Journal of Computer Security 13 (2005) 3–47
IOS Press

Relating multiset rewriting and process algebras for security protocol analysis

Stefano Bistarelli [a,b], Iliano Cervesato [c], Gabriele Lenzini [d,e]
and Fabio Martinelli [a]

[a] *Istituto di Informatica e Telematica – CNR, Via G. Moruzzi 1, I-56100 Pisa, Italy*
E-mail: {stefano.bistarelli,fabio.martinelli}@iit.cnr.it
[b] *Dipartimento di Scienze, Università "D'Annunzio" di Chieti-Pescara, Viale Pindaro 87,*
65127 Pescara, Italy
E-mail: bista@sci.unich.it
[c] *Advanced Engineering and Science Division, ITT Industries Inc., Alexandria, VA 22303, USA*
E-mail: iliano@itd.nrl.navy.mil
[d] *Istituto di Scienza e Tecnologie dell'Informazione – CNR, Via G. Moruzzi 1, I-56100 Pisa, Italy*
E-mail: gabriele.lenzini@isti.cnr.it
[e] *Department of Computer Science, University of Twente, 7500 AE Enschede, The Netherlands*

When formalizing security protocols, different specification languages support very different reasoning methodologies, whose results are not directly or easily comparable. Therefore, establishing clear mappings among different frameworks is highly desirable, as it permits various methodologies to cooperate by interpreting theoretical and practical results of one system into another. In this paper, we examine the relationship between two general verification frameworks: multiset rewriting (MSR) and a process algebra (PA) inspired to CCS and the π-calculus. Although defining a simple and general bijection between MSR and PA appears difficult, we show that the sublanguages needed to specify cryptographic protocols admit an effective translation that is not only trace-preserving, but also induces a correspondence relation between the two languages. In particular, the correspondence sketched in this paper permits transferring several important trace-based properties such as secrecy and many forms of authentication.

1. Introduction

In the last decade, security-related problems have attracted the attention of many researchers from several different communities, especially formal methods (e.g., [1,3,7,9,11,14,18–21,23,28,36]). These researchers have often let their investigation be guided by the techniques and experiences specific to their own areas of knowledge. While on the one hand furthering research, this massive interest has on the other hand determined a plethora of results that often are not directly comparable or integrable with one another. In the last few years, attempts have been made to unify frameworks for specifying security properties often expressed in different ways [22], and to study the relationships between different models for representing security protocols [10].

In this paper, we relate transition-based and a form of process-based models for the description and the analysis of a large class of security protocols. We choose the multiset-rewriting formalism MSR as a representative of the former, and synthesize salient features of popular process algebras in a system that we call PA as an abstraction of the latter.

MSR, with its roots in concurrency theory and rewriting logic, has proved to be a language of choice for studying foundational issues in security protocols [9]. It is also playing a practical role through the closely related CIL intermediate language [14] of the CASPL security protocol analysis system [13], in particular since translators from several tools to CIL have been developed. For these reasons, MSR has become a central point when comparing languages for protocol specification. For example, ties between MSR and strand spaces [17], a popular specification language for crypto-protocols, have been analyzed in [10].

Process algebra encompasses a family of well-known formal frameworks proposed to describe features of distributed and concurrent systems. Here we use an instance, PA, that borrows concepts from different calculi, specifically CCS [30] and the π-calculus [31]. We expect our results to adapt to other (value passing) process algebras used for security protocol analysis, e.g., the *spi*-calculus [2] or CSP [35]. Indeed, when applied to security protocol analysis, most such languages rely only on a well-identified subset of primitives, that we have isolated in the language considered here.

We relate MSR and PA by defining *encodings* from one formalism to the other. Moreover we propose a *correspondence relation* among MSR and PA protocol models, preserved by our encodings, that is sufficient to transfer several useful trace-based properties such as secrecy and many forms of authentication. Informally it says that an MSR configuration and a PA process correspond if and only if the messages lying on the network and the messages known by the intruder are the same, step by step, in the two models.

The results in this paper yield several important consequences:

- First, our encodings establish a firm relationship between the *specification methodologies* underlying MSR and PA. MSR epitomizes a representation paradigm based on transitions between explicit states, as found, for example, in the vast majority of tools for security protocol analysis [9,11,13,16,28,34,35]. The approach underlying PA and the languages behind it, e.g., [2,6,18,20,23], represents concurrent systems, with security protocols as a particular instance, as independent threads of computation communicating through message passing. While specifications are obviously related, moving between paradigms is an error-prone proposition unless guided by formal encodings.
- Second, the relationship we developed helps relate verification results obtainable in each model, in particular as far as secrecy and authentication are concerned. Systems *à la* MSR overwhelmingly embrace a verification methodology based on some form of trace exploration: model-checking [11,13,16,35],

theorem proving [34], or a combination [28]. The situation is more complex in process-algebraic languages, which sometimes base their analysis on traces [6,20,37], but also on process equivalence [2], type-checking [23] and other forms of symbolic reasoning [24]. While we do not study what these last three forms of analysis map to in the MSR world, we believe that the present work opens the door to such an investigation. Authentication and secrecy are quintessential trace-based safety properties (they are expressed in terms of intruder knowledge and messages passed onto the network and our encodings preserve this information). Therefore relating trace-based results in MSR and PA is valuable, in particular as these languages rely on different notions of traces, and sometimes make different uses of them, e.g., [20].

- Finally, by bridging PA and MSR item we implicitly define a correspondence between PA and other languages for security analysis. MSR has already been related to other formalisms, such as strand spaces [17] in a setting with an interleaving semantics (a worthy investigation as remarked in [12]), while work about linear logic and MSR appears in [29].

The rest of the paper is organized as follows. Section 2 recalls the multiset rewriting and process algebra frameworks and in Section 3 their use in the specification of security protocols. Section 4 presents the encodings from multiset rewriting to process algebra (Section 4.1), and vice versa (Section 4.2). Section 5 defines the notion of equivalence motivating the encodings, while Section 6 shows how security properties are preserved when going from PA to MSR and vice versa via our encodings. Section 7 concludes with some final remarks.

2. Background

In this section, we recall the syntax and formal semantics of multiset rewriting (MSR) and we define the language, PA, that we will use as a representative of process algebras. Before doing so, we present our notation for tuples, as both MSR and PA rely on these objects. A *tuple* is defined by the following grammar:

$$\boldsymbol{t} ::= \epsilon \mid t;\boldsymbol{t}$$

A tuple $\boldsymbol{t}$ is a sequence of items. We use the semicolon (";") as the tuple constructor: it is associative but not commutative. We write ϵ for the empty tuple, which acts as the left and right identity of ";". We write $t \in \boldsymbol{t}$ to indicate that item t is present in tuple $\boldsymbol{t}$, and use the notation $\boldsymbol{t}' \sqsubseteq \boldsymbol{t}$ to indicate that $\boldsymbol{t}'$ is a subsequence of $\boldsymbol{t}$, i.e., that $\boldsymbol{t}'$ can be obtained by deleting zero or more symbols from $\boldsymbol{t}$. Finally, given tuples $\boldsymbol{t}$ and $\boldsymbol{t}'$ with $\boldsymbol{t}' \sqsubseteq \boldsymbol{t}$, we write $\boldsymbol{t} - \boldsymbol{t}'$ for the tuple obtained by filtering out all items $t' \in \boldsymbol{t}'$ from $\boldsymbol{t}$, while preserving the order of the remaining elements of the latter.

2.1. First order multiset rewriting

The language of first-order MSR is defined by the following grammar:

Elements	$\tilde{a} ::= \cdot \mid a(\boldsymbol{t}), \tilde{a}$
Rewriting Rules	$r ::= \tilde{a}(\boldsymbol{x}) \rightarrow \exists \boldsymbol{n}.\tilde{b}(\boldsymbol{x}; \boldsymbol{n})$
Rule sets	$\tilde{r} ::= \cdot \mid r, \tilde{r}$

Multiset elements are chosen as atomic formulas $a(\boldsymbol{t})$, where $\boldsymbol{t}$ is a tuple of terms over some first-order signature Σ. We write $\tilde{a}(\boldsymbol{x})$ to emphasize that variables, drawn from $\boldsymbol{x}$, appear in a multiset $\tilde{a}$. Similarly we write t (resp., $\boldsymbol{t}$) as $t(\boldsymbol{x})$ (resp., $\boldsymbol{t}(\boldsymbol{x})$), to underline that variables $\boldsymbol{x}$ appear in a term t (resp., in the tuple of terms $\boldsymbol{t}$). Instead, we write $\underline{t}$ (resp., $\underline{\boldsymbol{t}}$) to emphasize, when required, that a term t is (resp., all the term in $\boldsymbol{t}$ are) ground, i.e., variable-free.

In the sequel, the comma "," will denote multiset union and will implicitly be considered commutative and associative, while "$\cdot$", the empty multiset, will act as a neutral element; we will omit it when convenient. The operational semantics of MSR is expressed by the following two judgments:

Single rule application	$\tilde{r} : \tilde{a} \longrightarrow \tilde{b}$
Iterated rule application	$\tilde{r} : \tilde{a} \longrightarrow^* \tilde{b}$

The multisets $\tilde{a}$ and $\tilde{b}$ are called *states* and are always ground formulas. The arrow represents a transition. These judgments are defined as follows:

$$\frac{}{(\tilde{r}, \tilde{a}(\boldsymbol{x}) \rightarrow \exists \boldsymbol{n}.\tilde{b}(\boldsymbol{x}; \boldsymbol{n})) : (\tilde{c}, \tilde{a}[\underline{\boldsymbol{t}}/\boldsymbol{x}]) \longrightarrow (\tilde{c}, \tilde{b}[\underline{\boldsymbol{t}}/\boldsymbol{x}, \underline{\boldsymbol{k}}/\boldsymbol{n}])}\ \mathbf{msr_0}$$

$$\frac{}{\tilde{r} : \tilde{a} \longrightarrow^* \tilde{a}}\ \mathbf{msr_*} \qquad \frac{\tilde{r} : \tilde{a} \longrightarrow \tilde{b} \qquad \tilde{r} : \tilde{b} \longrightarrow^* \tilde{c}}{\tilde{r} : \tilde{a} \longrightarrow^* \tilde{c}}\ \mathbf{msr_1}$$

The first inference shows how a rewrite rule $r = \tilde{a}(\boldsymbol{x}) \rightarrow \exists \boldsymbol{n}.\tilde{b}(\boldsymbol{x}; \boldsymbol{n})$ is used to transform a state into a successor state: it identifies a ground instance $\tilde{a}(\underline{\boldsymbol{t}})$ of its antecedent and replaces it with the ground instance $\tilde{b}(\underline{\boldsymbol{t}}; \underline{\boldsymbol{k}})$ of its consequent, where $\underline{\boldsymbol{k}}$ are fresh constants. Here $[\underline{\boldsymbol{t}}/\boldsymbol{x}]$ denotes the substitution (also written θ) replacing every occurrence of a variable x among $\boldsymbol{x}$ with the corresponding term t in $\boldsymbol{t}$. These rules implement a non-deterministic but sequential computation model. This means that in general several rules are applicable at any step but only one rule, chosen non-deterministically among them, is applied at each step. Concurrency is captured as the permutability of (some) rule applications. The remaining rules define $\longrightarrow^*$ as the reflexive and transitive closure of $\longrightarrow$.

2.2. Process algebras

Process algebraic specifications of security protocols are generally limited to the parallel composition of a number of processes describing the sequence of actions performed by each agent. With this in mind, we forsake the full treatment of a traditional process algebra, such as the π-calculus, in favor of a more specific language, PA, that includes the features commonly used for describing cryptographic protocols. In particular, we lay out PA on two levels: *sequential processes* describe the sequence of atomic actions (input, output, name generation, etc.) performed by an individual agent and *parallel processes* bundle them into a multi-agent specifications. Sequential processes are synchronous, although a systematic use of buffer processes will prevent the possibility of blocking on an output action. For convenience, we will rely on polyadic communication channels.

With these premises, the language of PA is defined by the following grammar:

$$\begin{array}{lllllllllll}
\textit{Parallel processes} & Q ::= 0 & \mid & Q \parallel P & \mid & Q \parallel !P & & & & & \\
\textit{Sequential processes} & P ::= 0 & \mid & \overline{a}(\boldsymbol{t}).P & \mid & a(\boldsymbol{x}).P & \mid & [\boldsymbol{x} = \boldsymbol{t}]\, P & \mid & \nu x.P &
\end{array}$$

Parallel processes are defined as a parallel composition of, possibly replicated, sequential processes. These, in turn, are a sequence of communication actions (input or output), pattern matching and constant generation. An output process $\overline{a}(\boldsymbol{t}).P$ is ready to send a tuple of terms $\boldsymbol{t}$, each built over a signature Σ, along the polyadic channel named a. An input process $a(\boldsymbol{x}).P$ is ready to receive a tuple of (ground) messages, each in the corresponding variable $x \in \boldsymbol{x}$. The process $[\boldsymbol{x} = \boldsymbol{t}]\, P$ is a parallel pattern matching construct which forces any instantiation of $\boldsymbol{x}$ to match the pattern $\boldsymbol{t}$, possibly binding previously unbound variables in the latter. Finally, the creation of a new object in P (as in the π-calculus [32]) is written as $\nu x.P$ (we will sometimes abbreviate $\nu x_1 . \ldots . \nu x_n.P$ as $\nu \boldsymbol{x}.P$). The binders of our language are $\nu \boldsymbol{x}$, $a(\boldsymbol{x})$ which bind each x in $\boldsymbol{x}$, and $[\boldsymbol{x} = \boldsymbol{t}]$ which binds any first occurrence of a variable in $\boldsymbol{t}$. This induces the usual definition of free and bound variables in a term or process.

The operational semantics of PA is given by the following judgments:

$$\begin{array}{ll}
\textit{Single interaction} & Q \Rightarrow Q' \\
\textit{Iterated interaction} & Q \Rightarrow^* Q'
\end{array}$$

They are defined as follows:

$$\frac{}{(Q \parallel \overline{a}(\underline{\boldsymbol{t}}).P \parallel a(\boldsymbol{x}).P') \Rightarrow (Q \parallel P \parallel P'[\underline{\boldsymbol{t}}/\boldsymbol{x}])}\ \mathbf{pa_0}$$

$$\frac{\underline{\boldsymbol{t}} = \boldsymbol{t'}[\theta]}{(Q \parallel [\underline{\boldsymbol{t}} = \boldsymbol{t'}]\, P) \Rightarrow (Q \parallel P[\theta])}\ \mathbf{pa_{[]}} \qquad \frac{\underline{k} \notin c(Q) \cup c(P)}{(Q \parallel \nu x.P) \Rightarrow (Q \parallel P[\underline{k}/x])}\ \mathbf{pa_\nu}$$

$$\frac{P \equiv P' \quad P' \Rightarrow Q' \quad Q' \equiv Q}{P \Rightarrow Q}\,\mathbf{pa}_{\equiv} \qquad \frac{}{Q \Rightarrow^{*} Q}\,\mathbf{pa}_{*}$$

$$\frac{Q \Rightarrow Q'' \quad Q'' \Rightarrow^{*} Q'}{Q \Rightarrow^{*} Q'}\,\mathbf{pa_1}$$

The first inference (*reaction*) shows how two sequential processes, respectively one ready to perform an output of a tuple $\underline{\boldsymbol{t}}$ of ground terms, and one ready to perform an input over $\boldsymbol{x}$ react by applying the instantiating substitution $[\underline{\boldsymbol{t}}/\boldsymbol{x}]$ to P'. The second inference rule (*matching*) says that there must exist a substitution θ that matches terms $\boldsymbol{t'}$ with ground terms $\underline{\boldsymbol{t}}$, for $[\underline{\boldsymbol{t}} = \boldsymbol{t'}]P$ to evolve into $P[\theta]$. The third rule defines the semantics of νx as instantiation with a fresh constant i.e., a name which differs form those appearing in all the process terms (here $c(P)$ denotes the set of constant in P). The next rule allows interactions to happen modulo structural equivalence $\equiv$, that in our case contains the usual monoidal equalities of parallel processes with respect to $\|$ and 0, the unfolding of replication (i.e., $!P \equiv\, !P \parallel P$), and the equation $[\boldsymbol{t} = \boldsymbol{t'}]\, P \equiv [\boldsymbol{t}^{*} = \boldsymbol{t'}^{*}]\, P$ which filter out identities in tuple's matching, i.e., where $\boldsymbol{t}^{*}$ and $\boldsymbol{t'}^{*}$ are obtained from $\boldsymbol{t}$ and $\boldsymbol{t'}$ by removing all identical items that lay at the same position. Finally, the last two inferences define $\Rightarrow^{*}$ as the reflexive and transitive closure of $\Rightarrow$.

3. Security protocols

A cryptographic protocol is a collection of distributed programs supporting communication between participating agents and aimed at achieving predetermined security outcomes such as secrecy or authentication. The agents communicating in a protocol are called *principals*, while the individual programs they execute as part of the protocol are called *roles*. Communication happens through a public *network* and is therefore accessible to anyone, unless protected through cryptography.

Both transition- and process-based languages have been widely used for the specification of cryptographic protocols (see for example [1,3,9,11,14,18–21,23,28,36]). In this section, we define MSR$_P$ and PA$_P$, two security-oriented instances of MSR and PA respectively, and describe how they can be used to specify security protocols.

Narrowing our investigation to a specific domain allows us to directly compare these restricted versions of PA and MSR. Moreover by bounding our analysis to cryptographic protocols, we are able to obtain stronger correspondence results than what seems achievable in a general comparison between PA and MSR [4].

The two specifications will rely on a common first-order signature Σ_P that includes at least concatenation ($\langle\_, \_\rangle$) and encryption ($\{\_\}\_$). In both formalisms, terms in Σ_P stand for messages. Predicate symbols are interpreted as such in MSR$_P$, and as channel names in PA$_P$. Variables will also be allowed in rules and processes.

3.1. Formalizing protocols as multiset rewriting

MSR$_P$ relies on the following predicate symbols [10]:

Network Messages ($\tilde{N}$): are the predicates used to model the network, where $N(t)$ means that the term t is lying on the network.

Role States ($\tilde{A}$): are the predicates used to model roles. Assuming a set of *role identifiers* R, the family of *role state predicates* $\{A_{\rho_i}(\boldsymbol{t}) : i = 0 \ldots l_\rho\}$, is intended to hold the internal state, $\boldsymbol{t}$, of a principal in role $\rho \in R$ during the sequence of protocol steps $i = 0 \ldots l_\rho$. The behavior of each role ρ is described through a finite number of rules, indexed from 0 to l_ρ.

Intruder ($\tilde{I}$): are the predicates used to model the intruder I, where $I(t)$, means that the intruder knows the message t.

Persistent Predicates ($\tilde{\pi}$): are ground predicates holding data that does not change during the unfolding of the protocol (e.g., $\mathtt{Kp}(K; K')$ indicates that K and K' form a pair of public/private keys). Rules use these predicates in a read-only manner to access the value of persistent data.

