of the different employees. For example, the rôle `specialist` usually contains the rôles `doctor` and `intern`. This means that users activating rôle `specialist` are implicitly associated also with the permissions associated with the `doctor` and `intern` rôles, without the administrator having to explicitly list the `doctor` and `intern` permissions. This is an example of *multiple inheritance*, which provides us with the ability to inherit permissions from two or more rôle sources. Indeed, a rôle is composed from multiple subordinate rôles with fewer permissions as in the organisation and business structure which these rôles are intended to represent.

This hierarchy can be expressed by having $\mathtt{intern} \leq \mathtt{specialist}$ and $\mathtt{doctor} \leq \mathtt{specialist}$. Moreover, the rôles $\mathtt{cardiologist}$ and $\mathtt{radiologist}$ could each contain the $\mathtt{specialist}$ rôle. In this case, we also let $\mathtt{specialist} \leq \mathtt{cardiologist}$ and $\mathtt{specialist} \leq \mathtt{radiologist}$, leading to, e.g., $jnr(\mathtt{radiologist}) = \{\mathtt{radiologist}, \mathtt{specialist}, \mathtt{doctor}, \mathtt{intern}\}$. Now let

$$
\begin{array}{rcl}
\mathcal{U}(prescr_aspirin^{patient}) & = & \mathtt{prescr_aspirin} \\
\mathcal{U}(use_XRays^{hospital}) & = & \mathtt{use_XRays} \\
\mathtt{prescribe_aspirin}^{!} & \in & \mathcal{P}(\mathtt{doctor}) \\
\mathtt{use_XRays}^{!} & \in & \mathcal{P}(\mathtt{radiologist})
\end{array}
$$

then the user

$$
r\{\!|\, \mathbf{role}\ \mathtt{radiologist}.\ use_XRays^{hospital}\langle patient\rangle.\ prescr_aspirin^{patient}\langle posology\rangle\, |\!\}_{\emptyset}
$$

is typeable by only assuming that $\mathcal{U}(r) = \{\mathtt{radiologist}\}$. ◇

5.2 Constrained RBAC

The core RBAC model can be further extended by requiring different kinds of constraints to be satisfied before allowing a user to activate a rôle, or when defining the RBAC schema. According to [11], there are two possible forms of constraints: *static* and *dynamic*. The first ones deal with the permissions-to-rôles and with the rôles-to-users assignments. For example, it might be required that a user cannot be assigned some specified rôles at the same time, or that the same permission is not assigned to different rôles. These constraints are usually enforced during the definition of the RBAC schema [1, 16, 17, 25]. On the contrary, dynamic constraints deal with user sessions. By exploiting this form of constraints, it is possible, e.g., to assign to the same user two conflicting rôles, although requiring that these rôles are never activated simultaneously (for most practical purposes, this kind of requirement suffices).

We can easily extend our framework to deal in a uniform way also with different dynamic constraints. In this case, another component C is added to the RBAC schema. C is a finite set of binary predicates (that in this paper we assume to be first-order logic formulae built up over the atoms $\mathcal{R}$) relating a rôle and a set of rôles. Given a constrained RBAC schema $(\mathcal{U};\mathcal{P};C)$, we let $C(R,\rho)$ be $\bigwedge_{constr \in C} constr(R,\rho)$. As for the hierarchy extension, both the type system and the barbed congruence must be parameterised also with respect to C. Moreover, in rule (T-R$\hat{\ }$), the premise $C(R,\rho)$ must be added. With respect to Definition 3.3.3(a), we want to remark that extending C usually reduces the set of possible evolutions. Thus, instead of requiring that the equivalence holds in every extended schema, we now require that the equivalence is preserved when making the schema more liberal, i.e. when *reducing* the set of constraints.

By adapting the logic formulae in C to different situations, we are able to express the most typical examples of constraints:

1. *Mutual exclusion:* the same user can activate simultaneously at most one rôle in a mutually exclusive set. For example, if rôles R and S are mutually exclusive,

R cannot be activated in a user session where rôle S is already active, and vice-versa. This can be formalised as

$$constr_{R \oplus S}(R', \rho) \triangleq ((R' = R) \Rightarrow (S \notin \rho)) \wedge ((R' = S) \Rightarrow (R \notin \rho))$$

2. *Prerequisite rôle:* a user can activate rôle R only if he has already activated rôle S. This can be written as

$$constr_{S \to R}(R', \rho) \triangleq (R' = R) \Rightarrow (S \in \rho)$$

3. *Cardinality constraints (1):* at most n rôles can be activated in each user session. This can be expressed as

$$constr_{|\cdot| \leq n}(R, \rho) \triangleq |\rho \cup \{R\}| \leq n$$

4. *Cardinality constraints (2):* each user can own at most n permissions simultaneously. This can be enforced by requiring that

$$constr_{|\mathcal{P}(\cdot)| \leq n}(R, \rho) \triangleq |\mathcal{P}(\{R\} \cup \rho)| \leq n$$

Example 9 (Prerequisite rôle). The concept of *prerequisite rôle* is based on competency: in some circumstances, one may want to require a rôle to be activated only by a user already playing a certain rôle. For example, a common feature of a bank policy is to require an authentication phase to identify clients before any sensible operation, like money withdrawal. In practice, this amounts to ask for a valid identity document or a secret code/password. In our refined framework, this scenario can be modelled by letting the RBAC schema $(\mathcal{U}; \mathcal{P}; C)$ be such that $\{\texttt{client}, \texttt{authenticated}\} \in \mathcal{U}(r)$, $\mathcal{U}(wdrw^{bank}) = \texttt{wdrw}$, $\texttt{wdrw}^{!} \in \mathcal{P}(\texttt{client})$ and $constr_{\texttt{authenticated} \to \texttt{client}} \in C$. Hence, client

$$r\{\!\!| \, \textbf{role}\, \texttt{client}.wdrw^{bank}\langle amount \rangle \, |\!\!\}_{\emptyset}$$

cannot be typed, as the activation of rôle $\texttt{client}$ is forbidden by C, while client

$$r\{\!\!| \, \textbf{role}\, \texttt{authenticated}.\textbf{role}\, \texttt{client}.wdrw^{bank}\langle amount \rangle \, |\!\!\}_{\emptyset}$$

can be typed. ◇

Example 10 (Mutual Exclusion). For the sake of fairness, sometimes it is desirable to control the distribution of sensible permissions; e.g., a user willing to perform a sensible operation should be different from the user in charge of controlling the legality of such an operation. Consider a scenario where some scientists submit a paper to a journal. Clearly, the reviewers of that paper cannot be chosen among the authors of the paper itself. This requirement can be modelled in our framework by having two rôles, $\texttt{paperP} : \texttt{author}$ and $\texttt{paperP} : \texttt{reviewer}$, such that C contains the constraint $constr_{\texttt{paperP:author} \oplus \texttt{paperP:reviewer}}$. ◇

6 Related Work

To the best of our knowledge, no previous study building on process-calculi has ever been conducted on RBAC. A number of papers have instead dealt with the formal specification and verification of RBAC schemata. In [16, 25] formal methods are used only to verify the correctness of the schema definition but not of the whole system. In [25], the ALLOY language is used to detect possible conflicts in RBAC schemata supporting simultaneously delegation of authority and separation of duties. A constraint analyser allows the schema validation to be computed automatically. In [16, 17], the authors use a graph transformation which combines an intuitive visual description of the RBAC schema with solid semantical foundations. In [1], Ahn et al. introduce a formal language for the specification of more sophisticated rôle-based authorisation constraints, such as prohibition and obligation constraints. These approaches are complementary to ours: they can be integrated with our technique in order to verify the consistency of a schema $\mathcal{S}$, but they do not give any hint about the correct execution of a system as our method does.

In [3], Bertino et al. develop a logical framework for reasoning about access control models. The framework is general enough to model discretionary, mandatory and rôle-based access control models. Such a framework is useful for comparing the expressive power of the models, but it cannot be used to verify the correct execution of a system under a given schema.

Probably, the most related work, although not aiming at studying RBAC systems, is [6], insofar as rôles can be understood as (privilege) groups. *Groups* are introduced in *loc. cit.* as types for channels, and used to limit their visibility. A type system ensures that channels belonging to a fresh group can be only used by processes within the initial scope of the group. Thus, processes can access channels according to their physical distribution (with respect to group restrictions). In our work this feature is modified so that not only the place where the process runs (i.e., the user running the process) but also its execution history (i.e., the user session where the process runs and the associated activations/deactivations of rôles) is relevant to execute an action. E.g., outputs over a^r of group R can be executed only by processes whose user r is such that $R^! \in \mathcal{P}(\mathcal{U}(r))$; moreover, such an action must be enabled by at least one of the rôles active in r's session. The set of such sessions changes according to the computation and, thus, the processes enabled to access a channel change dynamically. In this sense, this work can be seen as a calculus of *dynamic* groups.

Acknowledgements. This work has been partially supported by EU FET – Global Computing initiative, projects MIKADO IST-2001-32222 and MyThS IST-2001-32617, by the MIUR project "Abstract Interpretation: Design and Applications" (AIDA), and by the FIRB project RBAU018RCZ_002. The funding bodies are not responsible for any use that might be made of the results presented here.

A first draft of this paper has been improved by following the valuable suggestions and comments of the CSFW'04 and of the JCS anonymous referees. The second author is very grateful to Angelo Monti for his fundamental support in the development of the algorithm of Table 6.

A Technical Proofs

In this section, we give details on the proofs omitted from the body of the paper.

A.1 Proofs of Section 2

To prove subject reduction, we first need three lemmata, that are standard results for a type system. The first one states that names can be replaced with other names of the same type. The second one states that enlarging the assumptions in a type judgement does not compromise the inference of the judgement itself. Finally, the third result states that well-typedness is an invariant of structural congruence. To prove the latter, we formally define a *system context*, as $\equiv$ is closed under all such contexts. Formally, a context $C[\cdot]$ is a system with an occurrence of a 'hole' to be filled with any system A, thus yielding $C[A]$. Formally,

$$C[\cdot] \ ::= \ [\cdot] \ | \ C[\cdot] \parallel B \ | \ (\nu a^r\!:\!R)C[\cdot]$$

Lemma A.1 (Substitution). *If* $\Gamma, x : T, \Gamma'; \rho \vdash^S_r P$ *and* $\Gamma \vdash n : T$*, then* $\Gamma, \Gamma'; \rho \vdash^S_r P[n/x]$.

Proof. The proof is by induction on the depth of the inference of the type judgement $\Gamma, x : T, \Gamma'; \rho \vdash^S_r P$. The proof is quite standard and faithfully rephrases the corresponding result for the pure π-calculus; thus, we omit it. ◻

Lemma A.2 (Weakening). *If* $\Gamma \vdash^{(\mathcal{U};\mathcal{P})} A$ *then* $\Gamma, \widetilde{n} : \widetilde{T} \vdash^{(\mathcal{U}\cup\widetilde{n}:\widetilde{T};\mathcal{P}\cup\mathcal{P}')} A$ *for all* $\mathcal{P}'$ *and* $\widetilde{n} : \widetilde{T}$ *such that* $\widetilde{n} \cap dom(\mathcal{U}) = \emptyset$.

Proof. The proof is by induction on the depth of the inference for $\Gamma \vdash^{(\mathcal{U};\mathcal{P})} A$. ◻

Lemma A.3. *If* $\Gamma \vdash^S A$ *and* $A \equiv B$*, then* $\Gamma \vdash^S B$.

Proof. By mutual induction on the depth of the inferences for $A \equiv B$ and $B \equiv A$. Let us consider how $A \equiv B$ has been inferred; the case for $B \equiv A$ is similar. The base case covers the axioms in Table 3; all the cases are simple. The inductive steps for symmetry and transitivity follow straightforwardly. For context closure, let $A \triangleq C[A_1]$ and $B \triangleq C[B_1]$, for some $A_1 \equiv B_1$. We now work by induction on the structure of $C[\cdot]$. The base case is when $C[\cdot] \triangleq [\cdot]$ and is trivial. For the inductive case, let us reason by case analysis on the outermost operator of $C[\cdot]$. If $C[\cdot] \triangleq \mathcal{D}[\cdot] \parallel \bar{A}$, then, by using (T-S P), we know by hypothesis that $\Gamma \vdash^S \mathcal{D}[A_1]$ and $\Gamma \vdash^S \bar{A}$. By induction hypothesis, as $\mathcal{D}[\cdot]$ is smaller than $C[\cdot]$, it holds that $\Gamma \vdash^S \mathcal{D}[B_1]$; then, by (T-S P), $\Gamma \vdash^S B$. The case for $C[\cdot] \triangleq (\nu a^r\!:\!R)\mathcal{D}[\cdot]$ is similar, but relies on (T-S R). ◻

Theorem 2.1 (Subject Reduction)**.** *If* $\Gamma \vdash^S A$ *and* $A \longmapsto A'$*, then* $\Gamma \vdash^S A'$.

Proof. By induction on the depth of the derivation of $A \longmapsto A'$.
Base Step: By case analysis on the axioms of the second part of Table 3.

(R-C) By hypothesis, we have that $\Gamma \vdash^{S} r\{\!|\, a(x).P \,|\!\}_{\rho} \parallel s\{\!|\, a^r\langle n\rangle.Q \,|\!\}_{\rho'}$. Due to the form of the system involved, (T-S P) is the last rule applied to deduce the type judgement, hence we also have that $\Gamma \vdash^{S} r\{\!|\, a(x).P \,|\!\}_{\rho}$ and $\Gamma \vdash^{S} s\{\!|\, a^r\langle n\rangle.Q \,|\!\}_{\rho'}$. The latter two judgements must have been derived by using (T-S), with $\Gamma \vdash r : \rho''[\widetilde{a} : \widetilde{C}]$ and $\rho \subseteq \rho''$, $\Gamma \vdash s : \rho'''[\widetilde{a'} : \widetilde{C'}]$ and $\rho' \subseteq \rho'''$, $\Gamma;\rho \vdash^{S}_{r} a(x).P$ and $\Gamma;\rho' \vdash^{S}_{s} a^r\langle n\rangle.Q$. Judgement $\Gamma;\rho \vdash^{S}_{r} a(x).P$ must have been derived by using (T-I), with $\Gamma \vdash a^r : R(T)$ and $\Gamma, x : T;\rho \vdash^{S}_{r} P$, whereas judgement $\Gamma;\rho' \vdash^{S}_{s} a^r\langle n\rangle.Q$ must have been derived by using (T-O), with $\Gamma \vdash a^r : R(T)$, $\Gamma \vdash n : T$, and $\Gamma;\rho' \vdash^{S}_{s} Q$. By Lemma A.1, we get that $\Gamma;\rho \vdash^{S}_{r} P[n/x]$. By a double application of (T-S) and of (T-S P), we get that $\Gamma \vdash^{\mathcal{P}} r\{\!|\, P[n/x] \,|\!\}_{\rho} \parallel s\{\!|\, Q \,|\!\}_{\rho'}$, as required.