A security protocol is expressed in MSR$_P$ as a set of rewrite rules $\tilde{r}$ of a specific format called a *security protocol theory*. Given roles R, it can be partitioned as $\tilde{r} = \cup_{\rho \in R}(\tilde{r}_\rho), \tilde{r}_I$, where $\tilde{r}_\rho$ and $\tilde{r}_I$ describe the behavior of a role $\rho \in R$ and of the intruder I. For each role ρ, the rules in $\tilde{r}_\rho$ consist of:

- one *initial rule*

$$\textit{instantiation} \quad r_{\rho_0} : \quad \tilde{\pi}(\boldsymbol{x}) \quad \rightarrow \quad \exists \boldsymbol{n}. A_{\rho_0}(\boldsymbol{x}; \boldsymbol{n}), \tilde{\pi}(\boldsymbol{x})$$

- zero or more ($i = 1 \ldots l_\rho$) *message exchange rules*:

$$\begin{array}{llrcl} \textit{send} & r_{\rho_i} : & A_{\rho_{i-1}}(\boldsymbol{x}) & \rightarrow & A_{\rho_i}(\boldsymbol{x}), N(t(\boldsymbol{x})) \\ \textit{receive} & r_{\rho_i} : & A_{\rho_{i-1}}(\boldsymbol{x}), N(y) & \rightarrow & A_{\rho_i}(\boldsymbol{x}; y) \\ \textit{analysis} & r_{\rho_i} : & A_{\rho_{i-1}}(\boldsymbol{t}(\boldsymbol{x})) & \rightarrow & A_{\rho_i}(\boldsymbol{x}) \end{array}$$

The first rule (*instantiation*) describes the instantiation step of a protocol role. All the new names required in a role ρ are generated during instantiation, and similarly all the variables $\boldsymbol{x}$ referring to permanent data $\tilde{\pi}(\underline{\boldsymbol{t}})$ are bound to ground permanent terms in that rule. The second rule (*send*) describes an action of sending a message $\underline{t}$ composed by using (all or a subset of) the ground terms in the role's state. The third rule (*receive*) describes a receiving, where a message $\underline{t}$ lying in the net is retrieved, bound to variable y and then stored into the internal state of the role. The last rule (*analysis*) simulates the action of a role when it analyses (e.g., decrypts or splits) previously received messages.

This fairly explicit formulation of MSR rules will simplify our comparison with PA$_P$. Equivalent, but more succinct, formulations can be found in [8,9].

Rules in $\tilde{r}_I$ are the standard rules describing the intruder in the style of Dolev–Yao [15], whose capabilities consist in intercepting, analyzing, synthesizing and constructing messages, with the ability to access some permanent data. Formally:

$$\begin{array}{lrcl}
r_{I_1}: & \pi(x) & \rightarrow & I(x), \pi(x) \\
r_{I_2}: & \cdot & \rightarrow & \exists n. I(n) \\
r_{I_3}: & N(x) & \rightarrow & I(x) \\
r_{I_4}: & I(x) & \rightarrow & N(x), I(x) \\
r_{I_5}: & I(\langle x_1,\, x_2\rangle) & \rightarrow & I(x_1), I(x_2), I(\langle x_1,\, x_2\rangle) \\
r_{I_6}: & I(x_1), I(x_2) & \rightarrow & I(\langle x_1,\, x_2\rangle), I(x_1), I(x_2) \\
r_{I_7}: & I(\{x\}_k), I(k), \mathtt{Kp}(k; k') & \rightarrow & I(x), \mathtt{Kp}(k; k'), I(\{x\}_k), I(k) \\
r_{I_8}: & I(x), I(k) & \rightarrow & I(\{x\}_k), I(x), I(k) \\
r_{I_9}: & I(x) & \rightarrow & \cdot
\end{array}$$

where x, x_i's and k are variables. Informally, the first rule allows the intruder to access (i.e., get knowledge of) persistent data. In the second, rule the intruder creates a new ground datum. In the third, a message lying in the network is intercepted, while in the fourth a known message is injected into the network channel. The remaining rules describe the intruder capabilities for managing the messages it knows: more precisely its ability to decompose pairs, to compose pairs, to decrypt a message (if the relative decryption key is known), and to create encrypted messages. Finally, the last one describes the capability of the intruder in deleting messages (i.e., forgetting knowledge).

In MSR$_P$, a state is a multiset of the form $\tilde{s} = (\tilde{N}, \tilde{A}, \tilde{I}, \tilde{\pi})$, where the components collect ground facts of the form $N(t)$, $A_{\rho_i}(\boldsymbol{t})$, $I(t)$, and $\pi(\boldsymbol{t})$ respectively. An *initial state* $\tilde{s}_0 = (\tilde{I}_0, \tilde{\pi})$ contains only the initial intruder knowledge ($\tilde{I}_0$) and persistent predicates ($\tilde{\pi}$). Note that $\tilde{\pi}$ remains the same in every state. A pair ($\tilde{r} : \tilde{s}$) consisting of a protocol theory $\tilde{r}$ and a state $\tilde{s}$ is called a *configuration*. The initial configuration is ($\tilde{r} : \tilde{s}_0$).

Example 1. We make these definitions more concrete by showing the MSR$_P$ representation of the classical Needham-Schroeder Public Key (*NSPK*) protocol [33]. In the common informal notation, it is written as follows:

$$\begin{array}{l}
1.\ A \longrightarrow B : \{A, N_A\}_{K_B} \\
2.\ B \longrightarrow A : \{N_A, N_B\}_{K_A} \\
3.\ A \longrightarrow B : \{N_B\}_{K_B}
\end{array} \tag{1}$$

The abstract principal A and the role it executes are called the *initiator* since it originates the first message. Dually, B is the *responder*. This first message, $\{A, N_A\}_{K_B}$, consists of A's name and a freshly generated random value N_A

(a nonce), and is encrypted using B's public key K_B. Upon successfully decrypting this message (using private key K_B^{-1}), B replies with the second message, $\{N_A, N_B\}_{K_A}$, where N_B is a second nonce, generated by B. Upon successfully processing this message, A sends the final message $\{N_B\}_{K_B}$ which shall be interpreted by B.

Here, A and B perform distinct although related sequences of actions: A generates N_A, sends $\{A, N_A\}_{K_B}$, waits for a message from B and verifies that it matches the format $\{N_A, N_B\}_{K_A}$, and finally sends the third message, $\{N_B\}_{K_B}$. This sequence of actions constitute A's role. B's role is similar. Both MSR$_P$ and PA$_P$ give a role-centric representation of a protocol.

The MSR$_P$ specification of the *NSPK* protocol consists of the rule-set $\mathcal{R}_{\text{NSPK}}$ which we partition as $(\mathcal{R}_A, \mathcal{R}_B, \tilde{r}_I)$. $\mathcal{R}_A$ and $\mathcal{R}_B$ implement the roles of the initiator (A) and the responder (B) respectively, while $\tilde{r}_I$ describes the actions of a potential attacker, and have been fixed earlier in the discussion.

First some abbreviations. We define

$$\tilde{\pi}(x; y; k_x; k'_x, k_y) = \mathtt{Pr}(x), \mathtt{PrK}(x; k'_x), \mathtt{PbK}(y; k_y), \mathtt{Kp}(k_x; k'_x)$$

Here, persistent predicate $\mathtt{Pr}(x)$ indicates that x is the name of a principal; the predicate $\mathtt{PbK}(x; k_x)$ defines k_x to be the public key of principal x; the predicate $\mathtt{PrK}(x; k'_x)$ says that k'_x is x's private key; finally, $\mathtt{Kp}(k_x; k'_x)$ relates a public key k_x and the corresponding private key k'_x. Two tuples of variable $(a; b; k_a; k'_a; k_b)$ and $(b; a; k_b; k'_b; k_a)$ will occur repeatedly in this example; therefore we shall abbreviate them as $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively.

Then, the following rules describe A's role:

$$\mathcal{R}_A \begin{cases} r_{A_0} : \tilde{\pi}(\boldsymbol{A}) & \to \exists n_a.\tilde{\pi}(\boldsymbol{A}), A_0(\boldsymbol{A}; n_a) \\ r_{A_1} : A_0(\boldsymbol{A}; n_a) & \to N(\{a, n_a\}_{k_b}), A_1(\boldsymbol{A}; n_a) \\ r_{A_2} : A_1(\boldsymbol{A}; n_a), N(m) & \to A_2(\boldsymbol{A}; n_a; m) \\ r_{A_3} : A_2(\boldsymbol{A}; n_a; \{n_a, n_b\}_{k_a}) & \to A_3(\boldsymbol{A}; n_a; n_b) \\ r_{A_4} : A_3(\boldsymbol{A}; n_a; n_b) & \to N(\{n_b\}_{k_b}), A_4(\boldsymbol{A}; n_a; n_b) \end{cases}$$

The first rule r_{A_0} in $\mathcal{R}_A$ is the instantiation rule of this role, and takes care of generating the initiator's nonce, n_a and collecting the persistent information used in the role. Rules r_{A_1} and r_{A_4} are send rules corresponding to the message transmission step 1 and 3 in protocol (1). Rules r_{A_2} and r_{A_3} realize the initiator's actions in the second step of *NSPK*, namely the reception of a message m from b and the verification that it matches the expected pattern $\{n_a, n_b\}_{k_a}$.

The responder's role is similarly specified by the following MSR$_P$ rule set:

$$\mathcal{R}_B \begin{cases} r_{B_0} : \tilde{\pi}(\boldsymbol{B}) & \rightarrow \exists n_b.\tilde{\pi}(\boldsymbol{B}), B_0(\boldsymbol{B}; n_b) \\ r_{B_1} : B_0(\boldsymbol{B}; n_b), N(m) & \rightarrow B_1(\boldsymbol{B}; n_b; m) \\ r_{B_2} : B_1(\boldsymbol{B}; n_b; \{a, n_a\}_{k_b}), & \rightarrow B_2(\boldsymbol{B}; n_b; n_a) \\ r_{B_3} : B_2(\boldsymbol{B}; n_b; n_a) & \rightarrow N(\{n_a, n_b\}_{k_a}), B_3(\boldsymbol{B}; n_b; n_a) \\ r_{B_4} : B_3(\boldsymbol{B}; n_b; n_a), N(m') & \rightarrow B_4(\boldsymbol{B}; n_b; n_a; m') \\ r_{B_5} : B_4(\boldsymbol{B}; n_b; n_a; \{n_b\}_{k_b}) & \rightarrow B_5(\boldsymbol{B}; n_b; n_a) \end{cases}$$

Again, the instantiation rule r_{B_0} instantiate all the variables $\boldsymbol{B}$ to ground terms. Rules r_{B_1}, r_{B_4} model the receiving steps 1 and 3 in protocol (1), while r_{B_3} is the rule corresponding the sending step 2. Finally rules r_{B_2}, r_{B_5} describe the analysis steps performed by the role.

Finally, we define the state portion of the initial configuration (i.e., the initial state) to consist of:

$$\underbrace{\qquad}_{\tilde{N}}\underbrace{\qquad}_{\tilde{A}}\underbrace{I(E), I(K_E), I(K'_E),}_{\tilde{I}}\underbrace{\begin{array}{l}\tilde{\pi}(A; B; K_A; K'_A; K_B), \\ \tilde{\pi}(B; A; K_B; K'_B; K_A), \\ \tilde{\pi}(B; E; K_B; K'_B; K_E), \\ \tilde{\pi}(E; A; K_E; K'_E; K_A),\end{array}}_{\tilde{\pi}}$$

where A, B, E, are specific principals (a and b above were variables), with E acting as the attacker. For each of them, the pseudo-functions $K\_$ and $K'\_$ denote their public and private key, respectively.

In this initial state, the intruder knowledge consists of its name E and its public/private key pair K_E, K'_E. The persistent data $\tilde{\pi}$ defines the attributes (name, public and private key) of each of these principals, in particular of the intruder E who may participate in the protocol as an honest player if he wishes. This is useful, for example, when testing some authenticity property.

3.2. *Protocols as processes*

A security protocol may be described in a fragment of PA where:

- Every communication happens through the net (here $P_{!net}$ is the process that manages the net as a public channel where protocol roles send and receive messages).
- There is an intruder, with some initial knowledge, able to intercept and forge messages passing through the net (here $Q_{!I}$, with initial knowledge Q_{I_0}).
- Each principal starts the protocol in a certain role ρ.

Formally a security protocol, involving a collection of roles $\{\rho\}$, is expressed in PA$_P$ as a *security protocol process* Q, defined as the parallel composition of five components: $P_{!net} \parallel \prod_\rho P_{!\rho} \parallel Q_{!I} \parallel Q_{!\pi} \parallel Q_{I_0}$ where $\prod \mathcal{P}$ denotes the parallel composition of all the processes in $\mathcal{P}$. More precisely:

$P_{!net} = !N_i(x).\overline{N_o}(x).0$ This process describes the behavior of the network as a buffer that copies messages from channel N_i (input to the net) to N_o (output from the net), implementing an asynchronous form of message transmission on top of a synchronous calculus.

$P_{!\rho}$ Each of these replicated sequential processes capture the actions that constitute a role, in the sense defined for MSR$_P$. These processes have the form

$$P_{!\rho} = !\tilde{\pi}(\boldsymbol{x}).\nu\boldsymbol{n}.P_\rho$$

where P_ρ is a sequential process that performs input and output only on the network channels, and that analyses the received messages.

Notice that pattern matching is sufficient for "extracting" a piece of information when Σ_P is used, but more general mechanisms could be considered (as in Crypto-CCS for example [22]). We have used $\tilde{\pi}(\boldsymbol{x}).P$ as a shortcut for $\pi_I(\boldsymbol{x}_1)\ldots\pi_k(\boldsymbol{x}_k).P$, where $\boldsymbol{x}_i \sqsubseteq \boldsymbol{x}$. Formally,

$$P_\rho \ ::= \ 0 \ \mid \ N_o(y).P_\rho \ \mid \ \overline{N_i}(t).P_\rho \ \mid \ [\boldsymbol{x}' = \boldsymbol{t}(\boldsymbol{x})]\, P_\rho$$

$Q_{!I} = !P_{I_1} \parallel \ldots \parallel !P_{I_9} \parallel !P_{I_{10}}$ This is the specification of the intruder model in a Dolev–Yao style. The dedicated channel I holds the information the intruder operates on (it can be either initial, intercepted, or forged). Each P_{I_i}, for $i = 1, \ldots, 9$ describes one capability of the intruder. The additional process $P_{I_{10}}$ has no meaning in term of intruder capability but technically it behaves as a "garbage" collector of messages in the intruder knowledge. Processes P_{I_i} are defined as follows:

$$\begin{aligned}
P_{I_1} &= \pi(x).\overline{I}(x).0\\
P_{I_2} &= \nu n.\overline{I}(n).0\\
P_{I_3} &= N_o(x).\overline{I}(x).0\\
P_{I_4} &= I(x).\overline{I}(x).\overline{N_i}(x).0\\
P_{I_5} &= I(x).\overline{I}(x).[x = \langle x_1,\, x_2\rangle].\overline{I}(x_1).\overline{I}(x_2).0\\
P_{I_6} &= I(x_1).\overline{I}(x_1).I(x_2).\overline{I}(x_2).\overline{I}(\langle x_1,\, x_2\rangle).0\\
P_{I_7} &= \mathtt{Kp}(w).I(y).\overline{I}(y).[w = \langle y,\, y'\rangle].I(x).\overline{I}(x).[x = \{z\}_{y'}].\overline{I}(z).0\\
P_{I_8} &= I(x).\overline{I}(x).I(k).\overline{I}(k).\overline{I}(\{x\}_k).0\\
P_{I_9} &= I(x).0\\
P_{I_{10}} &= I(x).\overline{I}(x).0
\end{aligned}$$

Processes P_{I_1} through P_{I_9} perform the same actions as the MSR$_P$ intruder rules with the same index in Section 3.1. For example, P_{I_5} retrieves an object

x previously memorized as $I(x)$, splits it into the pair (x_1, x_2), and then stores a copy of each of the terms x, x_1 and x_2: this is exactly what r_{I_5} achieved. Channel I is used to store the intruder's knowledge in a distributed way. Process $P_{I_{10}}$ ensures that writing on I is never blocking, even in our synchronous calculus. In particular, it allows expressing every term t known to the intruder as the singleton process $\overline{I}(\underline{t}).0$, since it can rewrite a trailing sequence of outputs $\overline{I}(\underline{t}).\overline{I}(\underline{t}').0$ into $\overline{I}(\underline{t}).0 \parallel \overline{I}(\underline{t}').0$.

$Q_{!\pi} = \prod !\overline{\pi}(\underline{t}).0$ This process represents what we called "persistent information" in the case of MSR$_P$. We can assume the same predicate (here channel) names with the same meaning. This information is made available to client processes on each channel π (e.g., `Kp`). It is assumed that no other process performs an output on π.

$Q_{I_0} = \prod \overline{I}(\underline{t}).0$ for terms $\underline{t}$. Q_{I_0} represents the initial knowledge of the intruder.

In PA$_P$, an *initial state* is a process $(P_{!net} \parallel \prod_\rho !P_\rho \parallel Q_{!I} \parallel Q_{!\pi} \parallel Q_{I_0})$. Subsequent states are obtained by applying the execution rules of PA defined in Section 2.2.

Example 2. In order to gain a better understanding of the PA$_P$ specification methodology, we will now express the *NSPK* protocol (1) in this language.

The PA$_P$ specification of *NSPK* protocol will consist of the following processes:

$$Q_{NSPK} = P_{!net} \parallel P_{!A} \parallel P_{!B} \parallel Q_{!I} \parallel Q_{!\pi} \parallel Q_{I_0}$$

where $P_{!net}$ and $Q_{!I}$ have already been defined. As with MSR$_P$, we rely on the abbreviations $\boldsymbol{A} = (a; b; k_a; k'_a; k_b)$ and $\boldsymbol{B} = (b; a; k_b; k'_b; k_a)$ for the given tuples of variables. The other processes are as follows:

$$P_{!A} =!\tilde{\pi}(\boldsymbol{A}).\ \nu n_a.\ \overline{N_i}(\{a, n_a\}_{k_b}).\ N_o(m).\ [m = \{n_a, n_b\}_{k_a}]\ .\ \overline{N_i}(\{n_b\}_{k_b}).\ 0$$

where precisely $\tilde{\pi}(\boldsymbol{A})$ is a shortcut for the prefix

$$\texttt{Pr}(a).\texttt{PrK}(a; k'_a).\texttt{PbK}(b; k_b).\texttt{Kp}(k_a; k'_a)$$

First, process $P_{!A}$ receives, through channels $\tilde{\pi}$, the instantiating constants of the initiator role. Then it sends the encrypted message $\{a, n_a\}_{k_b}$ on the net, where n_a is a fresh name and k_b the responder's public key. Then, $P_{!A}$ receives a message m that it tries to interpret as $\{n_a, n_b\}_{k_a}$ by decryption using the private key k_a and by splitting the results as the pair (n_a, n_b). If this step succeeds the message $\{n_b\}_{k_b}$ is sent back to the net.

The process $P_{!B}$ representing the responder of *NSPK* is similarly defined as follows:

$$\begin{aligned} P_{!B} = {} & !\tilde{\pi}(\boldsymbol{B}).\ \nu n_b.\ N_o(m).[m = \{a, n_a\}_{k_b}]\ . \\ & \overline{N_i}(\{n_a, n_b\}_{k_a}).\ N_o(m').\ [m' = \{n_b\}_{k_b}]\ .\ 0 \end{aligned}$$

The initial knowledge of the intruder is:

$$Q_{I_0} = \overline{I}(E).0 \parallel \overline{I}(K_E).0 \parallel \overline{I}(K'_E).0$$

i.e., the intruder knows its name and its private/public key pairs. Finally the processes modeling the persistent information are the following:

$$\begin{aligned} Q_{!\pi} = {} & Q_{\tilde{\pi}(A;B;K_A;K'_A;K_B)} \parallel Q_{\tilde{\pi}(B;A;K_B;K'_B;K_A)} \parallel \\ & Q_{\tilde{\pi}(B;E;K_B;K'_B;K_E)} \parallel Q_{\tilde{\pi}(E;A;K_E;K'_E;K_A)} \end{aligned}$$

where $Q_{\tilde{\pi}(x;y;k_x;k'_x;k_y)}$ is the parallel composition of simple replicated processes that output each object in $\tilde{\pi}(x; y; k_x; k'_x; k_y)$ on channels $\tilde{\pi}$, i.e., :

$$!\overline{\mathtt{Pr}}(x).0 \parallel !\overline{\mathtt{PrK}}(x; k'_x).0 \parallel !\overline{\mathtt{PbK}}(y; k_y).0 \parallel !\overline{\mathtt{Kp}}(k_x; k'_x).0\ .$$

4. Encoding protocol specifications

This section describes two encodings: one from MSR$_P$ to PA$_P$ and the other from PA$_P$ to MSR$_P$. As we define these encodings, we assume a common underlying signature Σ_P. In particular, the predicate symbols and terms in MSR$_P$ find their counterpart in channel names and messages in PA$_P$, respectively.