(R-R) By hypothesis, we have that $\Gamma \vdash^{S} r\{\!|\, \mathbf{role}\, R.P \,|\!\}_{\rho}$. Due to the form of the system involved, (T-S) is the last rule applied to deduce the type judgement, hence we also have that $\Gamma \vdash r : \rho'[\widetilde{a} : \widetilde{C}]$, $\rho \subseteq \rho'$ and $\Gamma;\rho \vdash^{S}_{r} \mathbf{role}\, R.P$. Judgement $\Gamma \vdash r : \rho'[\widetilde{a} : \widetilde{C}]$ must have been derived by using (T-I $_1$), with $\Gamma(r) = \rho'[\widetilde{a} : \widetilde{C}]$; judgement $\Gamma;\rho \vdash^{S}_{r} \mathbf{role}\, R.P$ has been derived by using (T-R$\hat{}$), with $\Gamma \vdash r : \rho'[\widetilde{a'} : \widetilde{C'}]$, $\Gamma;\rho \cup \{R\} \vdash^{S}_{r} P$ and $R \in \rho''$. Then, $\rho \cup \{R\} \subseteq \rho'$; by rule (T-S), we can derive $\Gamma \vdash^{S} r\{\!|\, P \,|\!\}_{\rho \cup \{R\}}$, as required.

(R-Y) By hypothesis, we have that $\Gamma \vdash^{S} r\{\!|\, \mathbf{yield}\, R.P \,|\!\}_{\rho}$. Due to the form of the system involved, (T-S) is the last rule applied to deduce the type judgement, hence we also have that $\Gamma \vdash r : \rho'[\widetilde{a} : \widetilde{C}]$, $\Gamma;\rho \vdash^{S}_{r} \mathbf{yield}\, R.P$ and $\rho \subseteq \rho'$. Judgement $\Gamma;\rho \vdash^{S}_{r} \mathbf{yield}\, R.P$ has been derived by using (T-Y), with $\Gamma;\rho \setminus \{R\} \vdash^{S}_{r} P$ and $R \in \rho$. By applying rule (T-S) to $\Gamma;\rho \setminus \{R\} \vdash^{S}_{r} P$, we can derive $\Gamma \vdash^{S} r\{\!|\, P \,|\!\}_{\rho \setminus \{R\}}$, as required.

Inductive Step: By case analysis of the last applied operational rule of the second part of Table 3.

(R-R) By definition, $A \triangleq (\nu a^r : R)B$ and $A' \triangleq (\nu a^r : R)B'$, where $B \longmapsto B'$; moreover, by hypothesis, we have that $\Gamma \vdash^{S} (\nu a^r : R)B$. By rule (T-S R), we have that $\Gamma, a^r \colon R(T) \vdash^{S} B$, for some T. By induction hypothesis, $\Gamma, a^r \colon R(T) \vdash^{S} B'$ that, by rule (T-S R), implies $\Gamma \vdash^{S} A'$.

(R-P) By hypothesis, $A \triangleq A_1 \parallel B$ is well-typed; hence, A_1 and B are well-typed too. Moreover, $A_1 \longmapsto A_1'$ and the induction hypothesis imply that A_1' is well-typed; thus, $A_1' \parallel B \triangleq A'$ is well-typed too.

(R-S) We now have that $A \equiv A_1 \longmapsto A_2 \equiv A'$. By well-typedness of A and Lemma A.3, it follows that A_1 is well-typed; by induction hypothesis, it follows that A_2 is well-typed and, again by Lemma A.3, A' is well-typed. □

Theorem 2.2 (Type Safety). *If A is well-typed in $\mathcal{S}$, then $A \rightsquigarrow_{\mathcal{S}}$ cannot hold.*

Proof. We prove the contrapositive, i.e. $A \rightsquigarrow_{\mathcal{S}}$ implies that A cannot be well-typed in $\mathcal{S}$; this is done by induction on the depth of the inference for $A \rightsquigarrow_{\mathcal{S}}$. Let $\mathcal{S}$ be $(\mathcal{U};\mathcal{P})$. For the base case, we consider only one sample, namely when the judgement

has been inferred via (E-I); the other cases are similar. By definition, A is $r\{\!| b(x).P |\!\}_\rho$ and $S^? \notin \mathcal{P}(\rho)$, for $S = \mathcal{U}(b^r)$. Thus, for any Γ respecting $\mathcal{U}$, it cannot hold that $\Gamma; \rho \vdash^S_r b(x).P$: indeed, the premises of rule (T-I) (that is the only applicable to infer the judgement) cannot be satisfied.

For the inductive step, we only consider the case when the last rule used is (E-R); the cases for (E-P) and (E-S) are simpler (the latter one relies on Lemma A.3). By definition, A is $(\nu a^r : R)B$ and $B \rightsquigarrow_{(\mathcal{U} \uplus \{a^r : R\}; \mathcal{P})}$. By induction hypothesis, B cannot be well-typed in $(\mathcal{U} \uplus \{a^r : R\}; \mathcal{P})$, i.e. for every Γ respecting $\mathcal{U} \uplus \{a^r : R\}$, judgement $\Gamma \vdash^{(\mathcal{U} \uplus \{a^r : R\}; \mathcal{P})} B$ cannot be inferred. Since Γ respects $\mathcal{U} \uplus \{a^r : R\}$, it must be that $\Gamma = \Gamma', a^r : R$; this easily implies that there is no Γ' such that $\Gamma' \vdash^S (\nu a^r : R)B$, as desired. □

A.2 Proofs of Section 4

Proposition 4.1. *Function can be used to induce a partition of t's nodes in subtrees satisfying the requirements of the minimal partition. Moreover, the overall procedure takes $O(|V| \times K^2)$, where K is the size of the largest set annotating a node of the tree.*

Proof. Having computed function $(\cdot)$, we proceed in the following way:

Visit t in preorder
When visiting the node v do
if $v = rt$ or $(v) \neq ((v))$
then add v in a new block
else add v in the block of (v)

It should be clear that the output of this procedure is a partition of V (no block is empty and each node is inserted in exactly one block); we call it the partition induced by $(\cdot)$. We have to prove that this partition satisfies the following conditions: (a) each block is a subtree of t; (b) each block β is such that $\exists R \in \mathcal{R}\ \forall v \in \beta : R \in \phi(v)$ (in this case, we call R the *pivot* for β); (c) it has the minimum number of blocks satisfying the previous two properties.

Condition (a) is proved by induction on the size of the generic block β. The base case is for $\beta = \{v\}$ and is trivial, as a single node is a subtree of t. For the inductive step, let $\beta = \beta' \cup \{v\}$, where v is the last node added by the above procedure. By construction, it must be that $(v) \in \beta'$ and, by induction hypothesis, β' is a subtree of t. Thus, easily, also β is a subtree of t. Condition (b) is simple: by construction, it holds that $(v) = (v')$ and $(v) \in \phi(v)$, for every v and v' belonging to β. Condition (c) easily follows, once we prove the following Lemma.

Lemma A.4 (Soundness of the algorithm in Table 6). *If $m[v, R] = h \neq \infty$, then there exists a partition of the subtree rooted in v with h blocks such that it satisfies conditions (a) and (b) above, and R is the pivot of v's block; moreover, each partition satisfying these properties has at least h blocks.*

Proof. By induction on the height of the tree rooted in v. For notational convenience, we denote the subtree of t rooted in v as t_v. The base case is when v is a leaf and is trivial. For the inductive step, we only consider the case when v has just one child, v'; the case when v has two children is similar. Since $m[v, R] \neq \infty$, it holds that $R \in \phi(v)$; thus, R can be the pivot of v's block. Since $(\cdot)$ is defined (by the hypothesis of Proposition 4.1), there must exist $S \in \phi(v')$; thus, $m[v', S] \neq \infty$. By induction hypothesis, there exists a partition of $t_{v'}$ with $m[v', S]$ blocks that satisfies conditions (a) and (b), and with S as pivot of v''s block; moreover, each partition satisfying these properties has at least $m[v', S]$ blocks. If R is not one of such S, then a minimal partition of t_v with R as pivot of v's block can be obtained by putting v is a block on its own and by considering the partition of $t_{v'}$ induced by a S that minimises $m[v', _]$. Otherwise, adding v to the block of v' in the partition of $t_{v'}$ induced by R could generate a minimal partition of t_v or not. In the first case, the partition of t_v has $m[v', R]$ blocks; in the second case, we put v in a block on its own and the resulting partition has $m[v', R] + 1$ blocks. In both cases, it is easy to prove that no partition with less blocks can exist. □

Now, let $(rt) = R$. Trivially, the partition induced by function $(\cdot)$ has $m[rt, R]$ blocks; thus, by Lemma A.4, each partition satisfying (a) and (b) has at least $m[rt, R]$ blocks. This proves (c).

We conclude with the complexity of the overall algorithm. The algorithm in Table 6 to compute $m[_, _]$ works in $O(|V| \times K^2)$. Indeed, matrix m has $|V|$ rows and K columns, and each element of this matrix is written exactly once. Moreover, to write a generic element $m[v, R]$, we have to check whether $R \in \phi(v)$ (this requires $O(K)$, as $|\phi(v)| \leq K$) and to analyse the rows associated to the children of v (if any); the latter task requires $O(K)$, that leads the overall complexity to $O(|V| \times K^2)$. The algorithm for computing function works in $O(|V| \times K)$. Indeed, for each node v, it has to compute m_v: this requires $O(K)$, as it has to scan all the row of m associated to v. Finally, the partition induced by $(\cdot)$ is derived by a preorder visit, that is linear in $|V|$. □

We now consider Proposition 4.2 and prove its two claims separately; for both of them, we only present a key sample, leaving the other cases (that are similar) to the interested reader.

1. Let $\mathcal{S} = (\mathcal{U}; \mathcal{P})$ be a RBAC schema and P be a finite process without **role/yield** *to be run by user r. Then $\Gamma; \mathcal{U}(r) \vdash_r^{\mathcal{S}} P$ implies that* R $^{r}_{\Gamma;\mathcal{P}}(P)$ *is defined.*

Proof. The proof is by structural induction on P. The base step is trivial: P is **nil** and, by definition, R $^{r}_{\Gamma;\mathcal{P}}(\mathbf{nil}) = \mathbf{nil}$. For the inductive step, we only consider the case for $P = a(x).Q$. By hypothesis and by rule (T-I), we have that

$$\frac{\Gamma \vdash a^r : R(T) \qquad R \overset{?}{\in} \mathcal{P}(\mathcal{U}(r)) \qquad \Gamma, x : T; \mathcal{U}(r) \vdash_r^{\mathcal{S}} Q}{\Gamma; \mathcal{U}(r) \vdash_r^{\mathcal{S}} P}$$

By induction hypothesis, R $^{r}_{\Gamma,x:T;\mathcal{P}}(Q)$ is defined; this implies that T $^{r}_{\Gamma,x:T;\mathcal{P}}(Q)$ is defined and is equipped with the matrix $m[\,\cdot\,,\,\cdot\,]$. Now, by construction, T $^{r}_{\Gamma;\mathcal{P}}(P)$ is defined; moreover, it can be equipped with a matrix $m'[\,\cdot\,,\,\cdot\,]$ such that

$$m'[x,R] \;=\; \begin{cases} m[x,R] & \text{if } x \text{ is not the root of T } {}^{r}_{\Gamma;\mathcal{P}}(P) \\ \\ min\{\, m[v',R]\,, \\ \qquad min_{S\neq R}\{m[v',S]\} + 1\,\} & \text{otherwise} \end{cases}$$

where, in the second case, v' is the root of T $^{r}_{\Gamma,x:T;\mathcal{P}}(Q)$. By the premise of (T-I), we know that there exists a rôle $S \in \mathcal{U}(r)$ such that $R \overset{?}{\in} \mathcal{P}(S)$. This fact, together with the fact that R $^{r}_{\Gamma,x:T;\mathcal{P}}(Q)$ is defined, implies that there exists $S \in \mathcal{U}(r)$ such that $m'[v,S] \neq \infty$, where v is the root of T $^{r}_{\Gamma;\mathcal{P}}(P)$. This fact suffices to conclude that R $^{r}_{\Gamma;\mathcal{P}}(P)$ is defined. □

2. *Let $\mathcal{S} = (\mathcal{U};\mathcal{P})$ be a RBAC schema, Γ be a typing environment respecting $\mathcal{U}$ and P be a finite process without* **role/yield** *to be run by user r. Then* R $^{r}_{\Gamma;\mathcal{P}}(P) = P'$ *implies that* $\Gamma;\emptyset \vdash^{\mathcal{S}}_{r} P'$.

Proof. Again, the proof is by structural induction on P. The base step is trivial: $P = P' \triangleq$ **nil** and $\Gamma;\emptyset \vdash^{\mathcal{S}}_{r}$ **nil**. For the inductive step, we only consider the case for $P = a(x).Q$. By construction, we have that $P' =$ **role** $R.a(x).Q'$, where R is the label of the root of T $^{r}_{\Gamma;\mathcal{P}}(P)$, that exists by hypothesis. By construction, this latter fact implies that T $^{r}_{\Gamma,x:T;\mathcal{P}}(Q)$ exists and that $\Gamma \vdash a^{r} : S(T)$, for some rôle S and type T. Let us now consider the matrix $m'[\,\cdot\,,\,\cdot\,]$ obtained from the matrix $m[\,\cdot\,,\,\cdot\,]$ for T $^{r}_{\Gamma;\mathcal{P}}(P)$ by deleting the row associated with the root of T $^{r}_{\Gamma;\mathcal{P}}(P)$. We can now say that there exists a rôle R' such that $m'[v',R'] \neq \infty$, where v' is the root of T $^{r}_{\Gamma,x:T;\mathcal{P}}(Q)$.[4] This suffices to conclude that R $^{r}_{\Gamma,x:T;\mathcal{P}}(Q)$ is defined; let us say that it returns the process Q''. By induction hypothesis, $\Gamma, x : T;\emptyset \vdash^{\mathcal{S}}_{r} Q''$. Again, by construction it must be that $Q'' =$ **role** $R''.\hat{Q}$; we now consider the only possible cases:

- if $(v) =$ (v'), i.e. $R = R''$, then $Q' = \hat{Q}$. In this case, we have that

$$\dfrac{R \in \Gamma\{r\} \qquad \dfrac{\Gamma \vdash a^{r} : S(T) \quad S \overset{?}{\in} \mathcal{P}(R) \quad \Gamma, x : T;\{R\} \vdash^{\mathcal{S}}_{r} Q'}{\Gamma;\{R\} \vdash^{\mathcal{S}}_{r} a(x).Q'}}{\Gamma;\emptyset \vdash^{\mathcal{S}}_{r} P'}$$

  This inference holds by using rules (T-Rˆ) and (T-I). Moreover, notice that $R \in \Gamma\{r\}$ and $S \overset{?}{\in} \mathcal{P}(R)$ must hold, otherwise (v) cannot be R. Finally, $\Gamma, x : T;\{R\} \vdash^{\mathcal{S}}_{r} Q'$ is implied by the induction hypothesis.