The first mapping, from MSR$_P$ to PA$_P$, is based on the observation that role state predicates force MSR$_P$ rules to be applied sequentially within a role (this is not true for general MSR theories). Minor technicalities are involved in dealing with the presence of multiple instances of a same role (they are addressed through replicated processes).

At its core, the inverse encoding, from PA$_P$ to MSR$_P$, maps sequential agents to a set of MSR$_P$ rules corresponding to roles: we generate appropriate role state predicates in correspondence of the intermediate stages of each sequential process. The bang operator is not directly involved in this mapping as it finds its counterpart in the way rewriting rules are applied. The transformation of the intruder, whose behavior is fixed a priori, is treated off-line in both directions.

Before proceeding we introduce some simplifying assumptions and a preliminary observation. Without loss of generality, we assume that the rewrite rules of an MSR$_P$ theory are written in the following form: variables occurring in two occurrences of a role state predicate $A_{\rho_i}(\boldsymbol{x})$, one in the antecedent and one in the consequent of two consecutive rules, have the same name. Moreover, in the antecedent $A_{\rho_i}(\boldsymbol{t}(\boldsymbol{x}))$ of an analysis rule, we require that all the variables introduced by $\boldsymbol{t}(\boldsymbol{x})$ be distinct from the variables $\boldsymbol{x}'$ in the consequent $A_{\rho_i}(\boldsymbol{x}')$ of the preceding rule. These assumptions, purely syntactical, simplify situations in the proofs without invalidating our analysis. Example 1 implements them.

We begin by characterizing the structure of a generic PA$_P$ *state* reachable from an initial specification (see Section 3.2) as the parallel composition of precisely identified processes. We have the following proposition:

Proposition 1. *Let Q be a* PA$_P$ *initial state. If Q is such that $Q_0 \Rightarrow^* Q$ then Q can be written as*:

$$Q \equiv \overbrace{(P_{!net} \parallel \prod_\rho P_{!\rho} \parallel Q_{!I} \parallel Q_{!\pi})}^{Q_!} \parallel (Q_{net} \parallel \prod_\rho P_\rho \parallel Q_I \parallel Q_{rem})$$

where:

$$\begin{array}{lll}
Q_{net} & ::= 0 \ \mid \ \prod \overline{N_o}(t).0 \\
P_\rho & ::= 0 \ \mid \ N_o(\boldsymbol{x}).P_\rho \ \mid \ \overline{N_i}(\underline{\boldsymbol{t}}).P_\rho \ \mid \ [\underline{\boldsymbol{t}} = \boldsymbol{t'}]P_\rho \\
Q_I & ::= \textit{suffix of } P_{I_j}, \textit{for all } j \\
Q_{rem} & ::= 0 \ \mid \ N_o(x).\overline{N_i}(x).0 \ \mid \ \tilde{\pi}(\boldsymbol{x}).\nu\boldsymbol{n}.P_\rho \ \mid \ \nu\boldsymbol{n}.P_\rho \ \mid \ \prod \overline{\pi}(\underline{t}).0
\end{array}$$

Proof. By induction over the number of transition steps. As the base of the induction let us observe that a PA$_P$ initial state Q_0 is exactly the process $Q_! \parallel Q_{I_0}$ (where $Q_! = P_{!net} \parallel \prod_\rho P_{!\rho} \parallel Q_{!I} \parallel Q_{!\pi}$), and that $Q_0 \Rightarrow^* Q_0$. Then, let be Q such that $Q_0 \Rightarrow^* Q' \Rightarrow Q$. For inductive hypothesis Q' may be written as a process of form $Q_! \parallel (Q_{net} \parallel \prod_\rho P_\rho \parallel Q_I \parallel Q_{rem})$, and it is easy to check that, each transition Q from Q' can be written as well as a process of form $Q_! \parallel (Q'_{net} \parallel \prod_\rho P'_\rho \parallel Q'_I \parallel Q'_{rem})$. □

4.1. *From* MSR$_P$ *to* PA$_P$

This section defines the transformation $\lceil\_\rceil$ that, given an MSR$_P$ configuration $(\tilde{r} : \tilde{s})$ with $\tilde{r} = (\cup_\rho(\tilde{r}_\rho), \tilde{r}_I)$ and $\tilde{s} = (\tilde{N}, \tilde{A}, \tilde{I}, \tilde{\pi})$ returns a PA$_P$ state $Q_! \parallel Q_{net} \parallel \prod_\rho P_\rho \parallel Q_I$ (with $Q_! = (P_{!net} \parallel \prod_\rho P_{!\rho} \parallel Q_{!I} \parallel Q_{!\pi})$).

More precisely $\lceil\_\rceil$ is a tuple of encodings $\lceil\_\rceil^{R_\rho}$, $\lceil\_\rceil^{R_I}$, $\lceil\_\rceil^{N}$ $\lceil\_\rceil^{A_\rho}$, $\lceil\_\rceil^{I}$, $\lceil\_\rceil^{\pi}$, each operating on a different component of the MSR$_P$ configuration, as depicted in the following scheme:

$$\lceil(\cup_\rho(\tilde{r}_\rho) \cup \tilde{r}_I : \tilde{N}, \tilde{A}, \tilde{I}, \tilde{\pi})\rceil =$$

$$\overbrace{(!P_{net} \parallel \underbrace{\prod_\rho P_{!\rho}}_{\lceil\cup_\rho(\tilde{r}_\rho)\rceil^{R_\rho}} \parallel \underbrace{Q_{!I}}_{\lceil\tilde{r}_I\rceil^{R_I}} \parallel \underbrace{Q_{!\pi}}_{\lceil\tilde{\pi}\rceil^{\pi}})}^{Q_!} \parallel (\underbrace{Q_{net}}_{\lceil\tilde{N}\rceil^{N}} \parallel \underbrace{\prod_\rho P_\rho}_{\lceil\tilde{A}\rceil^{A_\rho}} \parallel \underbrace{Q_I}_{\lceil\tilde{I}\rceil^{I}})$$

This definition is interpreted as follows:

- P_{net} is fixed a priori (see Section 3.2);
- $\prod_\rho P_{!\rho}$ and $Q_{!I}$, result from the transformation of respectively $\cup_\rho(\tilde{r}_\rho)$ and $\tilde{r}_I$;
- $Q_{!\pi}$ results from the transformation of $\tilde{\pi}$, and
- Q_{net}, $\prod_\rho P_\rho$, and Q_I result from transformation of, resp., $\tilde{N}$, $\tilde{A}$ and $\tilde{I}$.

Intuitively, transformations $\lceil \cup_\rho(\tilde{r}_\rho)\rceil^{R_\rho}$ and $\lceil \tilde{r}_I \rceil^{R_I}$ return the parallel composition of banged (i.e., proceeded by a !) processes modeling the sequence of actions of each role and of the intruder, respectively. The bang operator makes these processes always available for instantiation as the MSR rules are. The intruder process is fixed a priori and its transformation is obvious. The transformation of $\tilde{r}_\rho$, e.g., the rules of role ρ, is more interesting: it results in a sequential process P_ρ, whose send, receive or match sub-processes are obtained, resp., from send, receive and analysis rules in $\tilde{r}_\rho$ (see also Example 3). Particular attention is reserved for the translation of the first instantiation rule r_{ρ_0}.

The next transformations act on predicates $\tilde{N}$, $\tilde{A}$ and $\tilde{I}$ in the MSR$_P$ state, and return the parallel composition of sequential processes. More precisely, all the predicates $N(t)$ in $\tilde{N}$ are transformed into singleton output processes $\overline{N_o}(t).0$ representing the availability of the ground datum t on the net. Similarly predicates $I(t)$ in $\tilde{I}$ are transformed into output processes $\overline{I}(t).0$ representing the intruder knows the datum t. Finally the transformation of each predicates $A_{\rho_i}(\boldsymbol{t})$, in $\tilde{A}$ returns the suffix of the process P_ρ that model the remaining role rules $r_{\rho_{i+1}}, \ldots, r_{\rho_{l_\rho}}$. Variable in P_ρ are partially instantiated depending on terms in $\boldsymbol{t}$.

The acquisition of permanent facts and the creation of new variables $\boldsymbol{x}$ are mapped, resp., to a sequence of input actions from processes $Q_{!\pi}$, and actions νx for each x in $\boldsymbol{x}$. In turn $Q_{!\pi}$ is the parallel composition of banged output processes $\overline{\pi}(\boldsymbol{t}).0$, each obtained from a permanent predicates $\pi(\boldsymbol{t})$ in $\tilde{\pi}$. Their task is to make permanent fact always available to be received.

Whenever unambiguous, we will omit the identifying subscript from the encoding functions $\lceil\_\rceil^{R_\rho}$, $\lceil\_\rceil^{R_I}$, $\lceil\_\rceil^{N}$ $\lceil\_\rceil^{A_\rho}$, $\lceil\_\rceil^{I}$, or $\lceil\_\rceil^{\pi}$, simplifying them to $\lceil\_\rceil$.

$\lceil\_\rceil^{R_\rho}$. In transforming processes $P_{!\rho}$, for each role ρ, a subroutine function $\lceil\_\rceil^{\#}_{(\boldsymbol{x})}$ is called by the top level transformation $\lceil\_\rceil$. $\lceil\_\rceil^{\#}_{(\boldsymbol{x})}$ ranges over the set of role rules $\cup_\rho(\tilde{r}_\rho)$, and takes a tuple $\boldsymbol{x}$ of variables as parameter. This parameter, initially the empty tuple ϵ, collects variables used along the rewriting rule, and uses them opportunely in the building process. We define it on the structure of the role rule $r_{\rho_i} \in \tilde{r}_\rho$ involved. Formally for $i = 0$:

$$\lceil r_{\rho_0}\rceil = \tilde{\pi}(\boldsymbol{x}).\nu\boldsymbol{n}.\lceil r_{\rho_1}\rceil^{\#}_{(\boldsymbol{x};\boldsymbol{n})} \qquad \text{if } r_{\rho_0} : \tilde{\pi}(\boldsymbol{x}) \to \exists \boldsymbol{n}.A_{\rho_0}(\boldsymbol{x};\boldsymbol{n}), \tilde{\pi}(\boldsymbol{x})$$

A role generation rule is mapped onto a process which first receives, in sequence, permanent terms via the channels π in $\tilde{\pi}$ and then generates all the new names $\boldsymbol{n}$ used in this role.

For $0 < i \leqslant l_\rho - 1$:

$$\lceil r_{\rho_{i+1}} \rceil^{\#}_{(\boldsymbol{x})} = \begin{cases} \overline{N_i}(t(\boldsymbol{x})).\lceil r_{\rho_{i+2}} \rceil^{\#}_{(\boldsymbol{x})} & \text{, if } r_{\rho_{i+1}} = A_{\rho_i}(\boldsymbol{x}) \rightarrow A_{\rho_{i+1}}(\boldsymbol{x}), N(t(\boldsymbol{x})) \\ N_o(y)\lceil r_{\rho_{i+2}} \rceil^{\#}_{(\boldsymbol{x};y)} & \text{, if } r_{\rho_{i+1}} = A_{\rho_i}(\boldsymbol{x}), N(y) \rightarrow A_{\rho_{i+1}}(\boldsymbol{x};y) \\ [\boldsymbol{x} = \boldsymbol{t}(\boldsymbol{x'})]\ \lceil r_{\rho_{i+2}} \rceil^{\#}_{(\boldsymbol{x'})}, & \text{if } r_{\rho_{i+1}} = A_{\rho_i}(\boldsymbol{t}(\boldsymbol{x'})), \rightarrow A_{\rho_{i+1}}(\boldsymbol{x'}) \end{cases}$$

The transformation of a send or a receive rewriting rule is straightforward. The translation of an analysis rewriting rule is less obvious: the matching $[\boldsymbol{x} = \boldsymbol{t}(\boldsymbol{x'})]$ is intended to simulate the matching that – in the semantics of MSR – happens between the terms in consequent, $A_{\rho_i}(\boldsymbol{x})$, of rule r_{ρ_i} and the terms in the antecedent $A_{\rho_i}(\boldsymbol{t}(\boldsymbol{x'}))$ of (actual) rule $r_{\rho_{i+1}}$. Finally and with a little abuse of notation, we set $\lceil r_{\rho_{l_\rho+1}} \rceil^{\#}_{(\boldsymbol{x})} = 0$.

The final process defining the role ρ behavior is the following: $P_\rho \overset{def}{=} \lceil r_{\rho_0} \rceil$

$\lceil \_ \rceil^{R_I}$. The intruder is handled by simply mapping $\tilde{r}_I$ to $Q_{!I}$. More precisely, we define the transformation function $\lceil \_ \rceil$ that relates the intruder rewriting rule r_{I_j} with the sequential agents P_{I_j} defined in Section 3.2. Moreover the transformation produces the additional process $!P_{I_{10}}$.

At this point the transformation is complete as soon as the state $\tilde{s} = (\tilde{N}, \tilde{A}, \tilde{I}, \tilde{\pi})$ is treated.

$\lceil \_ \rceil^{A_\rho}$. For each $A_{\rho_i}(\boldsymbol{t}) \in \tilde{A}$, we define $P_{A_{\rho_i}(\boldsymbol{t})} = \lceil r_{\rho_{i+1}} \rceil^{\#}_{(\boldsymbol{x})}[\boldsymbol{t}/\boldsymbol{x}]$, where $\lceil r_{\rho_{i+1}} \rceil^{\#}_{(\_)}$ was defined above and $\boldsymbol{x}$ are the variables appearing as argument of the *consequent* predicate $A_{\rho_i}(\boldsymbol{x})$ in r_{ρ_i}.

$\lceil \_ \rceil^{N}$, $\lceil \_ \rceil^{I}$, $\lceil \_ \rceil^{\pi}$. The multiset $\tilde{N}$ guides the definition of Q_{net}, that is $Q_{net} \overset{def}{=} \prod_{N(t)\in\tilde{N}} \overline{N}(t).0$. Similarly, $Q_I \overset{def}{=} \prod_{I(t)\in\tilde{I}} \overline{I}(t).0$, and $Q_{!\pi} \overset{def}{=} \prod_{\pi(t)\in\tilde{\pi}} !\overline{\pi}(t).0$. Formally:

$$\begin{array}{ll|ll|ll} \lceil \cdot \rceil & = 0 & \lceil \cdot \rceil & = 0 & \lceil \cdot \rceil & = 0 \\ \lceil N(t), \tilde{N} \rceil & = \overline{N_o}(t).0 \parallel \lceil \tilde{N} \rceil & \lceil I(t), \tilde{I} \rceil & = \overline{I}(t).0 \parallel \lceil \tilde{I} \rceil & \lceil \pi(t), \tilde{\pi} \rceil & = !\overline{\pi}(t).0 \parallel \lceil \tilde{\pi} \rceil \end{array}$$

Example 3 (Translation of NSPK from MSR$_P$ to PA$_P$). We now provide an example on how the translation $\lceil \_ \rceil$ works, by applying it to the MSR$_P$ specification of *NSPK* given in Section 3.1.

$$\lceil \overbrace{r_{A_0}}^{\tilde{\pi}(\boldsymbol{A})\rightarrow\exists n_a.\tilde{\pi}(\boldsymbol{A}),A_0(\boldsymbol{A};n_a)} \rceil \quad = \quad !\tilde{\pi}(\boldsymbol{A}).\nu n_a.\lceil r_{A_1} \rceil^{\#}_{(\boldsymbol{A};n_a)}$$

$$\lceil \overbrace{r_{A_1}}^{A_0(\boldsymbol{A};n_a)\rightarrow N(\{a,n_a\}_{k_b}),A_1(\boldsymbol{A};n_a)} \rceil^{\#}_{(\boldsymbol{A};n_a)} \quad = \quad \overline{N_i}(\{a, n_a\}_{k_b}).\lceil r_{A_2} \rceil^{\#}_{(\boldsymbol{A};n_a)}$$

$$
\begin{aligned}
\lceil \overbrace{r_{A_2}}^{A_1(\boldsymbol{A};n_a),N(m)\rightarrow A_2(\boldsymbol{A};n_a;m)} \rceil^{\#}_{(\boldsymbol{A};n_a)} &= N_o(m).\lceil r_{A_3} \rceil^{\#}_{(\boldsymbol{A};n_a;m)} \\
\lceil \overbrace{r_{A_3}}^{A_2(\boldsymbol{A};n_a;\{n_a,n_b\}_{k_a})\rightarrow A_3(\boldsymbol{A};n_a;n_b)} \rceil^{\#}_{(\boldsymbol{A};n_a;m)} &= \begin{array}{l}[(\boldsymbol{A};n_a;m) = (\boldsymbol{A};n_a;\{n_a,n_b\}_{k_a})]. \\ \lceil r_{A_4} \rceil^{\#}_{(\boldsymbol{A};n_a;n_b)}\end{array} \\
\lceil \overbrace{r_{A_4}}^{A_3(\boldsymbol{A};n_a;n_b)\rightarrow N(\{n_b\}_{k_b}),A_4(\boldsymbol{A};n_a;n_b)} \rceil^{\#}_{(\boldsymbol{A};n_a;n_b)} &= \overline{N_i}(\{n_b\}_{k_b}).\lceil \cdot \rceil^{\#}_{(\boldsymbol{A};n_a;n_b)} \\
\lceil \cdot \rceil^{\#}_{(\boldsymbol{A};n_a;n_b)} &= 0
\end{aligned}
$$

In summary:

$$
\begin{aligned}
\lceil \mathcal{R}_A \rceil = {} & !\tilde{\pi}(\boldsymbol{A}).\nu n_a.\overline{N_i}(\{a,n_a\}_{k_b}).N_o(m). \\
& [\boldsymbol{A};n_a;m = \boldsymbol{A};n_a;\{n_a,n_b\}_{k_a}]\,.\overline{N_i}(\{n_b\}_{k_b}).0
\end{aligned}
$$

which can be simplified into

$$
\begin{aligned}
\lceil \mathcal{R}_A \rceil = {} & !\tilde{\pi}(\boldsymbol{A}).\nu n_a.\overline{N_i}(\{a,n_a\}_{k_b}).N_o(m). \\
& [m = \{n_a,n_b\}_{k_a}]\,.\overline{N_i}(\{n_b\}_{k_b}).0
\end{aligned}
$$

by means of the structural equivalence (that removes positional corresponding and identical items in a tuples pattern matching). This process is exactly the same provided in Section 3.2.