- if $(v) \neq$ (v'), then $Q' =$ **yield** $R.Q''$. This case is similar to the previous one, but $\Gamma, x : T;\{R\} \vdash^{\mathcal{S}}_{r} Q'$ is inferred from the induction hypothesis, by using rule (T-Y). □

[4] To see this, proceed by contradiction. Assume that, for all R', $m'[v',R'] = \infty$; then $m[v,R'] = \infty$, where v is the root of T $^{r}_{\Gamma;\mathcal{P}}(P)$, for every rôle R' (see Table 6, second item). Thus, R $^{r}_{\Gamma;\mathcal{P}}(P)$ would be undefined, as function would be. Contradiction.

B Alternative Characterisation of Barbed Congruence

As pointed out in Section 3, barbed congruence is hard to prove because of its universal quantification over language contexts. A standard way to overcome this problem is to reformulate the semantics of the language via a *labelled transition system* (LTS for short), that makes apparent the external interaction offered, and build up over it a bisimulation, adequate for barbed congruence. In this section, we present a possible way to adapt known techniques to our framework; however, to make the presentation lighter, most of the proofs in this section are only sketched; the interested reader is referred to a technical report [5] for full details.

The LTS allows to study system components in isolation and compositionally. Thus, in general, we cannot assume such components to be well-typed, as this would require a full knowledge of the system. Hence, we embody in the LTS some dynamic policy checks. In this way, the LTS also provides a tight operational specification for the minimal engine underlying any implementation of a RBAC-based run-time system.

The standard way to describe the interactions a system can offer externally is by labelling the system evolution with this information. Thus, we define a labelled transition system, $\xrightarrow{\mu}$, that makes apparent the action performed (and, thus, the external interaction offered). In order to account for systems' rôles varying over time, the LTS relates *configurations*, i.e. pairs $\mathcal{S} \triangleright A$ made up of a RBAC schema $\mathcal{S}$ and a system A. Configurations are ranged over by $D, E, \ldots$. The labels of the LTS are derived from those of the π-calculus and can be described as follows.

$$\mu \quad ::= \quad \tau \quad | \quad a^r n \quad | \quad a^r n : R \quad | \quad \overline{a}^r n \quad | \quad \overline{a}^r n : R$$

Label τ represents an internal computation of the system. Labels $\overline{a}^r n$ and $a^r n$ describe the intention to send/receive value n, known to the environment, on/from channel a^r. Labels $\overline{a}^r n : R$ and $a^r n : R$ are similar to but the value sent/received is 'fresh' (i.e. unknown to the environment) and has group R. We now extend functions F (_) and B (_) to labels.

Label	F (_)	B (_)
τ	$\emptyset$	$\emptyset$
$a^r n$	$\{a^r, n\}$	$\emptyset$
$\overline{a}^r n$	$\{a^r, n\}$	$\emptyset$
$a^r n : R$	$\{a^r\}$	$\{n\}$
$\overline{a}^r n : R$	$\{a^r\}$	$\{n\}$

The definition of $\xrightarrow{\mu}$ is given in Tables 7 and 8. The overall structure of the LTS is similar to π-calculus' early-style one (see, e.g., [24]) and implicitly assumes alpha-conversion. The premises of rules (LTS-K-I), (LTS-F-I), (LTS-O), (LTS-R^) and (LTS-Y) adapt respectively the premises of the typing rules (T-I), (T-O), (T-R^) and (T-Y), and block the evolution of ill-typed systems. Rule (LTS-K-I) can be applied when the received value is known to the schema, while (LTS-F-I) is used when a fresh value (i.e. unknown to the schema) is received. In this case, the schema is extended to record the group of the fresh value. Similarly, when extruding a restricted channel b^s, rule (LTS-O) enlarges the relation $\mathcal{U}$ of the current configuration by recording that b^s has the rôle declared in the

(LTS-Rˆ)
$$\frac{R \in \mathcal{U}(r)}{(\mathcal{U};\mathcal{P}) \triangleright r\{\!\!|\, \textbf{role}\, R.P \,|\!\!\}_{\rho} \xrightarrow{\tau} (\mathcal{U};\mathcal{P}) \triangleright r\{\!\!|\, P \,|\!\!\}_{\rho \cup \{R\}}}$$

(LTS-Y)
$$\frac{R \in \rho}{\mathcal{S} \triangleright r\{\!\!|\, \textbf{yield}\, R.P \,|\!\!\}_{\rho} \xrightarrow{\tau} \mathcal{S} \triangleright r\{\!\!|\, P \,|\!\!\}_{\rho - \{R\}}}$$

(LTS-O)
$$\frac{\mathcal{U}(a^s) = R \qquad R^! \in \mathcal{P}(\rho)}{(\mathcal{U};\mathcal{P}) \triangleright r\{\!\!|\, a^s\langle n\rangle.P \,|\!\!\}_{\rho} \xrightarrow{\overline{a}^s n} (\mathcal{U};\mathcal{P}) \triangleright r\{\!\!|\, P \,|\!\!\}_{\rho}}$$

(LTS-K-I)
$$\frac{\mathcal{U}(a^r) = R \quad R^? \in \mathcal{P}(\rho) \quad n \in dom(\mathcal{U})}{(\mathcal{U};\mathcal{P}) \triangleright r\{\!\!|\, a(x).P \,|\!\!\}_{\rho} \xrightarrow{a^r n} (\mathcal{U};\mathcal{P}) \triangleright r\{\!\!|\, P[^n\!/x] \,|\!\!\}_{\rho}}$$

(LTS-F-I)
$$\frac{\mathcal{U}(a^r) = R \qquad R^? \in \mathcal{P}(\rho) \qquad n \notin dom(\mathcal{U})}{(\mathcal{U};\mathcal{P}) \triangleright r\{\!\!|\, a(x).P \,|\!\!\}_{\rho} \xrightarrow{a^r n:S} (\mathcal{U} \uplus \{n : S\};\mathcal{P}) \triangleright r\{\!\!|\, P[^n\!/x] \,|\!\!\}_{\rho}}$$

Table 7: Axioms for the Labelled Transition System

restriction. The information about a fresh/extruded channel is deleted from the schema when the channel is communicated: indeed, the restriction is pushed back in the system and closes the scope of the channel – cf. rule (LTS-C). Notice that a bound output can synchronise only with a fresh input (and vice versa), and the rôle declared for the extruded/fresh channel must be the same. Also observe that τ-moves do not modify the schema $\mathcal{S}$.

The semantics given in Table 3 and the LTS just presented are related by the following Proposition.

Proposition B.1. *If $\mathcal{S} \triangleright A \xrightarrow{\tau} \mathcal{S} \triangleright A'$, then $A \longmapsto A'$. Conversely, if A is well-typed in $\mathcal{S}$, then $A \longmapsto A'$ implies $\mathcal{S} \triangleright A \xrightarrow{\tau} \mathcal{S} \triangleright B$, for some $B \equiv A'$.*

Proof. The first statement is proved by a simple induction over the depth of the inference for $\xrightarrow{\tau}$. The second statement is proved by induction over the depth of the shortest inference for $\longmapsto$. The only complicate case is when the last rule applied to infer the reduction is (R-S), i.e. $A \equiv B$, and $B \longmapsto B'$ and $B' \equiv A'$. We proceed by mutual induction on the depth of the inferences for $A \equiv B$ and $B \equiv A$; notice that we can assume that the last rule to infer $B \longmapsto B'$ is not (R-S), otherwise the original inference of $A \longmapsto A'$ could be shortened, thanks to transitivity of $\equiv$. □

Next, we build upon this LTS a standard bisimulation. As usual, $\Longrightarrow$ denotes the reflexive and transitive closure of $\xrightarrow{\tau}$, and $\stackrel{\mu}{\Longrightarrow}$ denotes $\Longrightarrow\xrightarrow{\mu}\Longrightarrow$. Finally, $\stackrel{\hat{\mu}}{\Longrightarrow}$ is $\Longrightarrow$ if $\mu = \tau$, and $\stackrel{\mu}{\Longrightarrow}$ otherwise.

Definition B.1 (Bisimilarity). A *bisimulation* is a binary symmetric relation $\mathfrak{R}$ between configurations such that, if $(D, E) \in \mathfrak{R}$ and $D \xrightarrow{\mu} D'$, there exists a configuration E' such that $E \stackrel{\hat{\mu}}{\Longrightarrow} E'$ and $(D', E') \in \mathfrak{R}$. *Bisimilarity*, $\approx$, is the largest bisimulation.

(LTS-C)

$$\frac{\mathcal{S} \triangleright A \xrightarrow{a^r n} \mathcal{S} \triangleright A' \qquad \mathcal{S} \triangleright B \xrightarrow{\overline{a}^r n} \mathcal{S} \triangleright B'}{\mathcal{S} \triangleright A \parallel B \xrightarrow{\tau} \mathcal{S} \triangleright A' \parallel B'}$$

(LTS-R)

$$\frac{(\mathcal{U} \uplus \{a^r\!:\!R\};\mathcal{P}) \triangleright A \xrightarrow{\mu} (\mathcal{U} \uplus \{a^r\!:\!R\};\mathcal{P}) \triangleright A' \qquad a^r \notin \mathrm{F}\ (\mu)}{(\mathcal{U};\mathcal{P}) \triangleright (\nu a^r\!:\!R)A \xrightarrow{\mu} (\mathcal{U};\mathcal{P}) \triangleright (\nu a^r\!:\!R)A'}$$

(LTS-O)

$$\frac{(\mathcal{U} \uplus \{b^s\!:\!S\};\mathcal{P}) \triangleright A \xrightarrow{\overline{a}^r b^s} (\mathcal{U} \uplus \{b^s\!:\!S\};\mathcal{P}) \triangleright A' \qquad a^r \neq b^s}{(\mathcal{U};\mathcal{P}) \triangleright (\nu b^s\!:\!S)A \xrightarrow{\overline{a}^r b^s:S} (\mathcal{U} \uplus \{b^s\!:\!S\};\mathcal{P}) \triangleright A'}$$

(LTS-C)

$$\frac{\mathcal{S} \triangleright A \xrightarrow{a^r b^s:S} \mathcal{S}' \triangleright A' \qquad \mathcal{S} \triangleright B \xrightarrow{\overline{a}^r b^s:S} \mathcal{S}' \triangleright B' \qquad b^s \notin \mathrm{F}\ (A)}{\mathcal{S} \triangleright A \parallel B \xrightarrow{\tau} \mathcal{S} \triangleright (\nu b^s\!:\!S)(A' \parallel B')}$$

(LTS-P)

$$\frac{\mathcal{S} \triangleright A \xrightarrow{\mu} \mathcal{S}' \triangleright A' \qquad \mathrm{B}\ (\mu) \cap \mathrm{F}\ (B) = \emptyset}{\mathcal{S} \triangleright A \parallel B \xrightarrow{\mu} \mathcal{S}' \triangleright A' \parallel B}$$

(LTS-R)

$$\frac{\mathcal{S} \triangleright r\{\!\!|\, P \mid !P \,|\!\!\}_\rho \xrightarrow{\mu} \mathcal{S}' \triangleright A}{\mathcal{S} \triangleright r\{\!\!|\, !P \,|\!\!\}_\rho \xrightarrow{\mu} \mathcal{S}' \triangleright A}$$

(LTS-E)

$$\frac{\mathcal{S} \triangleright (\nu a^r\!:\!R) r\{\!\!|\, P \,|\!\!\}_\rho \xrightarrow{\mu} \mathcal{S}' \triangleright A \qquad a \neq r}{\mathcal{S} \triangleright r\{\!\!|\, (\nu a\!:\!R)P \,|\!\!\}_\rho \xrightarrow{\mu} \mathcal{S}' \triangleright A}$$

(LTS-E)

$$\frac{\mathcal{S} \triangleright r\{\!\!|\, P \,|\!\!\}_\rho \xrightarrow{\mu} \mathcal{S}' \triangleright A}{\mathcal{S} \triangleright r\{\!\!|\, [n = n]P \,|\!\!\}_\rho \xrightarrow{\mu} \mathcal{S}' \triangleright A}$$

(LTS-S)

$$\frac{\mathcal{S} \triangleright r\{\!\!|\, P \,|\!\!\}_\rho \parallel r\{\!\!|\, Q \,|\!\!\}_\rho \xrightarrow{\mu} \mathcal{S}' \triangleright A}{\mathcal{S} \triangleright r\{\!\!|\, P \mid Q \,|\!\!\}_\rho \xrightarrow{\mu} \mathcal{S}' \triangleright A}$$

plus the symmetric version of rules of (LTS-P), (LTS-C) and (LTS-C)

Table 8: Inference Rules for the Labelled Transition System

We now state and prove some properties of $\approx$. First, we consider the congruence properties of $\approx$; then, we prove that it is a sound proof technique for barbed congruence.

Theorem B.2 (Congruence Properties of $\approx$). *The following facts hold.*

1. *If* $\mathcal{S}_1 \triangleright A_1 \approx \mathcal{S}_2 \triangleright A_2$ *and* $\mathcal{S}_1 \triangleright B \approx \mathcal{S}_2 \triangleright B$*, then* $\mathcal{S}_1 \triangleright A_1 \parallel B \approx \mathcal{S}_2 \triangleright A_2 \parallel B$.
2. *If* $(\mathcal{U}_1 \uplus \{a^r\!:\!R\};\mathcal{P}_1) \triangleright A_1 \approx (\mathcal{U}_2 \uplus \{a^r\!:\!R\};\mathcal{P}_2) \triangleright A_2$*, then* $(\mathcal{U}_1;\mathcal{P}_1) \triangleright (\nu a^r\!:\!R)A_1 \approx (\mathcal{U}_2;\mathcal{P}_2) \triangleright (\nu a^r\!:\!R)A_2$.

Proof. Both the clauses of the theorem are proved once shown that relation

$$\begin{aligned}\mathfrak{R} \quad \triangleq \quad \{ & ((\mathcal{U}_1;\mathcal{P}_1) \triangleright (\nu\,\widetilde{a^r} : \widetilde{R})(A_1 \parallel B)\,,\; (\mathcal{U}_2;\mathcal{P}_2) \triangleright (\nu\,\widetilde{a^r} : \widetilde{R})(A_2 \parallel B)) \;: \\ & (\mathcal{U}_1 \uplus \widetilde{a^r} : \widetilde{R};\mathcal{P}_1) \triangleright A_1 \approx (\mathcal{U}_2 \uplus \widetilde{a^r} : \widetilde{R};\mathcal{P}_2) \triangleright A_2 \;\wedge \\ & (\mathcal{U}_1 \uplus \widetilde{a^r} : \widetilde{R};\mathcal{P}_1) \triangleright B \approx (\mathcal{U}_2 \uplus \widetilde{a^r} : \widetilde{R};\mathcal{P}_2) \triangleright B \,\}\end{aligned}$$

is a bisimulation. □

To prove that $\approx$ is a sound proof technique for $\cong$, we must only consider well-typed configurations, i.e. configurations $\mathcal{S} \triangleright A$ such that A is well-typed in $\mathcal{S}$. Indeed, as already said, ill-typed systems are not considered in the definition of barbed congruence. Given a typing environment Γ, we let $\mathcal{U}_\Gamma$ be the rôles-to-users assignment extracted from Γ, that is the least assignment such that, for any association $r : \rho[\widetilde{a} : \widetilde{R(T)}]$ in Γ, it holds that $\mathcal{U}_\Gamma(r) = \rho$ and $\mathcal{U}_\Gamma(a^r) = R$, for any $a : R(T) \in \widetilde{a} : \widetilde{R(T)}$. For notational convenience, we write $\mathcal{S} \triangleright A \approx \mathcal{S} \triangleright B$ as $\mathcal{S} \triangleright A \approx B$.