Similarly (omitting the details) it is easy to check that:

$$
\begin{aligned}
\lceil \mathcal{R}_B \rceil = {} & !\tilde{\pi}(\boldsymbol{B}).\nu n_b.N_o(m). \\
& [\boldsymbol{B};n_b;m = \boldsymbol{B};n_b;\{a,n_a\}_{k_b}]\,.\overline{N_i}(\{n_a,n_b\}_{k_a}). \\
& N_o(m).[\boldsymbol{B};n_b;n_a;m' = \boldsymbol{B};n_b;n_a;\{n_b\}_{k_b}]\,.0
\end{aligned}
$$

4.2. *From* PA$_P$ *to* MSR$_P$

This section defines the transformation $\lfloor\_\rfloor$ that given a PA$_P$ state returns a configuration in MSR$_P$. Indeed $\lfloor\_\rfloor$ consists of encodings $\lfloor\_\rfloor_{!\rho}$, $\lfloor\_\rfloor_{!I}$, $\lfloor\_\rfloor_{net}$, $\lfloor\_\rfloor_{\rho}$ $\lfloor\_\rfloor_{I}$ and $\lfloor\_\rfloor_{\pi}$, each operating on different sub-processes of the PA$_P$ state. The following schema describes the overall encoding pictorially (processes involved in any transformation are boxed):

$$
\lfloor(\overbrace{P_{!net} \parallel \boxed{\textstyle\prod_\rho P_{!\rho} \parallel Q_{!I} \parallel Q_{!\pi}}}^{Q_!})\rfloor \parallel (\boxed{Q_{net} \parallel \textstyle\prod_\rho P_\rho \parallel Q_I} \parallel Q_{rem}) =
$$

$$
(\ \underbrace{\cup_\rho(\tilde{r}_\rho)}_{\lfloor\prod_\rho P_{!\rho}\rfloor_{!\rho}} \ \cup\ \underbrace{\tilde{r}_I}_{\lfloor Q_{!I}\rfloor_{!I}} \ :\ \underbrace{\tilde{N}}_{\lfloor Q_{net}\rfloor_{net}} \ ,\ \underbrace{\tilde{A}}_{\lfloor\prod_\rho P_\rho\rfloor_\rho} \ \underbrace{\tilde{I}}_{\lfloor Q_I\rfloor_I} \ \underbrace{\tilde{\pi}}_{\lfloor Q_{!\pi}\rfloor_\pi} \ ,)
$$

Note that the following processes are not involved in any transformation:

- $P_{!net}$, since it implements a form of buffering that is unnecessary in MSR;
- Q_{rem}, since it represents partial computations (see Proposition 1). As we will see later, they will not have any significant MSR$_P$ counterpart.

Intuitively $\lfloor\prod_\rho P_{!\rho}\rfloor_{!\rho}$ analyzes each (un-banged) sequential processes P_ρ in $\prod_\rho P_{!\rho}$ and for each ρ returns the multiset of rule corresponding to P_ρ's sequential steps. Input, output and analysis sub-process in P_ρ are mapped into receive, send, and analysis rewriting rules for role ρ, respectively. Prefixes $\nu\boldsymbol{x}$ and input sequences $\tilde{\pi}(\boldsymbol{x})$ are turned into an instantiation rule. Technicalities are needed for the management of variables and of the predicate indexes in building rules r_{ρ_i}'s. Two parameters, the step number and the variables, are passed along the transformation. Similar devices support the transformation of each processes P_ρ in $\prod_\rho P_\rho$. They represent partial execution of the protocol by role ρ, their analysis produces the state predicates $A_{\rho_i}(\boldsymbol{t})$, for suitable i and $\boldsymbol{t}$.

The transformation of $Q_{!I}$ and $Q_{!\pi}$ are straightforward: the former maps directly to the intruder rewriting rules of MSR$_P$, while in the latter each $!\overline{\pi}(\boldsymbol{t}).0$ in $Q_{!\pi}$ is mapped to the persistent predicates $\pi(\boldsymbol{t})$. The same can be said about processes Q_{net}: each sequential process $\overline{N_o}(t).0$ is mapped into a predicate $N(t)$ in the MSR$_P$ state.

The transformation of the processes in Q_I is more complex. Indeed, we need to distinguish between processes that represent immediately available intruder knowledge (e.g., $\overline{I}(t).0$) from processes that do not (e.g., $N_o(x).\overline{I}(x).0$). The former are transformed in corresponding intruder predicates $I(t)$, while the latter are generally discarded. Generally speaking $\lfloor\_\rfloor$ is not injective, and similar situations can happen while transforming processes into MSR$_P$ states. Said differently, PA$_P$ steps are finer grained then MSR$_P$'s, and as a consequence some processes do not represent proper MSR objects (for example processes in Q_{rem}) and they have to be ignored, while others represent MSR$_P$ objects even when they are only partially completed (for example processes $\overline{I}(\underline{t}).P_I')$ and their translation can be anticipated (see also Fig. 1 or later for details).

In the following, with a little abuse of notation, we drop the subscript from the transformations, $\lfloor\_\rfloor_{!\rho}$, $\lfloor\_\rfloor_{!I}$, $\lfloor\_\rfloor_{net}$, $\lfloor\_\rfloor_\rho$, $\lfloor\_\rfloor_I$ and $\lfloor\_\rfloor_\pi$, when no ambiguity arises, writing them instead as $\lfloor\_\rfloor$. We now describe each transformation in detail.

$\lfloor\_\rfloor_{!\rho}$. The basic translation involves the transformation function $\lfloor\_\rfloor^{\#}_{(i;\boldsymbol{x})}$ for the $P_{!\rho}$'s (called as a subroutine by the top level transformation $\lfloor\_\rfloor$) which, given a sequential agent representing a role ρ, returns the multiset of rules $\tilde{r}_\rho$. Here i is a non-negative integer. Formally:

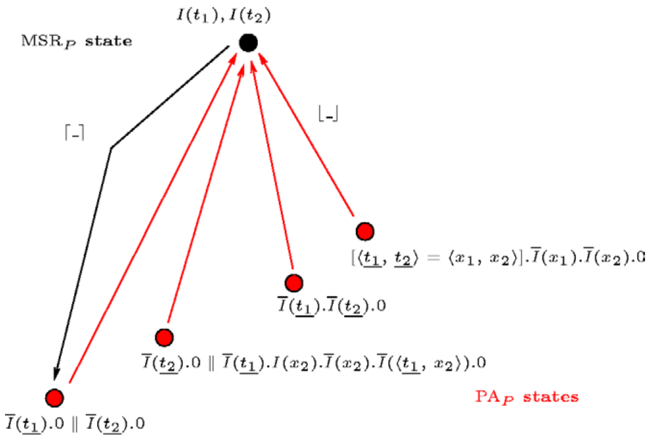
$I(t_1), I(t_2)$
MSR$_P$ state
$\lceil\_\rceil$
$\lfloor\_\rfloor$
$[\langle \underline{t_1},\ \underline{t_2}\rangle = \langle x_1,\ x_2\rangle].\overline{I}(x_1).\overline{I}(x_2).0$
$\overline{I}(\underline{t_1}).\overline{I}(\underline{t_2}).0$
$\overline{I}(\underline{t_2}).0 \parallel \overline{I}(\underline{t_1}).I(x_2).\overline{I}(x_2).\overline{I}(\langle \underline{t_1},\ x_2\rangle).0$
PA$_P$ states
$\overline{I}(\underline{t_1}).0 \parallel \overline{I}(\underline{t_2}).0$

Fig. 1. A Possible Situation when Applying $\lfloor\lceil\_\rceil_I\rfloor_I$.

$$\begin{aligned}
\lfloor\tilde{\pi}(\boldsymbol{x}).\nu\boldsymbol{n}.P'_\rho\rfloor &= \{\tilde{\pi}(\boldsymbol{x}) \to \exists\boldsymbol{n}.A_{\rho_0}(\boldsymbol{n};\boldsymbol{x})\} \ \cup \ \lfloor P'_\rho\rfloor^{\#}_{(1\,:(\boldsymbol{x};\boldsymbol{n}))}\\
\lfloor N_o(y).P'_\rho\rfloor^{\#}_{(i\,:\boldsymbol{x})} &= \{A_{\rho_{i-1}}(\boldsymbol{x}), N(y) \to A_{\rho_i}(\boldsymbol{x};y)\} \ \cup \ \lfloor P'_\rho\rfloor^{\#}_{(i+1\,:(\boldsymbol{x};y))}\\
\lfloor\overline{N_i}(t).P'_\rho\rfloor^{\#}_{(i\,:\boldsymbol{x})} &= \{A_{\rho_{i-1}}(\boldsymbol{x}) \to A_{\rho_i}(\boldsymbol{x}), N(t)\} \ \cup \ \lfloor P'_\rho\rfloor^{\#}_{(i+1\,:\boldsymbol{x})}\\
\lfloor[\boldsymbol{x}' = \boldsymbol{t}(\boldsymbol{x}'')]\,.P'_\rho\rfloor^{\#}_{(i\,:\boldsymbol{x})} &= \{A_{\rho_{i-1}}(\boldsymbol{x}[\boldsymbol{t}(\boldsymbol{x}'')/\boldsymbol{x}']) \to A_{\rho_i}(\boldsymbol{x}[(\boldsymbol{x}''- \boldsymbol{x})/\boldsymbol{x}']), N(t)\}\\
&\qquad \cup \ \lfloor P'_\rho\rfloor^{\#}_{(i+1\,:(\boldsymbol{x}[(\boldsymbol{x}''-\boldsymbol{x})/\boldsymbol{x}']))}\\
\lfloor 0\rfloor^{\#}_{(i;\boldsymbol{x})} &= \cdot
\end{aligned}$$

The transformation of a send, of a receive and of a new process are quite obvious and require no additional comment. The translation of a match process $[\boldsymbol{x}' = \boldsymbol{t}(\boldsymbol{x}'')]\,.P'_\rho$, whose aim is to analyze some previously received message, yields an analysis rewrite rule. It would be straightforward if all the variables of the role were matched each time (possibly redundantly) as these variables could be used to build the corresponding role predicate. Instead, only a subset of variable appears during matching (the one that are being analyzed), while the corresponding role predicate needs all of them. We reconstruct them be carrying a parameter which stores the tuple of all the variables used so far by the role. With this as a template, we can construct the right tuples in the rule's antecedent and in the rule's consequent.

$\lfloor\_\rfloor_{!I}$. The intruder process $Q_{!I}$ is mapped directly to the MSR$_P$ intruder rules $\tilde{r}_I$, with each $!P_{I_j}$ associated with r_{I_j}. Process $!P_{I_{10}}$ is dropped.

$\lfloor\_\rfloor_{net}$. Each occurrence of a process $\overline{N_o}(t).0$ in Q_{net} is mapped to a state element $N(t)$.

$\lfloor\_\rfloor_\rho$. Let $P_{!\rho}$ be the role specification of which an object P_ρ in $\prod_\rho P_\rho$ is an instantiated suffix and $\theta = [\boldsymbol{x}/\boldsymbol{t}]$ the witnessing substitution. If P_ρ starts with either a persistent input $\pi(\boldsymbol{x})$ or the ν operator, we set $\lfloor P_\rho \rfloor = \cdot$. Otherwise, let i be the index at which P_ρ occurs in $P_{!\rho}$ as for the above definition. Then $\lfloor P_\rho \rfloor = A_{\rho_i}(\boldsymbol{t})$.

$\lfloor \_ \rfloor_I$. Each object in Q_I (that, we remind, contains all the prefixes of P_{I_j} processes), is translated using the function $\lfloor \_ \rfloor_I$, defined below:

$$\lfloor 0 \rfloor_I = \lfloor \overline{N_o}(\underline{t}).0 \rfloor_I = \lfloor \nu n.P_I \rfloor_I = \lfloor I(x).P_I \rfloor_I = \lfloor \pi(x).P_I \rfloor_I = \cdot$$

$$\begin{aligned} \lfloor \overline{I}(\underline{t}).P_I \rfloor_I &= I(\underline{t}), \lfloor P_I \rfloor_I \\ \lfloor [\underline{\boldsymbol{t}} = \boldsymbol{t}(\boldsymbol{x})]\ .P_I \rfloor_I &= \begin{cases} \lfloor P_I[\theta] \rfloor_I & \text{if } \boldsymbol{t}(\boldsymbol{x})[\theta] = \underline{\boldsymbol{t}} \\ \cdot & \text{otherwise} \end{cases} \end{aligned}$$

$\lfloor \_ \rfloor_\pi$. Each process $!\pi(\boldsymbol{x})$ in $P_{!\pi}$, or $\pi(\boldsymbol{x})$ in P_π is translated into the state object $\pi(\boldsymbol{x})$.

The intuition underlying the definition of $\lfloor \_ \rfloor_I$ is to collect all the ground output events of a partially executed intruder processes (i.e., processes that are suffixes of some P_{I_j}, but that do have not the form $\overline{I}(t).0)$[1] as process $P_{I_{10}}$ has the potential of turning them into the canonical form $\overline{I}(t).0$. In this way, we map any such intruder suffix into an MSR$_P$ state where this knowledge is already present. In particular, each object $\overline{I}(t).0$ (resp., the $\overline{I}(t).\overline{I}(t).0$) in Q_I is rendered as the state element $I(t)$ (resp., pair of elements $I(t), I(t)$), and that the unbanged processes P_{I_j} are mapped into the empty multiset. Note that $\lfloor \_ \rfloor_I$ is not injective.

$P_{!net}$ and Q_{rem} disappear (i.e., they are mapped onto the empty multiset).

Example 4 (Translation of NSPK from PA$_P$ to MSR$_P$). We now provide an example on how translation $\lfloor \_ \rfloor$ works, by applying it to the PA$_P$ specification of *NSPK* given in Section 3.2. Let us start by considering the process P_A:

$$P_A = \tilde{\pi}(\boldsymbol{A}).\nu n_a.\overbrace{\overline{N_i}(\{a, n_a\}_{k_b}).\underbrace{N_o(m).\overbrace{[m = \{n_a, n_b\}_{k_a}]\ .\underbrace{\overline{N_i}(\{n_b\}_{k_b}).0}_{P''''_A}}^{P'''_A}}_{P''_A}}^{P'_A}$$

[1] From now on let us call them all *intruder partial suffixes*.

we have:

$$\begin{aligned}
\lfloor P_A \rfloor &= \tilde{\pi}(\boldsymbol{A}) \rightarrow \exists n_a.\tilde{\pi}(\boldsymbol{A}), A_0(\boldsymbol{A}; n_a) \\
&\quad \cup \lfloor P'_A \rfloor^{\#}_{(1:(\boldsymbol{A};n_a))} \\
\lfloor P'_A \rfloor^{\#}_{(1:(\boldsymbol{A};n_a))} &= A_0(\boldsymbol{A}; n_a) \rightarrow N(\{A, n_a\}_{k_b}), A_1(\boldsymbol{A}; n_a) \\
&\quad \cup \lfloor P''_A \rfloor^{\#}_{(2:(\boldsymbol{A};n_a))} \\
\lfloor P''_A \rfloor^{\#}_{(2:(\boldsymbol{A};n_a))} &= A_1(\boldsymbol{A}; n_a), N(m) \rightarrow A_2(\boldsymbol{A}; n_a; m) \\
&\quad \cup \lfloor P'''_A \rfloor^{\#}_{(3:(\boldsymbol{A};n_a;m))} \\
\lfloor P'''_A \rfloor^{\#}_{(3:(\boldsymbol{A};n_a;m))} &= A_2(\boldsymbol{A}; N_A; \{n_a, n_b\}_{k_a}) \rightarrow A_3(\boldsymbol{A}; n_a; n_b) \\
&\quad \cup \lfloor P'''_A \rfloor^{\#}_{(4:(\boldsymbol{A};n_a;n_b))} \\
\lfloor P''''_A \rfloor^{\#}_{(4:(\boldsymbol{A};n_a;n_b))} &= A_3(\boldsymbol{A}; n_a; n_b) \rightarrow N(\{n_b\}_{k_b}), A_4(\boldsymbol{A}; n_a; n_b) \\
&\quad \cup \lfloor 0 \rfloor^{\#}_{(5:(\boldsymbol{A};n_a;n_b))} \\
\lfloor 0 \rfloor^{\#}_{(5:(\boldsymbol{A};n_a;n_b))} &= \cdot
\end{aligned}$$

In summary:

$$\lfloor P_A \rfloor = \begin{cases}
\tilde{\pi}(\boldsymbol{A}) & \rightarrow \exists n_a.\tilde{\pi}(\boldsymbol{A}), A_0(\boldsymbol{A}; n_a) \\
A_0(\boldsymbol{A}; n_a) & \rightarrow N(\{a, n_a\}_{k_b}), A_1(\boldsymbol{A}; n_a) \\
A_1(\boldsymbol{A}; n_a), N(m) & \rightarrow A_2(\boldsymbol{A}; n_a; m) \\
A_2(\boldsymbol{A}; n_a; \{n_a, n_b\}_{k_a}) & \rightarrow A_3(\boldsymbol{A}; n_a; n_b) \\
A_3(\boldsymbol{A}; n_a; n_b) & \rightarrow N(\{n_b\}_{k_b}), A_4(\boldsymbol{A}; n_a; n_b)
\end{cases}$$

Similarly (omitting details):

$$\lfloor P_B \rfloor = \begin{cases}
\tilde{\pi}(\boldsymbol{B}) & \rightarrow \exists n_b.\tilde{\pi}(\boldsymbol{B}), B_0(\boldsymbol{B}; n_b) \\
B_0(\boldsymbol{B}; n_b), N(m) & \rightarrow B_1(\boldsymbol{B}; n_b; m) \\
B_1(\boldsymbol{B}; n_b; \{a, n_a\}_{k_b}), & \rightarrow B_2(\boldsymbol{B}; n_b; n_a) \\
B_2(\boldsymbol{B}; n_b; n_a) & \rightarrow N(\{n_a, n_b\}_{k_a}), B_3(\boldsymbol{B}; n_b; n_a) \\
B_3(\boldsymbol{B}; n_b; n_a), N(m') & \rightarrow B_4(\boldsymbol{B}; n_b; n_a; m') \\
B_4(\boldsymbol{B}; n_b; n_a; \{n_b\}_{k_b}) & \rightarrow B_5(\boldsymbol{B}; n_b; n_a)
\end{cases}$$

5. Correspondence

This section introduces a correspondence relation between MSR$_P$ configurations and PA$_P$ states, such that two corresponding computations are characterized by *identical network messages and intruder knowledge, step by step*. This will allow us to prove that the translations presented in this paper are reachability-preserving in a very strong sense. Indeed, we show that our encodings transform a configuration (resp., a state) into a state (resp., configuration) that correspond to each other in our relation, and this implies that our encodings can preserve secrecy and authenticity properties while going from MSR to PA and vice versa (this is further discussed in Section 6). In the following we formalize the notion of observation and transition step *w.r.t.* the intruder and the network in the MSR and PA frameworks.

Our notion of observation is concerned with only those messages representing terms in the net and the intruder knowledge. They are given by the predicates $N(t)$ and $I(t)$ in an MSR$_P$ configuration. Formally we have:

Definition 1. Given a multiset of ground atoms $\tilde{s}$ and a predicate name $a \in \{N, I\}$, we define the *projection* of $\tilde{s}$ along a as the set $\mathit{Prj}_a(\tilde{s}) = \{t : a(t) \in \tilde{s}\}$. If $C = (\tilde{r}; \tilde{s})$ is a configuration, we set $\mathit{Prj}_a(\tilde{C}) = \mathit{Prj}_a(\tilde{s})$.