The proof relies on the following lemma.

Lemma B.3 (Weakening for $\approx$). *If $(\mathcal{U};\mathcal{P}) \triangleright A \approx B$ and A and B are well-typed in $(\mathcal{U};\mathcal{P})$, then $(\mathcal{U} \uplus \mathcal{U}';\mathcal{P} \cup \mathcal{P}') \triangleright A \approx B$ for all $\mathcal{U}'$ and $\mathcal{P}'$.*

Proof. We have to prove that the relation

$$\mathfrak{R} \;\triangleq\; \{((\mathcal{U} \uplus \mathcal{U}';\mathcal{P} \cup \mathcal{P}') \triangleright A\,,\; (\mathcal{U} \uplus \mathcal{U}';\mathcal{P} \cup \mathcal{P}') \triangleright B) \;:\; (\mathcal{U};\mathcal{P}) \triangleright A \approx B\}$$

is a bisimulation. □

Theorem B.4 (Soundness of $\approx$). *Let $\mathcal{S} = (\mathcal{U};\mathcal{P})$, $\Gamma \vdash^{\mathcal{S}} A$ and $\Gamma \vdash^{\mathcal{S}} B$. If $(\mathcal{U}_\Gamma;\mathcal{P}) \triangleright A \approx B$, then $\Gamma \models^{\mathcal{S}} A \cong B$.*

Proof. It suffices to prove that the relation

$$\mathfrak{R} \;\triangleq\; \{\,\Gamma \models^{\mathcal{S}} (A, B) \;:\; \Gamma \vdash^{\mathcal{S}} A \;\wedge\; \Gamma \vdash^{\mathcal{S}} B \;\wedge\; (\mathcal{U}_\Gamma;\mathcal{P}) \triangleright A \approx B\}$$

is barb preserving, reduction closed and contextual.

1. Let $A \downarrow a^r$. By Definition 3.1 and well-typedness, it holds that $(\mathcal{U}_\Gamma;\mathcal{P}) \triangleright A$ may perform an input from a^r; then, $(\mathcal{U}_\Gamma;\mathcal{P}) \triangleright B$ may perform an input from a^r, possibly together with some τ-steps. Then, by Proposition B.1 and Definition 3.1, it is easy to prove that $B \Downarrow a^r$. The case for $A \downarrow \overline{a}^r$ is similar.

2. Let $A \longmapsto A'$. By Proposition B.1 and well-typedness, this implies that $(\mathcal{U}_\Gamma;\mathcal{P}) \triangleright A \xrightarrow{\tau} (\mathcal{U}_\Gamma;\mathcal{P}) \triangleright A'$. Thus, $(\mathcal{U}_\Gamma;\mathcal{P}) \triangleright B \Rightarrow (\mathcal{U}_\Gamma;\mathcal{P}) \triangleright B'$ and $(\mathcal{U}_\Gamma;\mathcal{P}) \triangleright A' \approx B'$. Again by Proposition B.1, $B \longmapsto^* B'$ and $\Gamma \models^{\mathcal{P}} A' \,\mathfrak{R}\, B'$. Indeed, by Theorem 2.1, it holds that $\Gamma \vdash^{\mathcal{P}} A'$ and $\Gamma \vdash^{\mathcal{P}} B'$. Moreover, it is easy to prove that $\approx$ is an equivalence relation and that it contains all the configurations formed with structurally equivalent systems; so, we have that $(\mathcal{U}_\Gamma;\mathcal{P}) \triangleright A' \approx A''$ and, thus, $(\mathcal{U}_\Gamma;\mathcal{P}) \triangleright A' \approx B'$.

3. We pick up $\Gamma \models^{\mathcal{S}} A \,\mathfrak{R}\, B$ and analyse the three clauses defining the contextuality property.

(a) Let $\mathcal{P}'$ be a permissions-to-rôles assignment and $\widetilde{n} : \widetilde{T}$ be such that $\widetilde{n} \cap dom(\mathcal{U}) = \emptyset$; since Γ respects $\mathcal{U}$, this implies that $\Gamma, \widetilde{n} : \widetilde{T}$ is defined. By Lemma A.2, we know that $\Gamma, \widetilde{n} : \widetilde{T} \vdash^{(\mathcal{U} \uplus \widetilde{n}:\widetilde{T}; \mathcal{P} \cup \mathcal{P}')} A$ and $\Gamma, \widetilde{n} : \widetilde{T} \vdash^{(\mathcal{U} \uplus \widetilde{n}:\widetilde{T}; \mathcal{P} \cup \mathcal{P}')} B$. Moreover, we let $\mathcal{U}_{\widetilde{n}:\widetilde{T}}$ to be the rôles-to-users assignment such that $\mathcal{U}_{\widetilde{n}:\widetilde{T}}(r) = \rho$, whenever $r : \rho[\widetilde{a} : \widetilde{C}] \in \widetilde{n} : \widetilde{T}$, and $\mathcal{U}_{\widetilde{n}:\widetilde{T}}(a^r) = R$, whenever $a^r : R(T) \in \widetilde{n} : \widetilde{T}$. It is easy to check that $\mathcal{U}_{\Gamma, \widetilde{n}:\widetilde{T}} = \mathcal{U}_\Gamma \uplus \mathcal{U}_{\widetilde{n}:\widetilde{T}}$ (indeed, $\Gamma, \widetilde{n} : \widetilde{T}$ is defined if and only if names in $\widetilde{n}$ do not occur in the domain of Γ; thus, $\mathcal{U}_\Gamma$ and $\mathcal{U}_{\widetilde{n}:\widetilde{T}}$ have disjoint domains, and their union coincides with $\mathcal{U}_{\Gamma, \widetilde{n}:\widetilde{T}}$). Thus, by Lemma B.3, $(\mathcal{U}_{\Gamma, \widetilde{n}:\widetilde{T}}; \mathcal{P} \cup \mathcal{P}') \triangleright A \approx B$. This suffices to conclude that $\Gamma, \widetilde{n} : \widetilde{T} \models^{(\mathcal{U} \cup \widetilde{n}:\widetilde{T}; \mathcal{P} \cup \mathcal{P}')} A \ \mathfrak{R}\ B$.

(b) Let $\bar{A}$ be a system such that $\Gamma \vdash^{\mathcal{S}} \bar{A}$. By Theorem B.2(1), we can state that $(\mathcal{U}_\Gamma; \mathcal{P}) \triangleright A \parallel \bar{A} \approx B \parallel \bar{A}$. Moreover, by rule (T-S P), it holds that $\Gamma \vdash^{\mathcal{S}} A \parallel \bar{A}$ and $\Gamma \vdash^{\mathcal{S}} B \parallel \bar{A}$. Thus, $\Gamma \models^{\mathcal{S}} A \parallel \bar{A} \ \mathfrak{R}\ B \parallel \bar{A}$, as required.

(c) Let $\Gamma = \Gamma', a^r : R(T)$ and $\mathcal{U} = \mathcal{U}' \uplus \{a^r : R(T)\}$. It is easy to check that $\mathcal{U}_\Gamma = \mathcal{U}_{\Gamma'} \uplus a^r{:}R$ and, thus, $(\mathcal{U}_{\Gamma'} \uplus a^r{:}R; \mathcal{P}) \triangleright A \approx B$. By Theorem B.2(2), this implies that $(\mathcal{U}_{\Gamma'}; \mathcal{P}) \triangleright (\nu a^r : R)A \approx (\nu a^r : R)B$; moreover, by rule (T-S R), $\Gamma' \vdash^{(\mathcal{U}'; \mathcal{P})} (\nu a^r : R)A$ and $\Gamma' \vdash^{(\mathcal{U}'; \mathcal{P})} (\nu a^r : R)B$. Thus, $\Gamma' \models^{(\mathcal{U}'; \mathcal{P})} (\nu a^r : R)A \ \mathfrak{R}\ (\nu a^r : R)B$, as required. □

We remark that $\approx$ is used as a proof-technique for barbed congruence. Indeed, while the former is easy to use, the latter is very hard to handle because of the contextual closure requirement. This suffices for our purpose in the present paper, whose intention is to present the calculus and lay out its main properties. Nevertheless, for theoretical reasons, it is often important to know whether $\approx$ is a complete characterisation of $\cong$. This is a laborious question to answer. We leave it as future work to follow well-known paths towards the answer (as, e.g. [12, 18]) to prove the converse of Theorem B.4, i.e. that bisimilarity is complete for barbed congruence.

To conclude, we now briefly discuss some possible use of the bisimulation, apart from proving barbed congruence. Mainly, its distinctive features are the possibility of relating ill-typed systems and/or systems under different schemata. For example, by letting α to range over action prefixes (i.e. inputs/outputs and **role/yield**), it holds that

$$\mathcal{S} \triangleright r\{\!\!\{ \alpha.P \}\!\!\}_\rho \approx \mathbf{0}$$

whenever α is not legal for a session $r\{\!\!\{ \cdot \}\!\!\}_\rho$ with respect to $\mathcal{S}$, that is, if the premises of rules (LTS-Rˆ), (LTS-Y), (LTS-K-I), (LTS-F-I) and (LTS-O) are not satisfied. This law stresses that LTS and types both enforce the same requirements (compare the run-time checks of the LTS with the run-time errors in Table 5 and, consequently, the results in Proposition B.1). As a consequence, the following law differentiates our language from the π-calculus. Indeed, it holds that

$$\mathcal{S} \triangleright (\nu a^r : R)(r\{\!\!\{ a(x).P \}\!\!\}_\rho \parallel s\{\!\!\{ a^r\langle n \rangle.Q \}\!\!\}_{\rho'}) \approx \mathbf{0}$$

whenever $R^? \notin \mathcal{P}(\rho)$ or $R^! \notin \mathcal{P}(\rho')$.

Finally, we can use the bisimulation to find a 'minimal schema' to run a given system without altering its functionalities. Let A be a system well-typed in a RBAC

schema $\mathcal{S}$. Potentially, there are many schemata under which the system can run correctly; thus, it seems reasonable to look for a 'minimal' such. According to the metrics chosen, several properties can be associated to this element. For example, if the metrics is the size of the schema (seen as a pair of sets), the minimal element would be one of the smallest; thus, its storage and handling would be cheaper. We now define the set of configurations whose second component is A as follows:

$$CONF_A = \{\mathcal{S}' \triangleright A \ : \ \mathcal{S}' \text{ is a RBAC schema}\}$$

We now partition $CONF_A$ with respect to $\approx$ and consider the equivalence class containing $\mathcal{S} \triangleright A$, called $CONF_A^{\mathcal{S}}$. By fixing a metrics over schemata, the minimal schema to run the system A will be a minimal element of $CONF_A^{\mathcal{S}}$. Indeed, such an element behaves like $\mathcal{S} \triangleright A$, because they both belong to the same equivalence class $CONF_A^{\mathcal{S}}$, but its schema is smaller, as its is a minimal element of $CONF_A^{\mathcal{S}}$. Clearly, the existence of such a minimal element and the way in which it is chosen depend on the chosen metrics. Possible metrics could be based on the memory required to store the schema, on the number of rôles used to define the schema, on the weight of the permissions associated with some users (once assumed a weight function to discriminate sensible permissions from common ones), on the average number of permissions associated with each rôle, and so on.

References

[1] G.-J. Ahn and R. Sandhu. Rôle-based authorisation constraints specification. *ACM Transactions on Information and System Security*, 3(4):207–226, 2000.

[2] R. Amadio and L. Cardelli. Subtyping recursive types. *ACM Transactions on Programming Languages and Systems*, 15(4):575–631, 1993.

[3] E. Bertino, B. Catania, E. Ferrari, and P. Perlasca. A logical framework for reasoning about access control models. In *Proc. of 6th SACMAT*, pages 41–52. ACM Press, 2001.

[4] C. Braghin, D. Gorla, and V. Sassone. A distributed calculus for rôle-based access control. In *Proc. of 17th Computer Security Foundations Workshop (CSFW'04)*, pages 48–60. IEEE Computer Society, 2004.

[5] C. Braghin, D. Gorla, and V. Sassone. A distributed calculus for rôle-based access control. Technical Report 08/2004, Dip. di Informatica, Univ. di Roma "La Sapienza", 2004.

[6] L. Cardelli, G. Ghelli, and A. D. Gordon. Secrecy and group creation. *Information and Computation*, 196(2):127–155, 2005.

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

[8] F. Cardone and M. Coppo. Two extensions of Curry's type inference system. In *Logic and Computer Science*, pages 19–75. Academic Press, 1990.

[9] R. De Nicola, G. Ferrari, and R. Pugliese. K : a Kernel Language for Agents Interaction and Mobility. *IEEE Transactions on Software Engineering*, 24(5):315–330, 1998.

[10] D. Ferraiolo and D. Kuhn. Rôle-based access control. In *Proc. of the NIST-NSA National Computer Security Conference*, pages 554–563, 1992.

[11] D. Ferraiolo, R. Sandhu, S. Gavrila, D. Kuhn, and R. Chandramouli. Proposed NIST standard for rôle-based access control. *ACM Transactions on Information and System Security*, 4(3):224–274, 2001.

[12] C. Fournet and C. Laneve. Bisimulations for the join-calculus. *Theoretical Computer Science*, 266(1-2):569–603, 2001.

[13] M. Hennessy and J. Rathke. Typed behavioural equivalences for processes in the presence of subtyping. *Mathematical Structures in Computer Science*, 14(5):651–684, 2004.

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

[15] K. Honda and N. Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 152(2):437–486, 1995.

[16] M. Koch, L. Mancini, and F. Parisi-Presicce. A formal model for rôle-based access control using graph transformation. In *Proc. of 5th ESORICS*, volume 1895 of *LNCS*, pages 122–139. Springer, 2000.

[17] M. Koch, L. Mancini, and F. Parisi-Presicce. Decidability of safety in graph-based models for access control. In *Proc. of 7th ESORICS*, volume 2502 of *LNCS*, pages 229–243. Springer, 2002.

[18] M. Merro and D. Sangiorgi. On asynchrony in name-passing calculi. *Mathematical Structures in Computer Science*, 14(5):715–767, 2004.

[19] R. Milner. The polyadic π-calculus: a tutorial. In *Logic and Algebra of Specification*, volume 94 of *Series F: Computer and System Sciences*. NATO Advanced Study Inst., 1993.

[20] B. C. Pierce. *Types and programming languages*. MIT Press, 2002.

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

[22] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[23] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Rôle-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[24] D. Sangiorgi and D. Walker. *The π-calculus: a theory of mobile processes*. Cambridge University Press, 2001.

[25] A. Schaad and J. Moffett. A lightweight approach to specification and analysis of rôle-based access control extensions. In *Proc. of 7th SACMAT*, pages 13–22. ACM Press, 2002.

[26] F. B. Schneider. Least privilege and more. *Security and Privacy*, 1(3):55–59, 2003.