Collecting the network messages and the intruder knowledge of a PA$_P$ state P is trickier because of the particular form of the processes representing the intruder and the network (see Section 3). More precisely, these terms appear in output actions (over channels N_o or I) that will be surely executed by either Q_I or Q_{net}. Indeed, Q_I and Q_{net} outputs (on those channels) are always realizable, because processes $P_{I_{10}}$ and $P_{!net}$ can always accept them as input. In order to collect those messages we introduce the notation $Q \overset{\alpha}{\rightarrow}$ to indicate that α is the set of output actions that process Q (intended to be Q_I or Q_{net}) is able to execute in later steps of execution. Formally:

Definition 2. Given a process Q, the judgment $Q \overset{\alpha}{\rightarrow}$ is defined by the following rules:

$$\frac{}{0 \overset{\emptyset}{\rightarrow}} \qquad \frac{}{a(\boldsymbol{x}).P \overset{\emptyset}{\rightarrow}} \qquad \frac{P \overset{\alpha}{\rightarrow}}{\overline{a}(\boldsymbol{t}).P \overset{\{\overline{a}(\boldsymbol{t})\} \cup \alpha}{\longrightarrow}} \qquad \frac{}{\nu n.P \overset{\emptyset}{\rightarrow}} \qquad \frac{Q' \overset{\alpha}{\rightarrow} \qquad Q \equiv Q'}{Q \overset{\alpha}{\rightarrow}},$$

$$\frac{Q \overset{\alpha}{\rightarrow} \qquad\qquad P \overset{\alpha'}{\rightarrow}}{(Q \parallel P) \overset{\alpha \cup \alpha'}{\longrightarrow}} \qquad \frac{P[\theta] \overset{\alpha}{\rightarrow} \qquad\qquad \underline{\boldsymbol{t}'} = \boldsymbol{t}[\theta]}{[\underline{\boldsymbol{t}'} = \boldsymbol{t}]\,.P \overset{\alpha}{\rightarrow}} \qquad \frac{\nexists \theta : \underline{\boldsymbol{t}'} = \boldsymbol{t}[\theta]}{[\underline{\boldsymbol{t}'} = \boldsymbol{t}]\,.P \overset{\emptyset}{\rightarrow}}.$$

In the following we write $\overline{a}(\boldsymbol{t}) \in Q$ if $\overline{a}(\boldsymbol{t}) \in \alpha$ where $\alpha : Q \overset{\alpha}{\rightarrow}$.

Definition 3. Let a be a channel label in $\{N_o, I\}$, we define the *observations* of process Q along a as the set $\mathit{Obs}_a(Q) = \{\boldsymbol{t} : \overline{a}(\boldsymbol{t}) \in Q\}$.

Using Definitions 1 and 3, we make precise what we intend for an MSR$_P$ configuration and a PA$_P$ state to be corresponding.

Definition 4. Given an MSR$_P$ configuration C and a PA$_P$ state Q. We say that C and Q are *corresponding*, written $C \bowtie Q$, if and only if the following conditions hold:

1. $\mathit{Prj}_N(C) = \mathit{Obs}_{N_o}(Q)$,
2. $\mathit{Prj}_I(C) = \mathit{Obs}_I(Q)$.

Informally $C \bowtie Q$ means that the messages that are lying on the net and the intruder knowledge are the same in configuration C and state Q.

The interaction between our notions of observation and our encodings is captured in the following proposition:

Proposition 2. *Let C be an* MSR$_P$ *configuration, and Q be a* PA$_P$ *state. Then*:

$$\lfloor\lceil C\rceil\rfloor = C; \tag{2}$$

$$\lceil\lfloor Q\rfloor\rceil = Q' \text{ where } Q' \text{ is such that } \lfloor Q'\rfloor \bowtie Q, \tag{3}$$

$$\mathit{Obs}_{N_o}(Q') = \mathit{Obs}_{N_o}(Q) \text{ and } \mathit{Obs}_I(Q') = \mathit{Obs}_I(Q).$$

Proof. The critical point here is when the non injective $\lfloor\_\rfloor$ function is applied. More precisely, $\lfloor\_\rfloor$ shows its non-injectivity when dealing with:

(a) *intruder partial suffixes* i.e., suffixes of some P_{I_j} that do not have the form $\bar{I}(t).0$;

(b) not-yet-instantiated process roles, i.e., un-banged processes in P_ρ starting with π or ν.

In proving (2), we observe that starting from an MSR$_P$ configuration C, process $\lceil C\rceil$ contain neither intruder partial suffixes nor not-yet-instantiated role processes. As a consequence by applying again $\lfloor\_\rfloor$, an easy induction yields C back.

More difficult is the proof of (3). Here Q may contain some process that is an intruder partial suffix, or a not-yet-instantiated process role. In this case different Q, Q', may converge, via $\lfloor\_\rfloor$, to the same set of predicates $\tilde{\pi}, \tilde{I}$. However not-yet-instantiated process roles do not affect the $\bowtie$ relation, because only communication over π or $\mathbf{pa}_\nu$ transitions are possible from them. Then all the remaining difficulties are hidden in intruder partial suffixes. In Fig. 1, we have depicted one of these situation, involving where partial suffixes of P_{I_5} and P_{I_6}. Now we can observe that:

- because of the way we have defined $\mathit{Obs}_I(\_)$ and from the fact that $\lfloor Q\rfloor_I = \lfloor Q'\rfloor_I = \ldots = \tilde{I}$, we have that $\mathit{Obs}_I(Q) = \mathit{Obs}_I(Q') = \ldots$, i.e., all the P_I's are equivalent *w.r.t.* the following relation

$$\mathcal{O}(Q_1, Q_2) \stackrel{\mathit{def}}{=} \mathit{Obs}_I(Q_1) = \mathit{Obs}_I(Q_2)$$

  From now on let us consider a witness $[Q]$ of the quotient class $Q_I/\mathcal{O}$.
- $\mathit{Prj}_I(\lfloor Q'\rfloor_I) = \mathit{Obs}_I(Q')$ for all $Q' \in [Q']$, because $\lfloor\_\rfloor_I$ is build exactly to maintain the intruder knowledge.

Now when applying $\lceil\lfloor Q\rfloor_I\rceil_I$ back for some $Q' \in [Q]$, by definition of $\lceil\_\rceil_I$, we obtain exactly that $Q^{\#} \in [Q]$ that contain no partial suffixes of P_{I_j}. Again Fig. 1 may help visualize the intuition. Analogous considerations (indeed simpler) can be provided when predicates $\tilde{N}$ and processes in P_{net} are involved. □

Moreover we have that an MSR$_P$ configuration always corresponds to its encoding in PA$_P$:

Lemma 1. *Let C be an* MSR$_P$ *configuration. Then $C \bowtie \lceil C \rceil$.*

Proof. Observe that $\lceil \tilde{N} \rceil = \prod_{N(t) \in \tilde{N}} \overline{N_o}(t).0$, that $\lceil \tilde{I} \rceil = \prod_{I(t) \in \tilde{I}} \overline{I}(t).0$, and that no other multiset in C generates any $\overline{N_o}(t).0$ or $\overline{I}(t).0$, via $\lceil \_ \rceil$. Then it easily follows that:

$$\begin{aligned} Prj_N(C) &= Obs_{N_o}(\lceil C \rceil) \\ Prj_I(C) &= Obs_I(\lceil C \rceil). \end{aligned}$$ □

The dual result holds as well, i.e., every PA$_P$ state always corresponds to its MSR$_P$ encoding:

Lemma 2. *Let be Q a* PA$_P$ *state. Then $\lfloor Q \rfloor \bowtie Q$.*

Proof. The proof follows considering similar argument of Lemma 1. □

On the basis of these concepts, we can now define a relation between MSR$_P$ configurations and PA$_P$ states, a form of weak bisimulation we call *correspondence*, such that if in MSR$_P$ is possible to perform an action (by applying a rule) that will lead to a new configuration, then in PA$_P$ is possible to follow some transitions that will lead in a corresponding state, and vice versa.

Definition 5. Let $\mathcal{C}$ and $\mathcal{Q}$ be the set of all MSR$_P$ configurations and PA$_P$ states, respectively. We call *correspondence* the largest relation $\sim \subseteq \mathcal{C} \times \mathcal{Q}$ satisfying the following conditions: for all $(\tilde{r} : \tilde{s}) \sim Q$

1. $(\tilde{r} : \tilde{s}) \bowtie Q$;
2. if $\tilde{r} : \tilde{s} \longrightarrow \tilde{s}'$, then $Q \Rightarrow^* Q'$ and $(\tilde{r} : \tilde{s}') \sim Q'$;
3. if $Q \Rightarrow Q'$, then $\tilde{r} : \tilde{s} \longrightarrow^* \tilde{s}'$ and $(\tilde{r} : \tilde{s}') \sim Q'$.

We say $(\tilde{r} : \tilde{s})$ and Q are *correspondent* is there exists a correspondence $\sim$ such that $(\tilde{r} : \tilde{s}) \sim Q$.

The following theorems affirm that there is a correspondence between security protocol specifications written in MSR$_P$ and PA$_P$ when related via the encodings here presented.

Theorem 1. *Given an* MSR$_P$ *security protocol theory C. Then $C \sim \lceil C \rceil$.*

Proof. See Appendix A. □

Theorem 2. *Given an* PA$_P$ *security protocol process* Q. *Then* $\lfloor Q \rfloor \sim Q$.

Proof. See Appendix A. □

This means that any MSR$_P$ step can be faithfully simulated by zero or more steps in PA$_P$ through the mediation of the encoding $\lceil\_\rceil$, and vice versa, the reverse translation $\lfloor\_\rfloor$ will map steps in PA$_P$ into corresponding steps in MSR$_P$.

We conclude by observing that our encodings and Theorem 1 and 2 allow us to reason about security properties in one of either frameworks and transfer the results to the other. In the next section, we explore this idea relative to secrecy and authentication.

6. Security analysis

This section shows how our encodings preserve some security properties from one formalisms to the other: precisely those security properties whose definitions can be expressed in terms of predicates over the intruder knowledge or the set of messages on the networks, specifically *secrecy* and *authenticity*.

6.1. Secrecy

A secrecy property requires that a certain message, say M, cannot be discovered by an intruder during any possible interactions with protocol participants. Generally speaking the discovery of a secrecy flaw can be performed by looking for traces where the intruder acquires knowledge of the secret. If no such trace exists, then secrecy is preserved.

In MSR$_P$, the formal definition of such a secrecy violation is straightforwards in our context by using the $\mathit{Prj}_I(\_)$ function:

Definition 6 (Secrecy violation in MSR$_P$). Let be C be an MSR$_P$ configuration of a protocol, and M be a ground message. We say that C *does not preserve the secrecy* of M if and only if

$$\exists C'.\, C \longrightarrow^{*} C' \quad \text{and} \quad M \in \mathit{Prj}_I(C').$$

Definition 6 can often be verified quite efficiently using modern model checking and theorem proving techniques [9,34].

A secrecy flaw is defined similarly in PA$_P$:

Definition 7 (Secrecy violation in PA$_P$). Let Q be a PA$_P$ model of a protocol, and M be a ground message. We say that Q *does not preserve the secrecy of* M if and only if

$$\exists Q',\ Q \Rightarrow^{*} Q',\ \text{and}\ M \in \mathit{Obs}_I(Q').$$

Again, Definition 7 can be efficiently verified by one of the existing strategies for checking secrecy violation or secrecy preservation developed for process algebras, e.g., using reachability analysis techniques [6,19].

The main fact here is that, independently from the checking strategy chosen, our correspondence relation preserves secrecy. Indeed, the intruder knowledge in two corresponding models, an MSR$_P$ configuration and a PA$_P$ state respectively, is the same step by step. So whenever there is a computation that leads the intruder to discover a secret M in the MSR$_P$ model, there shall be a computation in the PA$_P$ model where the intruder is able to capture the same message. Then, by producing corresponding models, our encodings are able to map secrecy properties from MSR$_P$ to PA$_P$ and vice versa. In fact:

Proposition 3. *Let be C an* MSR$_P$ *configuration and M a ground message. Then*

$$M \in \mathit{Prj}_I(C) \text{ iff } M \in \mathit{Obs}_I(\lceil C \rceil).$$

Proof. Straightforward by Theorem 1. □

Proposition 4. *Let be Q a* PA$_P$ *state and M a ground message. Then*

$$M \in \mathit{Obs}_I(Q) \text{ iff } M \in \mathit{Prj}_I(\lfloor Q \rfloor).$$

Proof. Straightforward by Theorem 2. □

The obvious conclusion is that secrecy is preserved by our encodings.

Theorem 3. *Let be C an* MSR$_P$ *model of a protocol (i.e., an initial* MSR$_P$ *configuration). Then for any message M, a secrecy violation (w.r.t M) happens in C if and only if a secrecy violation (w.r.t. M) happens in $\lceil C \rceil$.*

Proof. Straightforward by Theorem 1 and Proposition 3. □

Theorem 4. *Let be Let be Q a* PA$_P$ *model of a protocol (i.e., an initial* PA$_P$ *state). Then for any message M, a secrecy violation (w.r.t. M) happens in Q if and only if a secrecy violation (w.r.t. M) happens in $\lfloor Q \rfloor$.*

Proof. Straightforward by Theorem 2 and Proposition 4. □

6.2. Authentication

The treatment of authentication properties is a bit more intricate. There are several notions of authentication. One of the most popular techniques was introduced by Woo and Lam [38]. Roles are annotated with unforgeable control actions called *assertions* that describe the state of the protocol execution from the point of view of the principal executing it: for example the initiator may use $\mathsf{begin}(L)$ to assert that the protocol has started, while the responder may assert $\mathsf{end}(L)$ when it reaches its last event. The label L uniquely identifies relevant parameters of this session (the principals involved, their role, nonces, etc.).

Generally speaking, if a protocol guarantees authentication, then in every run each $\mathsf{end}(L)$ event matches a distinct $\mathsf{begin}(L)$ event proceeding it, even in the presence of an attacker. If this is the case, we know that the initiator and the responder have a compatible view of the world. If we abstract a run as the sequence of assertions issued by all parties, this is equivalent [27] to checking that in each run the number of $\mathsf{end}(L)$ never exceeds the number of $\mathsf{begin}(L)$, for the same L.

Definition 8. A protocol P satisfies *authenticity* if and only if for every run of the protocol and for every L, the number of $\mathsf{end}(L)$ events never exceeds the number of $\mathsf{begin}(L)$ events.

We show how this mechanism works for detecting Lowe's attack on the *NSPK* protocol [25]. Consider that when one user A starts to run the protocol as initiator apparently with a responder B, it sends a control message $\mathsf{begin}(\langle A,\ B\rangle)$. When one user B running the role of responder finishes a protocol apparently with an initiator A running the role of initiator then it sends the message $\mathsf{end}(\langle A,\ B\rangle)$. Ideally, if we assume that these messages are never removed from the net, the number of messages of the form $\mathsf{begin}(\langle A,\ B\rangle)$ must be greater than the number of messages of the form $\mathsf{end}(\langle A,\ B\rangle)$ at any point of any computation.

The attack is given by the following sequence of actions. We only need three users: A, B and E such that A initiates a run with a dishonest principal E who reroute it as a run with B. We write $E(A)$ to denote the intruder impersonating the agent A:

$$\begin{array}{lll}
A & \longrightarrow E & : \{N_A, A\}_{K_E} \\
E(A) & \longrightarrow B & : \{N_A, A\}_{K_B} \\
B & \longrightarrow E(A) & : \{N_A, N_B\}_{K_A} \\
E & \longrightarrow A & : \{N_A, N_B\}_{K_A} \\
A & \longrightarrow E & : \{N_B\}_{K_E} \\
E(A) & \longrightarrow B & : \{N_B\}_{K_B}
\end{array}$$

Principal A starts a run of the protocol with the dishonest agent E, who decrypts the transmitted values and repackages them as if they were intended for principal B. Agent B, believing he is responding to A, sends the message $\{N_A, N_B\}_{K_A}$ to E, who simply forwards it to A. This principal replies to E with the last message $\{N_A, N_B\}_{K_A}$, that E repackages for B as earlier. In the end, A correctly believes she has authenticated E, but B incorrectly assumes he has authenticated A while he was talking to E only. Woo and Lam's method reveals this failure of authentication: if we start the initiator role with the assertion $\mathsf{begin}(\langle A,\ B\rangle)$ and conclude the responder role with $\mathsf{end}(\langle A,\ B\rangle)$, we extract from the above run the trace

$\{\mathsf{begin}(\langle A,\, E\rangle), \mathsf{end}(\langle A,\, B\rangle)\}$, which violates Definition 8. While this method may seem rather simple it has been shown very useful for detecting attacks on security protocols (e.g., see [26]).

A possible solution to include authenticity in our framework comes from the observation that it is possible to encode begin-end assertions through particular control messages in such a way that the observational power of our correspondence relation is enough. Since our correspondence relation "observes" only the status of the net and of the intruder knowledge, this implies that we have to find a way to record the begin-end events in either the intruder knowledge or in the network. Moreover because our notion of observation concerns sets we must face the problem of losing the number of repetitions of events in sets. Both problem can be easily solved (e.g., see [27]).

The latter one, for example can be solved by introducing in each control message information that makes it unique e.g., a timestamp. This information is then filtered out when used to check related begin-end events.

To solve the former problem we will develop a different strategy that consists in sending begin-end assertions over a *private network*, we call N^P. The goal of this private network is only to collect control messages for sake of verification. Moreover we assume assertions be coded as control messages $\langle \mathsf{begin},\, L\rangle$, $\langle \mathsf{end},\, L\rangle$, where the label L carries sufficient information for uniquely identify the session. Moreover we assume that L carries timestamp information that make them unique in different run of the protocol.

In MSR$_P$ to model such a network we need a new predicate N^P. A role may assert something by sending a control message over N^P. This can be done, for example, by using the send rewriting rule. This requires a new class of *assertion rules*, similar to send rules:

$$\textit{assertion rule} \qquad A_{\rho_{i-1}}(\boldsymbol{x}) \to A_{\rho_i}(\boldsymbol{x}), N^P(\langle a,\, L(\boldsymbol{x})\rangle)$$

where $a \in \{\mathsf{begin}, \mathsf{end}\}$.

In PA$_P$ the private network N^P is modeled by the process $!N_i^P(x).\overline{N_o^P}(x).0$, while a process's assertion is modeled by sending a message, of form either $\langle \mathsf{begin},\, L(\boldsymbol{x})\rangle$ or $\langle \mathsf{end},\, L(\boldsymbol{x})\rangle$, towards the channel N_i^P. We deal with authentication by slightly modifying our encodings to take into account the new symbols N^P. The correspondence relation needs to be modified too. We handle N^P by simply mirroring the treatment of N.

We can now define our instances of Definition 8 as in the following.

Definition 9 (Authenticity violation in MSR$_P$). Let be C be an MSR$_P$ model of a protocol (i.e., an initial configuration). We say that C *violates authenticity* if and only if for some L, $\exists C', C \longrightarrow^{*} C'$, such that in $\mathit{Prj}_{N^P}(C')$ the number of $\langle \mathsf{end},\, L\rangle$ is greater of the number of $\langle \mathsf{begin},\, L\rangle$.

If it is the case will write $C \not\models \{\mathsf{end}(L) \hookrightarrow \mathsf{begin}(L)\}$.

Definition 10 (Authenticity violation in PA$_P$). Let be Q be a PA$_P$ model of a protocol. We say that Q *violates authenticity* if and only if for some $L, \exists Q', Q \Rightarrow^* Q'$ such that in $\mathit{Obs}_{N^P}(Q')$ the number of $\langle \mathsf{end},\ L\rangle$ if greater of the number of $\langle \mathsf{begin},\ L\rangle$.

If it is the case will write $Q \not\models \{\mathsf{end}(L) \hookrightarrow \mathsf{begin}(L)\}$.

All the results stated in Section 5, remain valid. Precisely because the messages lying on the network in two correspondent models, resp., an MSR$_P$ and a PA$_P$, are the same step by step if there is a computation that leads to a authenticity flaw in the MSR$_P$ model, there would be another computation in the PA$_P$ model where the same flaw is shown, and vice versa. Then our encodings, mapping models into correspondent models, are able to map authenticity properties from MSR to PA and vice versa. The previous results can be formalized into the following propositions

Proposition 5. *Let be C an* MSR$_P$ *model of a protocol and L a ground control message. Then $C \not\models \{\mathsf{end}(L) \hookrightarrow \mathsf{begin}(L)\}$ iff $\lceil C \rceil \not\models \{\mathsf{end}(L) \hookrightarrow \mathsf{begin}(L)\}$.*

Proof. Straightforward by Theorem 1. □

Proposition 6. *Let be Q a* PA$_P$ *model of a protocol and L a ground control message. Then $Q \not\models \{\mathsf{end}(L) \hookrightarrow \mathsf{begin}(L)\}$ iff $\lfloor Q \rfloor \not\models \{\mathsf{end}(L) \hookrightarrow \mathsf{begin}(L)\}$.*

Proof. Straightforward by Theorem 2. □

The obvious conclusion is that authenticity is reserved by our encodings.

Theorem 5. *Let be Let be C an* MSR$_P$ *model of a protocol. Then C preserves authenticity if and only if $\lceil C \rceil$ does.*

Proof. Straightforward by Theorem 1 and Proposition 5. □

Theorem 6. *Let be Let be Q a* PA$_P$ *model of a protocol. Then Q preserves authenticity if and only if $\lfloor Q \rfloor$ does.*

Proof. Straightforward by Theorem 2 and Proposition 6. □

7. Conclusions

This paper shows how multiset rewriting theories (MSR) and process algebras (PA) used to describe security protocols may be related. Indeed we show how to define transformations between MSR and PA describing protocols, and we prove their semantics (based on labeled transition systems) to be related. The paper introduces a correspondence relation based on what messages appear on the network and on what messages the intruder knows. A direct consequence of this results is that many security property established in one framework can automatically be ported to the other.

Acknowledgement

S. Bistarelli was partially supported by MIUR project "Constraint Based Verification of Reactive Systems" (COVER), by the MIUR project "Network Aware Programming: Object, Languages, Implementation" (NAPOLI), and the project "SeTAPS". I. Cervesato was partially supported by NRL under contract N00173-00-C-2086. G. Lenzini was supported by the MIUR-CNR Project SP4, and partially by the project "Privacy in an Ambient World" a TUD/DIES/KUN/TNO-EIB/TNO-FEL collaboration funded by IOP GenCom under project nr. IGC-03001-IOP. F. Martinelli was partially supported by MIUR project "Constraint Based Verification of Reactive Systems" (COVER), by MIUR project "MEFISTO", by Microsoft Research and by the CSP project "SeTAPS II".

We would like to thank the selection committee and attendees of the WITS'03 workshop where a preliminary version of this paper was presented [5]. There we had stimulating discussions there led to the present revision.

Finally we thank our anonymous referees whose feedbacks gave us precious suggestions for improving the presentation of the present work.

Appendix A. Proofs

This appendix provides a proof for Theorem 1 and a proof for Theorem 2.

We begin this section by reminding that a MSR$_P$ state is a multiset of form $\tilde{s} = (\tilde{N}, \tilde{A}, \tilde{I}, \tilde{\pi})$, where the components collect ground facts $N(t)$, $A_{\rho_i}(\boldsymbol{t})$, $I(t)$ and $\pi(\boldsymbol{t})$ respectively, while a PA_P state is a process (see Proposition 1)

$$\overbrace{(P_{!net} \parallel \prod_{\rho} P_{!\rho} \parallel Q_{!I} \parallel Q_{!\pi})}^{Q_!} \parallel (Q_{net} \parallel \prod_{\rho} P_{\rho} \parallel Q_I \parallel Q_{rem})$$

where:

$$\begin{array}{ll} Q_{net} & ::= 0 \mid \prod \overline{N_o}(t).0 \\ P_\rho & ::= 0 \mid N_o(\boldsymbol{x}).P_\rho \quad \mid \quad \overline{N_i}(\underline{\boldsymbol{t}}).P_\rho \quad \mid \quad [\underline{\boldsymbol{t}} = \boldsymbol{t'}]\, P_\rho \\ Q_I & ::= \text{suffixes of } P_{I_j}, \text{ for all } j \\ Q_{rem} & ::= 0 \quad \mid \quad N_o(x).\overline{N_i}(x).0 \quad \mid \quad \tilde{\pi}(\boldsymbol{x}).\nu\boldsymbol{n}.P_\rho \quad \mid \quad \tilde{\pi}(\boldsymbol{x}).\nu\boldsymbol{n}.P_\rho \quad \mid \quad \prod \tilde{\pi}(\underline{t}).0 \end{array}$$

Moreover in the following we will use implicitly the following proposition:

Proposition 7. $\lfloor !P \parallel P \parallel Q \rfloor = \lfloor !P \parallel Q \rfloor$.

Proof. It is based on the fact that $\lfloor \_ \rfloor$ maps processes P, coming from any transition $!P \Rightarrow P \parallel !P$, into the empty multiset. Formally:

$$\lfloor !P \parallel P \parallel Q \rfloor = \lfloor !P \rfloor, \lfloor P \rfloor, \lfloor Q \rfloor = \lfloor !P \rfloor, \cdot, \lfloor Q \rfloor = \lfloor !P \rfloor, \lfloor Q \rfloor = \lfloor !P \parallel Q \rfloor \quad \square$$

We now prove the following main theorem:

Theorem 1 (Reminder). *Given an* MSR$_P$ *security protocol theory* C. *Then* $C \sim \lceil C \rceil$.

Proof. The proof consists in showing that

$$\mathcal{R} = \{(C, \lceil C \rceil) : C_0 \longrightarrow^* C\} \cup \{(C, Q) : C_0 \longrightarrow^* C, \lfloor Q \rfloor = C\}$$

is a correspondence $\sim$. Specifically, because of Lemma 1 and Lemma 2 it is sufficient to show that for all $(C, Q) \in \mathcal{R}$:

(I) $C \longrightarrow C'$ implies $Q \Rightarrow^* Q'$ and $(C', Q') \in \mathcal{R}$

(II) $Q \Rightarrow Q'$ implies $C \longrightarrow^* C'$ and $(C', Q') \in \mathcal{R}$.

Precisely $(C', Q') \in \mathcal{R}$ means that either $\lfloor Q' \rfloor = C'$ or $Q' = \lceil C' \rceil$.

Before explaining the technical steps of the proof, let us focus on the following question. What are the $(C', Q') \in \mathcal{R}$ that are reachable via a MSR$_P$ or PA$_P$ transition from $(C, Q) \in \mathcal{R}$? In other words, given a transition $C \longrightarrow C'$ (resp., $Q \Rightarrow Q'$) what transitions $\lceil C \rceil \Rightarrow^* Q'$ or $Q \Rightarrow^* Q'$ where $\lfloor Q \rfloor = C$ (resp., $\lfloor Q \rfloor \longrightarrow^* C'$ or $C \longrightarrow^* C'$ where $\lceil C \rceil = Q$) satisfy condition (**I**) (resp., condition (**II**)) above?

Let us first focus on **(I)** and on Fig. 2 and suppose that a MSR$_P$ transition $C \longrightarrow C'$ triggers. Via $\lceil \_ \rceil$ the only possible PA$_P$ transition is $\lceil C \rceil \Rightarrow^* \lceil C' \rceil$ (e.g., states Q and Q' and the relative $Q \Rightarrow^* Q'$ transition in Fig. 2). Instead via $\lfloor \_ \rfloor$, more transitions $Q \Rightarrow^* Q'$ are possible; precisely all those such that $\lfloor Q \rfloor = C$ and $\lfloor Q' \rfloor = C'$ (e.g., processes Q''', Q and Q'' in Fig. 2 and transitions $Q''' \Rightarrow^* Q'$, $Q \Rightarrow^* Q'$ and $Q'' \Rightarrow^* Q'$).

Let now focus on **(II)** and on Fig. 2 again. Let suppose a PA$_P$ transition $Q \Rightarrow Q'$ triggers. Here it may be that the only couple (C', Q') corresponding in $\mathcal{R}$, via either $\lfloor \_ \rfloor$ or $\lceil \_ \rceil$, to (C, Q) is such that $C = C'$. This happens when transition $Q \Rightarrow Q'$ is not able to simulate any complete MSR$_P$ step (e.g., as the transition $Q \Rightarrow Q''$ and its correspondent $C \longrightarrow^* C$, in Fig. 2).

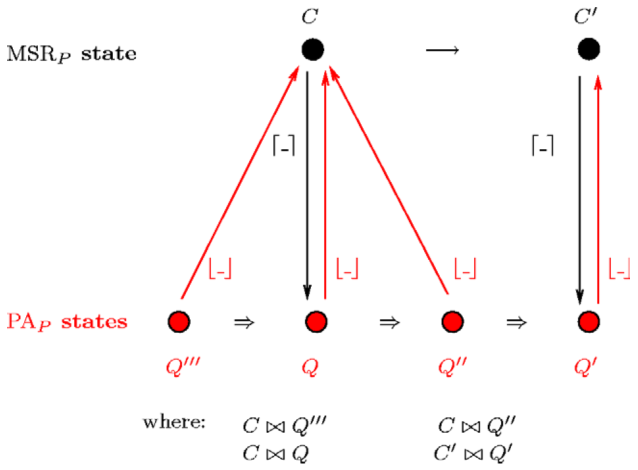

Fig. 2. A possible scenario of corresponding couples (C, Q) and (C', Q') in $\mathcal{R}$ when either a transition $C \longrightarrow C'$ or a transition $Q \Rightarrow Q'$, triggers.

Proof of Part (I). The scheme which guides the proof of this part, is the following:

$$\textbf{(I)} \quad C \longrightarrow C' \text{ implies } \begin{array}{l} (a)\ \lceil C \rceil \Rightarrow^* Q' \text{ and } (C', Q') \in \mathcal{R} \\ (b)\ \forall Q : \lfloor Q \rfloor = C, Q \Rightarrow^* Q' \text{ and } (C', Q') \in \mathcal{R} \end{array} \qquad (4)$$

In the following we will itemize each sub-case with $(I.a)$, $(I.a')$, etc., or $(I.b)$ $(I.b')$, etc., depending on it is respectively the first, second, etc., sub-case of branches (a) or (b) of (4); moreover let us observe that, because $\lfloor \lceil C \rceil \rfloor = C$ (see Lemma 2)

$$\{(C, \lceil C \rceil) : C_0 \longrightarrow^* C\} \cap \{(C, Q) : C_0 \longrightarrow^* C, \lfloor Q \rfloor = C\} \neq \emptyset$$

As a consequence some sub-cases of (b) will coincide with some sub-case of (a). Precisely those that do really differ, are those involving pairs $(\lfloor Q \rfloor, Q)$ such that $Q \neq \lceil C \rceil$; to avoid repetitions we will treat in $(I.b)$ only those cases that differ from cases in $(I.a)$.

Let be C' such that $C \longrightarrow C'$. It must have happened as a consequence of an application of either a rewriting rule r_{ρ_0}, r_{ρ_i} send or r_{ρ_i} receive or r_{ρ_i} analysis for $i > 2, \ldots, l_\rho$ or finally an intruder rule r_{I_j} for $j = 0, \ldots, 9$. We will treat each rule separately. We also remind that for each rule we will list different sub-cases $(I.a)$ and $(I.b)$.

- $\boxed{\textbf{(instantiation rule) } r_{\rho_0} = \tilde{\pi}(\boldsymbol{x}) \rightarrow \exists \boldsymbol{n}.A_{\rho_0}(\boldsymbol{n}, \boldsymbol{x}), \tilde{\pi}(\boldsymbol{x})}$

In this case transition $C \longrightarrow C'$ can be specifically rewritten as:

$$\begin{aligned}
C = {} & \tilde{\pi}(\underline{\boldsymbol{k}}), C'' \\
\rightarrow {} & A_{\rho_0}[\overbrace{\underline{\boldsymbol{k}}/\boldsymbol{x}; \underline{\boldsymbol{m}}/\boldsymbol{n}}^{\theta}], \tilde{\pi}(\underline{\boldsymbol{k}}), C'' \\
= {} & \underbrace{A_{\rho_0}(\underline{\boldsymbol{k}}; \underline{\boldsymbol{m}}), C}_{C'}
\end{aligned}$$

where, we remind, $\tilde{\pi}(\underline{\boldsymbol{k}})$ is a shortcut for $\pi(\underline{\boldsymbol{k}}_1), \ldots, \pi(\underline{\boldsymbol{k}}_r)$ where $\underline{\boldsymbol{k}}_i$ for all i, are all ground tuples of terms.
$\star$ Case $(I.a)$: $(C, Q) = (C, \lceil C \rceil)$. We have:

$$\begin{aligned}
\lceil C \rceil = {} & \overbrace{!\overline{\tilde{\pi}}(\underline{\boldsymbol{k}}).0}^{\lceil \tilde{\pi}(\underline{\boldsymbol{k}}) \rceil} \parallel \underbrace{\overbrace{!\tilde{\pi}(\boldsymbol{x}).\nu\boldsymbol{n}.\quad \overbrace{P_\rho}^{\lceil r_{\rho_1} \rceil^{\#}_{(\boldsymbol{x};\boldsymbol{n})}}}^{\lceil r_{\rho_0} \rceil} \quad \parallel Q''}_{\lceil C'' \rceil} && [\text{def. of } \lceil \_ \rceil] \\
\equiv {} & \overline{\tilde{\pi}}(\underline{\boldsymbol{k}}).0 \parallel \tilde{\pi}(\boldsymbol{x}).\nu\boldsymbol{n}.P_\rho \parallel \underbrace{!\overline{\tilde{\pi}}(\underline{\boldsymbol{k}}).0 \parallel !\tilde{\pi}(\boldsymbol{x}).\nu\boldsymbol{n}.P_\rho \parallel Q''}_{\lceil C \rceil} \\
\Rightarrow^* {} & \underbrace{0 \parallel P_\rho[\theta] \parallel \lceil C \rceil}_{Q'} && [pa_0, pa_\equiv, pa_\nu] \\
= {} & 0 \parallel \lceil r_{\rho_1} \rceil^{\#}_{(\boldsymbol{x};\boldsymbol{n})}[\theta] \parallel \lceil C \rceil \\
= {} & 0 \parallel \lceil A_{\rho_0}(\underline{\boldsymbol{k}}; \underline{\boldsymbol{m}}) \rceil \parallel \lceil C \rceil && [\text{def. of } \lceil A_{\rho_i}(\boldsymbol{t}) \rceil] \\
\equiv {} & \lceil A_{\rho_0}(\underline{\boldsymbol{k}}; \underline{\boldsymbol{m}}) \rceil \parallel \lceil C \rceil \\
= {} & \lceil C' \rceil
\end{aligned}$$

$\star$ Case $(I.b)$: $(C, Q) = (\lfloor Q \rfloor, Q)$. We need to identify those Q's such that $\lfloor Q \rfloor = C = \tilde{\pi}(\underline{\boldsymbol{k}}), C''$. The only different case, w.r.t. $(I.a)$, (indeed a family of cases) happen when

$$Q = \Big(\prod_{i=m}^{r} \overline{\pi}(\underline{k}_i).0\Big) \parallel \pi_m(x_m). \cdots .\pi_r(x_r).\nu\boldsymbol{n}.P_\rho[\theta'] \parallel \lceil C \rceil$$

where $\theta' = [\underline{k}_1/x_1, \ldots, \underline{k}_{m-1}/x_{m-1}]$. In words, Q is a partially instantiated role that has already started receiving its permanent terms, but not all. It is worth to underline that both $\prod_{i=m,\ldots,r} \overline{\pi}(\underline{k}_i).0$ and $\pi_m(x_m). \cdots .\pi_r(x_r).\nu\boldsymbol{n}.P_\rho[\theta']$ are mapped by $\lfloor \_ \rfloor$ into the empty multiset; as a consequence $\lfloor Q \rfloor = C$. Let now observe that:

$$\begin{array}{rll} & \left(\prod_{i=m,\dots,r} \overline{\pi}(\underline{k}_i).0\right) \parallel & \\ Q = & \pi_m(x_m).\cdots.\pi_r(x_r).\nu\boldsymbol{n}.P_\rho[\theta'] & \\ & \parallel \lceil C \rceil & \\ \Rightarrow^* & 0 \parallel P_\rho[\theta] \parallel \lceil C \rceil & [pa_0 \text{ and } pa_\nu \text{ with } \underline{\boldsymbol{m}} \text{ as new names}] \\ \equiv & \underbrace{P_\rho[\theta] \parallel \lceil C \rceil}_{Q'} & \end{array}$$

and it easy to check that $\lceil C' \rceil = Q'$.

- $\boxed{\textbf{(send rule) } r_{\rho_i} = A_{\rho_{i-1}}(\boldsymbol{x}) \to A_{\rho_i}(\boldsymbol{x}), N(t(\boldsymbol{x}))}$

  In this case transition $C \longrightarrow C'$ can be specifically rewritten as:

$$C = A_{\rho_{i-1}}(\underline{\boldsymbol{x}[\theta]}), C'' \to \underbrace{A_{\rho_i}(\underline{\boldsymbol{x}[\theta]}), N(\underline{t[\theta]}), C''}_{C'} \tag{5}$$

where θ is the substitution that allows the rule r_{ρ_i} to be applied. The only significative situation happens as a sub-case of statement (a) of (4).
⋆ Case $(I.a)$: $(C, Q) = (C, \lceil C \rceil)$. We have:

$$\begin{array}{rll} \lceil C \rceil = & \lceil r_{\rho_i} \rceil^{\#}_{(\boldsymbol{x})}[\theta] \parallel \lceil C'' \rceil & [\text{def. of } \lceil A_{\rho_{i-1}}(\underline{\boldsymbol{x}[\theta]}) \rceil] \\ = & \overline{N_i}(t[\theta]).\lceil r_{\rho_{i+1}} \rceil^{\#}_{(\boldsymbol{x})}[\theta] \parallel \lceil C'' \rceil & [\text{unfolding } \lceil r_{\rho_i} \rceil^{\#}_{(\boldsymbol{x})}[\theta]] \\ & \overline{N_i}(\underline{t[\theta]}).\lceil r_{\rho_{i+1}} \rceil^{\#}_{(\boldsymbol{x})}[\theta] \parallel & \\ = & \underbrace{!N_i(x).\overline{N_o}(x).0 \parallel \lceil C''' \rceil}_{\lceil C'' \rceil} & [\text{def. of } P_{!net} \text{ in } \lceil C'' \rceil] \\ & \overline{N_i}(\underline{t[\theta]}).\lceil r_{\rho_{i+1}} \rceil^{\#}_{(\boldsymbol{x})}[\theta] & \\ \equiv & \parallel N_i(x).\overline{N_o}(x).0 \parallel & \\ & \underbrace{!N_i(x).\overline{N_o}(x).0 \parallel \lceil C''' \rceil}_{\lceil C'' \rceil} & \\ \Rightarrow & \underbrace{\overbrace{\lceil r_{\rho_{i+1}} \rceil^{\#}_{(\boldsymbol{x})}[\theta]}^{\lceil A_{\rho_i}(\underline{\boldsymbol{x}[\theta]}) \rceil} \parallel \overbrace{\overline{N_o}(\underline{t[\theta]}).0}^{\lceil N(\underline{t[\theta]}) \rceil} \parallel \lceil C'' \rceil}_{Q'} & [\text{def. of } pa_0] \\ = & \lceil C' \rceil & \end{array}$$

- $\boxed{\textbf{(receive rule) } r_{\rho_i} = A_{\rho_{i-1}}(\boldsymbol{x}), N(y) \longrightarrow A_{\rho_i}(\boldsymbol{x}; y)}$

  In this case transition $C \longrightarrow C'$ can be specifically rewritten as:

$$C = A_{\rho_{i-1}}(\underline{\boldsymbol{x}[\theta]}), N(\underline{t}), C'' \longrightarrow \underbrace{A_{\rho_i}(\underline{\boldsymbol{x}[\theta]}; y[\underline{t}/y]), C''}_{C'} \tag{6}$$

where θ is the substitution that allows the rule r_{ρ_i} to be applied. Again the only significative case happens as a sub-case of class (a) in statement (4).
$\star$ Case $(I.a)$: $(C, Q) = (C, \lceil C \rceil)$. We have:

$$\begin{aligned} \lceil C \rceil &= \lceil r_{\rho_i} \rceil^{\#}_{(\boldsymbol{x})}[\theta] \parallel \overline{N_o}(\underline{t}).0 \parallel \lceil C'' \rceil && [\text{def. of } \lceil A_{\rho_{i-1}}(\underline{\boldsymbol{x}[\theta]}) \rceil] \\ &= N_o(y).\lceil r_{\rho_{i+1}} \rceil^{\#}_{(\boldsymbol{x};y)}[\theta] \parallel \overline{N_o}(\underline{t}).0 \parallel \lceil C'' \rceil && [\text{espanding } \lceil r_{\rho_i} \rceil^{\#}_{(\boldsymbol{x})}[\theta]] \\ &\Rightarrow \underbrace{\lceil r_{\rho_{i+1}} \rceil^{\#}_{(\boldsymbol{x};y)}[\theta][\underline{t}/y] \parallel 0 \parallel \lceil C'' \rceil}_{Q'} && [pa_0] \\ &= \lceil C' \rceil \end{aligned}$$

- $\boxed{\textbf{(analysis rule) } r_{\rho_i} = A_{\rho_{i-1}}(\boldsymbol{t}(\boldsymbol{x})) \longrightarrow A_{\rho_i}(\boldsymbol{x}).}$

  In this case transition $C \longrightarrow C'$ can be specifically rewritten as:

$$C = A_{\rho_{i-1}}(\underline{\boldsymbol{t}(\boldsymbol{x})[\theta']}), C'' \longrightarrow \underbrace{A_{\rho_i}(\underline{\boldsymbol{x}[\theta']}), C''}_{C'} \tag{7}$$

Again the only interesting scenario comes from sub-case (a) of (4). While analyzing this case let us:

- rewrite the ground term $\underline{\boldsymbol{t}(\boldsymbol{x})[\theta']}$ as $\underline{\boldsymbol{k}}$;
- assume that the consequent predicate of rule $r_{\rho_{i-1}}$ is $A_{\rho_{i-1}}(\boldsymbol{x}')$, i.e., rule $r_{\rho_{i-1}} = \cdots \longrightarrow A_{\rho_{i-1}}(\boldsymbol{x}')$.
- assume θ be the unifier such that $\boldsymbol{x}'[\theta] = \underline{\boldsymbol{k}}$, that is the substitution that unifies the predicate $A_{\rho_{i-1}}(\boldsymbol{x}')$ with the ground predicate $A_{\rho_{i-1}}(\underline{\boldsymbol{k}})$ in the MSR$_P$ state C.

$\star$ Case $(I.a)$: $(C, Q) = (C, \lceil C \rceil)$. We have:

$$\begin{aligned} \lceil C \rceil &= \lceil r_{\rho_i} \rceil^{\#}_{(\boldsymbol{x}')}[\theta] \parallel \lceil C'' \rceil \\ &= [\overbrace{\boldsymbol{x}'[\theta]}^{\underline{\boldsymbol{k}}} = \boldsymbol{t}(\boldsymbol{x})[\theta]].\lceil r_{\rho_{i+1}} \rceil^{\#}_{(\boldsymbol{x})}[\theta] \parallel \lceil C'' \rceil \ [\text{def. of } \lceil r_{\rho_i} \rceil^{\#}_{(\boldsymbol{x}')}] \end{aligned}$$

$$
\begin{aligned}
&\Rightarrow \lceil r_{\rho_{i+1}}\rceil^{\#}_{(\boldsymbol{x})}[\theta][\theta''] \parallel \lceil C''\rceil \;\; [pa_{[]},\ \text{and } \boldsymbol{t}(\boldsymbol{x})[\theta][\theta''] = \underline{\boldsymbol{k}}]\\
&= \underbrace{\lceil r_{\rho_{i+1}}\rceil^{\#}_{(\boldsymbol{x})}[\theta'] \parallel \lceil C''\rceil}_{Q'} \qquad [\text{(see text below)}]\\
&= \lceil C'\rceil
\end{aligned}
$$

Note that here, θ' can be used instead of $\theta\theta''$ because θ' and $\theta\theta''$ coincide on $\boldsymbol{x}$, that in turn are all the variables appearing in $\lceil r_{\rho_{i+1}}\rceil$.

- $\boxed{\textbf{(intruder rules) } r_{I_j}, \text{ for } j = 0, \ldots, 9.}$

Let us consider just a significative rule, for example rule $r_{I_6} = I(x_1), I(x_2) \rightarrow I(\langle x_1,\, x_2\rangle), I(x_1), I(x_2)$. The proofs for the other intruder's rules are similar. In this case transition $C \longrightarrow C'$ can be specifically rewritten as:

$$
C = I(\underline{t_1}), I(\underline{t_2}), C'' \longrightarrow \underbrace{I(\langle \underline{t_1},\, \underline{t_2}\rangle), I(\underline{t_1}), I(\underline{t_2}), C''}_{C'} . \tag{8}
$$

$\star$ Case $(I.a)$: $(C, Q) = (C, \lceil C\rceil)$. Then we have:

$$
\begin{aligned}
\lceil C\rceil &= \overbrace{\bar{I}(\underline{t_1}).0}^{\lceil I(\underline{t_1})\rceil^{I}} \parallel \overbrace{\bar{I}(\underline{t_2}).0}^{\lceil I(\underline{t_2})\rceil^{I}} \parallel \lceil C''\rceil && [\text{def. of } \lceil \_ \rceil]\\
&= \bar{I}(\underline{t_1}).0 \parallel \bar{I}(\underline{t_2}).0 \parallel \underbrace{Q_{!I} \parallel Q''}_{\lceil C''\rceil} && [\text{expanding } PA_P \text{ state}]\\
&\equiv \begin{array}{l} \bar{I}(\underline{t_1}).0 \parallel \bar{I}(\underline{t_2}).0\\ \parallel I(x_1).\bar{I}(x_1).I(x_2).\bar{I}(x_2).\bar{I}(\langle x_1,\, x_2\rangle).0\\ \parallel I(x).\bar{I}(x).0 \parallel I(x).\bar{I}(x).0\\ \parallel \lceil C''\rceil \end{array} && [\text{expanding } Q_{!I} \ (P_{I_6} \text{ and } P_{I_10})]\\
&\Rightarrow^{*} \underbrace{0 \parallel \overbrace{\bar{I}(\langle \underline{t_1},\, \underline{t_2}\rangle.0)}^{\lceil I(\langle \underline{t_1}, \underline{t_2}\rangle)\rceil^{I}} \parallel \overbrace{\bar{I}(\underline{t_1}).0}^{\lceil I(\underline{t_1})\rceil^{I}} \parallel \overbrace{\bar{I}(\underline{t_2}).0}^{\lceil I(\underline{t_2})\rceil^{I}} \parallel \lceil C''\rceil}_{Q'} && [pa_0]\\
&= \lceil C'\rceil
\end{aligned}
$$

Let now start analyzing the case $(C, Q) = (\lfloor Q\rfloor, Q)$. We need to identify those Q's such that $\lfloor Q\rfloor = I(\underline{t_1}), I(\underline{t_2}), C''$. In fact, more different Q's (precisely different Q_I) exist, for the non injective $\lfloor \_ \rfloor_I$ is now involved in the translation (see also Fig. 1). In addition, we remind, the only really significative (w.r.t. case $(I.a)$) situations are those ones where Q's are such that $Q \neq \lceil C\rceil$.

$\star$ Case $(I.b')$: a first case happens when Q contains both the process $\overline{I}(\underline{t_2}).0$ and the proper suffix of P_{I_6}, $\overline{I}(\underline{t_1}).I(x_2).\overline{I}(x_2).\overline{I}(\langle \underline{t_1},\, x_2\rangle).0$.

$$
\begin{array}{rll}
Q = & \overline{I}(\underline{t_2}).0 \parallel \overline{I}(\underline{t_1}).I(x_2).\overline{I}(x_2).\overline{I}(\langle \underline{t_1},\, x_2\rangle).0 \parallel \lceil C''\rceil & \\
\equiv & \begin{array}{l} \overline{I}(\underline{t_2}).0 \\ \parallel \overline{I}(\underline{t_1}).I(x_2).\overline{I}(x_2).\overline{I}(\langle \underline{t_1},\, x_2\rangle).0 \\ \parallel I(x).\overline{I}(x).0 \parallel I(x).\overline{I}(x).0 \\ \parallel \lceil C''\rceil \end{array} & \begin{array}{l}\text{[espanding} \\ Q_{!I}]\end{array} \\
\Rightarrow^* & \underbrace{0 \parallel \overline{I}(\langle \underline{t_1},\, \underline{t_2}\rangle).0 \parallel \overline{I}(\underline{t_1}).0 \parallel \overline{I}(\underline{t_2}).0 \parallel \lceil C''\rceil}_{Q'} & [pa_0]
\end{array}
$$

and it is easy to verify that $\lfloor Q'\rfloor = C'$.

$\star$ Case $(I.b'')$: a second case happens when Q is $[\langle \underline{t_1},\, \underline{t_2}\rangle = \langle x_1,\, x_2\rangle].\overline{I}(x_1).\overline{I}(x_2).0 \parallel \lceil C''\rceil$. In words Q contains a proper suffix of process P_{I_5}, standing for the intruder that has already acquired the message $\langle \underline{t_1},\, \underline{t_2}\rangle$, but that has not yet performed the output in which it splits it. We remind that in this case $\lfloor \_ \rfloor_I$ translates the process as it would have already performed the outputs, obtaining the predicates $I(\underline{t_1})$, $I(\underline{t_2})$. Then we have:

$$
\begin{array}{rll}
Q = & [\langle \underline{t_1},\, \underline{t_2}\rangle = \langle x_1,\, x_2\rangle].\overline{I}(x_1).\overline{I}(x_2).0 \parallel \lceil C''\rceil & \\
\equiv & \begin{array}{l} [\langle \underline{t_1},\, \underline{t_2}\rangle = \langle x_1,\, x_2\rangle].\overline{I}(x_1).\overline{I}(x_2).0 \\ \parallel I(x).\overline{I}(x).0 \parallel \lceil C''\rceil \end{array} & \begin{array}{l}\text{[from } !P_{I_{10}}; \\ pa_!]\end{array} \\
\Rightarrow & \overline{I}(\underline{t_1}).\overline{I}(\underline{t_2}).0 \parallel I(x).\overline{I}(x).0 \parallel \lceil C''\rceil & [pa_{[]}] \\
\Rightarrow & \overline{I}(\underline{t_2}).0 \parallel \overline{I}(\underline{t_1}).0 \parallel Q'' & [pa_0] \\
\Rightarrow^* & \underbrace{0 \parallel \overline{I}(\langle \underline{t_1},\, \underline{t_2}\rangle).0 \parallel \overline{I}(\underline{t_1}).0 \parallel \overline{I}(\underline{t_2}).0 \parallel Q''}_{Q'} & \text{[see Case } (I.a)]
\end{array}
$$

and it is easy to verify that $\lfloor Q'\rfloor = C'$.

$\star$ Case $(I.b''')$: the last case is when $Q = \overline{I}(\underline{t_1}).\overline{I}(\underline{t_2}).0 \parallel \lceil C''\rceil$, where again a suffix of P_{I_5} is involved. This case is simply a sub-case of the previous one.

Here ends the proof of **(I)**, where we have shown that for every $(C, Q) \in \mathcal{R}$ $C \longrightarrow C'$ implies $Q \Rightarrow^* Q'$, and $(C', Q') \in \mathcal{R}$.

Proof of Part (II). The scheme which guides the proof of this part is the following:

$$\textbf{(II)} \quad \forall (C, Q) \in \mathcal{R}, \qquad Q \Rightarrow Q' \text{ implies } C \longrightarrow^* C' \text{ and } (C', Q') \in \mathcal{R}$$

Because, we remind, $\mathcal{R} = \{(C, \lceil C\rceil) : C_0 \longrightarrow^* C\} \cup \{(C, Q) : C_0 \longrightarrow^* C, \lfloor Q\rfloor = C\}$, the previous statement can be specifically restated as:

$$\begin{aligned}
&\forall (C, Q) \in \mathcal{R}, \\
&(a)\ \lceil C\rceil \Rightarrow Q' \text{ implies } C \longrightarrow^* C' \text{ and } (C', Q') \in \mathcal{R} \qquad (9)\\
&(b)\ \forall Q : \lfloor Q\rfloor = C, Q \Rightarrow Q' \text{ implies } C \longrightarrow^* C' \text{ and } (C', Q') \in \mathcal{R}
\end{aligned}$$

where $(C', Q') \in \mathcal{R}$ means that either $\lfloor Q'\rfloor = C'$ or $Q' = \lceil C'\rceil$. In the following we treat a list of cases. Each case corresponds to a possible ⇒ transition. Again we will itemize each sub-case with $(II.a)$, $(II.a')$, etc., or $(II.b)$ $(II.b')$, etc., depending on it is respectively the first, second, etc., sub-case of branches (a) or (b) of (9).

- $\boxed{(pa_0\textbf{: i.e., communication transition})}$

Reasoning about pa_0, we must distinguish among the name of the channel a involved in the reaction i.e., $a = N_i, N_o, \pi, I$. Let us discuss each case separately.

$(a = N_i)$ Here we treat with transitions that involve channel N_i.

$\star$ Case $(II.a)$: $(C, Q) = (C, \lceil C\rceil)$.

This case may happens when $C = A_{\rho_{i-1}}(\underline{\boldsymbol{x}[\theta]}), C''$ and $r_{\rho_i} : A_{\rho_{i-1}}(\boldsymbol{x}) \longrightarrow A_{\rho_{i-1}}(\boldsymbol{x}), N(t(\boldsymbol{x}))$.

In this case transition $\lceil C\rceil \Rightarrow Q'$ can be specifically rewritten as:

$$\begin{aligned}
\lceil C\rceil &= \overbrace{\overline{N_i}(\underline{t(\boldsymbol{x})[\theta]}).\quad \overbrace{P_\rho[\theta]}^{\lceil r_{\rho_{i+1}}\rceil^{\#}_{(\boldsymbol{x})}[\theta]}}^{A_{\rho_{i-1}}(\underline{\boldsymbol{x}[\theta]})} \quad \parallel \lceil C''\rceil \\
&\equiv \overline{N_i}(\underline{t(\boldsymbol{x})[\theta]}).P_\rho[\theta] \parallel N_i(x).\overline{N}(x).0 \parallel \lceil C''\rceil \text{ [expanding PA}_P\text{ state]} \\
&\Rightarrow \underbrace{P_\rho[\theta] \parallel \overline{N}(\underline{t(\boldsymbol{x})}[\theta]).0 \parallel \lceil C''\rceil}_{Q'}
\end{aligned}$$

Then we have:

$$C = A_{\rho_{i-1}}(\underline{\boldsymbol{x}[\theta]}), C'' \longrightarrow \underbrace{N(\underline{t(\boldsymbol{x})[\theta]}), C''}_{C'}\ [r_{\rho_{i+1}}]$$

and it is easy to check that $\lceil C'\rceil = Q'$.

$\star$ Case $(II.b)$: $(C, Q) = (\lfloor Q\rfloor, Q)$. The only different case in this sub-part happens when $\lfloor Q\rfloor = I(\underline{t}), \lfloor Q''\rfloor$. We observe that a Q producing such a MSR$_P$ state is the following:

$$Q = \overline{I}(\underline{t}).0 \parallel \overline{N_i}(\underline{t}').0 \parallel N_i(x).\overline{N_o}(x).0 \parallel Q''$$

where $\overline{N_i}(\underline{t}).0$ is an intruder partial suffix of $P_{I_4} = I(x).\overline{N_i}(x).0$. We remind that $\overline{N_i}(\underline{t}).0$ and $N_i(x).\overline{N_o}(x).0$ are mapped, by $\lfloor\_\rfloor$, onto the empty multiset. Let observe that transition $Q \Rightarrow Q'$ can be specifically rewritten as:

$$\begin{aligned} Q = {} & \overline{I}(\underline{t}).0 \parallel \overline{N_i}(\underline{t}).0 \parallel N_i(x).\overline{N_o}(x).0 \parallel Q'' \\ \Rightarrow {} & \underbrace{\overline{I}(\underline{t}).0 \parallel 0 \parallel \overline{N_o}(\underline{t}).0 \parallel Q''}_{Q'} \end{aligned}$$

Then we have:

$$\lfloor Q \rfloor = I(\underline{t}), \lfloor Q'' \rfloor \longrightarrow \underbrace{I(\underline{t}), N(\underline{t}), \lfloor Q'' \rfloor}_{C'} \text{ [by } r_{I_4}\text{]}$$

and it is easy to check that $C' = \lfloor Q' \rfloor$.

$(a = N_o)$ Here we treat with transitions that involve channel N_o.

$\star$ Case $(II.a)$: $(C, Q) = (C, \lceil C \rceil)$. This case happens when $C = N(\underline{t})$, $A_{\rho_{i-1}}(\underline{\boldsymbol{x}[\theta]}), C''$ and $r_{\rho_i} : A_{\rho_{i-1}}(\boldsymbol{x}), N(y) \longrightarrow A_{\rho_{i-1}}(\boldsymbol{x}; y)$. In this case transition $\lceil C \rceil \Rightarrow Q'$ can be specifically rewritten as:

$$\begin{aligned} \lceil C \rceil = {} & \overline{N_o}(\underline{t}).0 \parallel \overbrace{N_o(y).\ \overbrace{P_\rho[\theta]}^{\lceil r_{\rho_{i+1}} \rceil^{\#}_{(\boldsymbol{x})}[\theta]}}^{\lceil A_{\rho_{i-1}}(\underline{\boldsymbol{x}[\theta]}) \rceil} \parallel \lceil C'' \rceil \\ \Rightarrow {} & \underbrace{0 \parallel P_\rho[\theta][\underline{t}/y] \parallel \lceil C'' \rceil}_{Q'} \end{aligned}$$

Then we have:

$$C = N(\underline{t}), A_{\rho_{i-1}}(\underline{\boldsymbol{x}[\theta]}), C'' \longrightarrow \underbrace{A_{\rho_{i-1}}(\boldsymbol{x}; y)[\theta][\underline{t}/y], C''}_{C'} \text{ [by } r_{\rho_i}\text{]}$$

and it is easy to check that $C' = \lfloor Q' \rfloor$.

$\star$ Case $(II.a')$. Another case of this class happen when $C = N(\underline{t}), C''$ and $r_{I_3} = N(x) \longrightarrow I(x)$. Let observe that transition $\lceil C \rceil \Rightarrow Q'$ can be specifically rewritten as:

$$\begin{aligned} \lceil C \rceil = {} & \overline{N_o}(\underline{t}).0 \parallel \lceil C'' \rceil \\ \equiv {} & \overline{N_o}(\underline{t}).0 \parallel N_o(x).\overline{I}(x).0 \parallel \lceil C'' \rceil \text{ [expanding } P_{!I}\text{]} \\ \Rightarrow {} & \underbrace{\overline{I}(\underline{t}).0 \parallel \lceil C'' \rceil}_{Q'} \end{aligned}$$

Then we have:

$$C = N(\underline{t}), C'' \longrightarrow \underbrace{I(\underline{t}), C''}_{C'} \text{ [by } r_{I_3}\text{]}$$

and it is easy to check that $C' = \lfloor Q' \rfloor$.

$(a = \pi)$ Here we will treat with transitions that involve channel π's.

$\star$ Case $(II.a)$: $(C, Q) = (C, \lceil C \rceil)$. The only interesting scenario in this sub-case happens when in C no role predicates, w.r.t. a role ρ are yet produced and when $r_{\rho_0} = \tilde{\pi}(\boldsymbol{t}(\boldsymbol{x})) \longrightarrow \exists \boldsymbol{n}.A_{\rho_0}(\boldsymbol{x}; \boldsymbol{n})$. Let observe that transition $\lceil C \rceil \Rightarrow Q'$ can be specifically rewritten as:

$$\begin{aligned}
\lceil C \rceil = {} & P_{!\rho} \parallel Q_{!\pi} \parallel \lceil C'' \rceil \\
\equiv {} & \overbrace{\tilde{\pi}(\boldsymbol{t})}^{\pi_I(x_1).\cdots.\pi_k(x_k))} .\nu\boldsymbol{n}. \overbrace{P_\rho}^{\lceil r_{\rho_1} \rceil^{\#}_{(\boldsymbol{x};\boldsymbol{n})}} \parallel !\overline{\pi_I}(\underline{t}).0 \parallel \lceil C \rceil \text{ [by expanding } Q_{!\pi}, P_{!\rho}\text{]} \\
\Rightarrow {} & \underbrace{\pi_2(x_2).\cdots.\pi_k(x_k).\nu\boldsymbol{n}.P_\rho[\underline{\boldsymbol{t}_0}/x_1] \parallel \lceil C \rceil}_{Q'}
\end{aligned}$$

At this point, by observing that process $\pi_2(\boldsymbol{t}_2).\cdots.\pi_k(\boldsymbol{t}_k).\nu\boldsymbol{n}.P_\rho[\underline{t_0}/t_1]$. is indeed one that is considered garbage by the $\lfloor \_ \rfloor$ (i.e., it is mapped into the empty multiset) it is easy to check that $\lfloor Q' \rfloor = C$, and we conclude observing that $C \longrightarrow^* C$ is a possible transition.[2]

$\star$ Case $(II.a')$. Another sub-case happens when intruder is involved. Specifically when $\lceil C \rceil = Q_{!\pi} \parallel Q_{!I} \parallel \lceil C'' \rceil$ and transition $\lceil C \rceil \Rightarrow Q'$ may be instantiated as:

$$\begin{aligned}
\lceil C \rceil = {} & Q_{!\pi} \parallel Q_{!I} \parallel \lceil C'' \rceil \\
\equiv {} & \overline{\pi}(\underline{t}).0 \parallel \pi(x).\overline{I}(x).0 \parallel \lceil C'' \rceil \\
\Rightarrow {} & \underbrace{0 \parallel \overline{I}(\underline{t}).0 \parallel \lceil C'' \rceil}_{Q'}
\end{aligned}$$

Then we have:

$$C = \pi(\underline{t}), C'' \longrightarrow \underbrace{\pi(\underline{t}), I(\underline{t}), \lceil C'' \rceil}_{C'} \text{ [by } r_{I_1}\text{]}$$

and it is easy to check that $C' = \lfloor Q' \rfloor$.

$\star$ Case $(II.b)$: $(C, Q) = (\lfloor Q \rfloor, Q)$. The only interesting cases in this side, arise by considering those Q's such that $\lfloor Q \rfloor = C$, for some $C : C_0 \longrightarrow^* C$. In

[2] Note that the particular case where $\lceil C \rceil = \nu n.P_\rho$ i.e., is part of the case pa_ν.

fact, if C contains no role predicates, w.r.t. a role ρ, every Q containing only partial instantiations of that role (i.e., processes starting with a π or ν that are suffix of P_ρ) is such that $\lfloor Q \rfloor = C$. Treating this class of case as a one general case, the transition $Q \Rightarrow Q'$ can be written as:

$$\begin{aligned} Q = {} & \pi_j(x_j).\cdots.\pi_k(x_k).\nu\boldsymbol{n}.P_\rho \parallel \overline{\pi_j}(\underline{t}).0 \parallel \lfloor C'' \rfloor \qquad [j > 1] \\ \Rightarrow {} & \underbrace{\pi_{j+1}(x_{j+1}).\cdots.\pi_k(x_k).\nu\boldsymbol{n}.P_\rho[\underline{t}/x_j].0 \parallel \lfloor C'' \rfloor}_{Q'} \end{aligned}$$

Note that despite this transition, $\lfloor Q' \rfloor = C$ still hold. In fact partial instantiated (role) processes are mapped onto the empty multiset. Then we conclude observing that $C \longrightarrow^* C$ is a possible transition.

(a = I**)** Here we treat with transitions that involve channel I. When the intruder channel I is involved, many different situations involving the intruder arise. Here we will treat just some of the most significative ones i.e., those involving the states in Fig. 1. The others can be analyzed in a similar way.
⋆ Case $(II.a)$: $(C, Q) = (C, \lceil C \rceil)$. A sub-case of this class happens when $C = I(\underline{t_1}), I(\underline{t_2}), C''$. We start observing that transition $\lceil C \rceil \Rightarrow Q'$ can be written as:

$$\begin{aligned} \lceil C \rceil = {} & \bar{I}(\underline{t_1}).0 \parallel \bar{I}(\underline{t_2}).0 \parallel \lceil C'' \rceil \\ & \bar{I}(\underline{t_1}).0 \parallel \bar{I}(\underline{t_2}).0 \parallel \\ \equiv {} & I(x_1).\bar{I}(x_1).I(x_2).\bar{I}(x_2).\bar{I}(\langle x_1,\ x_2 \rangle).0 \text{ [expanding PA}_P\text{ state]} \\ & \parallel \lceil C'' \rceil \\ & 0 \parallel \bar{I}(\underline{t_2}).0 \parallel \\ \Rightarrow {} & \underbrace{\bar{I}(\underline{t_1}).I(x_2).\bar{I}(x_2).\bar{I}(\langle \underline{t_1},\ x_2 \rangle).0 \quad \parallel \lceil C'' \rceil}_{Q'} \qquad \text{[expanding PA}_P\text{ state]} \end{aligned}$$

Note that despite this transition, $\lfloor Q' \rfloor = C$ still holds. In fact partial instantiated (role) processes are mapped onto the empty multiset. Then we conclude observing that $C \longrightarrow^* C$ is a possible transition.
No more interesting cases fall in this class. On the contrary, many cases arise when considering situation in class (b) i.e., those Q such that $\lfloor Q \rfloor = C = I(\underline{t_1}), I(\underline{t_2}), C''$.
⋆ Cases $(II.b), (II.b'), (II.b'')$: $(C, Q) = (\lfloor Q \rfloor, Q')$. Let us consider the following processes (see also Fig. 1)

$$\begin{aligned} Q_1 &= \bar{I}(\underline{t_1}).0 \parallel \bar{I}(\underline{t_1}).I(x_2).\bar{I}(x_2).\bar{I}(\langle \underline{t_1},\ x_2 \rangle).0 \parallel \lceil C'' \rceil \\ Q_2 &= \bar{I}(\underline{t_1}).\bar{I}(\underline{t_2}).0 \parallel \lceil C'' \rceil \\ Q_3 &= [\langle \underline{t_1},\ \underline{t_2} \rangle = \langle x_1,\ x_2 \rangle]\, \bar{I}(x_1).I(x_2).0 \parallel \lceil C'' \rceil \end{aligned}$$

each translated into C via $\lfloor \_ \rfloor$ (specifically via $\lfloor \_ \rfloor_I$). Let us observe that for any $Q'_i : Q_i \Rightarrow Q'_i$ then $\lfloor Q'_i \rfloor = C$, for $i = 1, 2, 3$. Then we conclude observing that $C \longrightarrow^* C$ is a possible corresponding transition.

$\star$ Case $(I.b)'''$: A last interesting situation happens when:

$$Q = \bar{I}(\underline{t_2}).0 \parallel \bar{I}(\underline{t_1}).0 \parallel I(x_2).\bar{I}(x_2).\bar{I}(\langle \underline{t_1},\, x_2 \rangle).0 \parallel \lceil C'' \rceil$$

In this case we observe that:

$$\begin{aligned} Q = {} & \bar{I}(\underline{t_2}).0 \parallel \bar{I}(\underline{t_1}).0 \parallel I(x_2).\bar{I}(x_2).\bar{I}(\langle \underline{t_1},\, x_2 \rangle).0 \parallel \lceil C'' \rceil \\ \Rightarrow {} & \underbrace{0 \parallel \bar{I}(\underline{t_1}).0 \parallel \bar{I}(\underline{t_2}).\bar{I}(\langle \underline{t_1},\, \underline{t_2} \rangle).0 \parallel \lceil C'' \rceil}_{Q'} \end{aligned}$$

Then we have:

$$\lfloor Q \rfloor = I(\underline{t_2}), I(\underline{t_1}), C'' \longrightarrow \underbrace{I(\underline{t_2}), I(\underline{t_1}), I(\langle \underline{t_1},\, \underline{t_2} \rangle), C''}_{C'} \text{ [by } r_{I_6}\text{]}$$

and it is easy to check that $\lfloor Q' \rfloor = C'$.

- $\boxed{pa_\nu \textbf{ (i.e., new name generation)}}$

The only possible transition pa_ν happens when analyzing cases in (b) i.e., when $(C, Q) = (Q, \lfloor Q \rfloor)$. In fact no process obtained from $\lceil \_ \rceil$ can perform a pa_ν transition as first step.

$\star$ Case $(II.b)$: $(C, Q) = (\lfloor Q \rfloor, Q')$. The first easy scenario is the following:

$$\begin{aligned} Q = {} & \overbrace{\nu \boldsymbol{n}}^{\nu n_1.\cdots.\nu n_h} .P_\rho \parallel \lceil C \rceil \\ \Rightarrow {} & \underbrace{\nu n_2. \cdots \nu n_h.P_\rho[\underline{m}/n_1] \parallel \lceil C \rceil}_{Q'} \end{aligned}$$

In this case, being $\nu n_2. \cdots \nu n_h.P_\rho[\underline{m}/n_1]$ one of the processes left out by encoding $\lfloor \_ \rfloor$, we have that $\lfloor Q' \rfloor = \lfloor Q \rfloor = C$, and we conclude observing that $C \longrightarrow^* C$ is a possible transition.

$\star$ Case $(II.b')$: the second, more interesting, scenario happens when:

$$\begin{aligned} Q = {} & \nu n_h.P_\rho[\theta] \parallel \lceil C \rceil \quad \text{[where } \theta \text{ are the substitutions applyed so far]} \\ \Rightarrow {} & \underbrace{P_\rho \overbrace{[\theta][\underline{m}/n_1]}^{\theta'} \parallel \lceil C \rceil}_{Q'} \end{aligned}$$

Then we have

$$\lfloor Q\rfloor = \overbrace{\tilde{\pi}(\underline{\boldsymbol{t}}), C''}^{\lceil C\rceil} \longrightarrow \underbrace{A_{\rho_0}(\boldsymbol{x};\boldsymbol{n})[\theta'], \lceil C\rceil}_{C'} \text{ [by } r_{\rho_0}]$$

and it is easy to check that $\lceil C'\rceil = Q'$.

- $\boxed{pa_{[]} \textbf{ (i.e., matching)}}$

The only interesting case happens when $C = A_{\rho_{i-1}}(\underline{\boldsymbol{x}'[\theta]}), C''$ and $r_{\rho_i} = A_{\rho_{i-1}}(\boldsymbol{t}(\boldsymbol{x})) \longrightarrow A_{\rho_i}(\boldsymbol{x})$. Let start observing that in this case transition $\lceil C\rceil \Rightarrow Q'$ can be written as:

$$\begin{aligned}
\lceil C\rceil = {} & \overbrace{[\underline{\boldsymbol{x}'[\theta]} = \boldsymbol{t}(\boldsymbol{x})].\quad \overbrace{P_\rho[\theta]}^{\lceil r_{\rho_{i+1}}\rceil^{\#}_{(\boldsymbol{x})}[\theta]}}^{\lceil A_{\rho_{i-1}}(\underline{\boldsymbol{x}'[\theta]})\rceil} \quad \parallel \lfloor C''\rfloor \\
\Rightarrow {} & \qquad \overbrace{P_\rho[\theta][\theta']}^{\lceil r_{\rho_{i+1}}\rceil^{\#}_{(\boldsymbol{x})}[\theta][\theta'] = \lceil A_{\rho_i}(\underline{\boldsymbol{x}[\theta][\theta']})\rceil} \qquad \parallel \lfloor C''\rfloor \text{ [where } \theta' : \underline{\boldsymbol{x}'[\theta]} = \boldsymbol{t}(\boldsymbol{x})[\theta']]
\end{aligned}$$

Then we have:

$$C = A_{\rho_{i-1}}(\underline{\boldsymbol{x}'[\theta]}), C'' \longrightarrow \underbrace{A_{\rho_i}(\underline{\boldsymbol{x}[\theta][\theta']}), C''}_{C'}$$

and it is easy to check that $\lceil C'\rceil = Q'$.

- $\boxed{pa_{\equiv} \textbf{ (i.e., structural equivalence)}}$

The proof in case of $pa_{\equiv}$ transitions, follows easily from the previous transition cases by induction.

Here ends proof of **(II)**, where we have shown that for every $(C, Q) \in \mathcal{R}\ Q \Rightarrow Q'$ implies $C \longrightarrow^{*} C'$, and $(C', Q') \in \mathcal{R}$. □

Theorem 2 (Reminder). *Given an* PA$_P$ *security protocol theory* Q*. Then* $\lfloor Q\rfloor \sim Q$.

Proof. Similar to the proof of Theorem 1, by defining the relation $\mathcal{R}' = \{(\lfloor Q\rfloor, Q) : Q_0 \Rightarrow^{*} Q\}$ and showing that it is a correspondence relation $\sim$. □

References

[1] M. Abadi and B. Blanchet, Analyzing security protocols with secrecy types and logic programs, *ACM SIGPLAN Notices* **31**(1) (2002), 33–44. *Proc. of the 29th ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages (POPL'02)*.

[2] M. Abadi and A.D. Gordon, Reasoning about cryptographic protocols in the spi calculus, in: *Proc. of CONCUR'97: Concurrency Theory, 8th International Conference*, Volume 1243 of *Lecture Notes in Computer Science*, Springer, 1997, pp. 59–73.

[3] M. Abadi and A.D. Gordon, A bisimulation methods for cryptographic protocols, in: *Proc. of ESOP'98*, 1998.

[4] S. Bistarelli, I. Cervesato, G. Lenzini and F. Martinelli, Relating multiset rewriting and process algebras (for immediate decryption protocols, in: *Proc. of the Second International Workshop on Mathematical Methods, Models and Architectures for Computer Network Security (MMM'03), LNAI 2776*, St. Peterburg, Russia, 2003.

[5] S. Bistarelli, I. Cervesato, G. Lenzini and F. Martinelli, Relating process algebras and multiset rewriting for security protocol analysis, in: *Third Workshop on Issues in the Theory of Security – WITS'03*, R. Gorrieri, ed., Warsaw, Poland, 2003, pp. 21–31.

[6] M. Boreale, Symbolic trace analysis of cryptographic protocols in the spi-calculus, in: *Proc. of ICALP 2001*, 2001.

[7] M. Burrows, M. Abadi and R. Needham, A logic of authentication, in: *Proc. of the Royal Society of London*, Volume 426 of *Lecture Notes in Computer Science*, Springer, 1989, pp. 233–271.

[8] I. Cervesato, Typed multiset rewriting specification of security protocols, in: *Electronic Notes in Theoretical Computer Science*, Volume 40, Elsevier, 2001.

[9] I. Cervesato, N.A. Durgin, P.D. Lincoln, J.C. Mitchell and A. Scedrov, A meta-notation for protocol analysis, in: *Proc. of the 12th IEEE Computer Security Foundations Workshop (CSFW'99)*, IEEE Computer Society Press, 1999.

[10] I. Cervesato, N.A. Durgin, P.D. Lincoln, J.C. Mitchell and A. Scedrov, Relating strands and multiset rewriting for security protocol analysis, in: *Proc. of the 13th IEEE Computer Security Foundations Workshop (CSFW'00)*, IEEE, 2000, pp. 35–51.

[11] E.M. Clarke, S. Jha and W. Marrero, A machine checkable logic of knowledge for protocols, in: *Proc. of Workshop on Formal Methods and Security Protocols*, 1998.

[12] F. Crazzolara and G. Winskel, Events in security protocols, in: *Proceedings of the 8th ACM conference on Computer and Communications Security*, ACM Press, 2001, pp. 96–105.

[13] G. Denker and J.K. Millen, Capsl integrated protocol environment, in: *Proc. of DARPA Information Survivability Conference (DISCEX 2000)*, IEEE Computer Society, 2000, pp. 207–221.

[14] G. Denker, J.K. Millen, A. Grau and J.K. Filipe, Optimizing protocol rewrite rules of CIL specifications, in: *CSFW*, 2000, pp. 52–62.

[15] D. Dolev and A. Yao, On the security of public-key protocols, *IEEE Transaction on Information Theory* **29**(2) (1983), 198–208.

[16] B. Donovan, P. Norris and G. Lowe, Analyzing library of security protocols using Casper and FDR, in: *Proc. of the Workshop on Formal Methods and Security Protocols*, 1999.

[17] J. Thayer Fábrega, J. Herzog and J.D. Guttman, Honest ideals on strand spaces, in: *Proc. of the 11th IEEE Computer Security Foundations Workshop (CSFW'98)*, Washington, Brussels, Tokyo, IEEE, 1998, pp. 66–78.

[18] J. Thayer Fábrega, J. Herzog and J.D. Guttman, Strand spaces: Why is a security protocol correct? in: *Proc. of the 19th IEEE Computer Society Symposium on Research in Security and Privacy*, 1998.

[19] M. Fiore and M. Abadi, Computing symbolic models for verifying cryptographic protocols, in: *Proc. of the 14th Computer Security Foundation Workshop (CSFW-14)*, IEEE Computer Society, 2001, pp. 160–173.

[20] R. Focardi and R. Gorrieri, The compositional security checker: a tool for the verification of information flow security properties, *IEEE Transactions on Software Engineering* **23**(9) (1997), 550–571.

[21] R. Focardi, R. Gorrieri and F. Martinelli, NonInterference for the analysis of cryptographic protocols, in: *Proc. of the ICALP'00*, Springer, 2000.

[22] R. Focardi and F. Martinelli, A uniform approach for the definition of security properties, in: *Proc. of Congress on Formal Methods (FM'99)*, Volume 1708 of *Lecture Notes in Computer Science*, Springer, 1999, pp. 794–813.

[23] A.D. Gordon and A. Jeffrey, Authenticity by typing for security protocols, in: *Proc. 14th IEEE Computer Security Foundations Workshop (CSFW 2001)*, IEEE Computer Society, 2001, pp. 145–159.

[24] J.D. Guttman and F.J. Thayer Fábrega, Authentication tests and the structure of bundles, *Theoretical Computer Science*, 2001.

[25] G. Lowe, Breaking and fixing the Needham–Schroeder public-key protocol using FDR, in: *Proc. of 19th IEEE Computer Security Foundations Workshop (CSFW'96)*, Volume 1055 of *Lecture Notes in Computer Science*, Springer, 1996, pp. 147–166.

[26] G. Lowe, Some new attacks upon security protocols, in: *Proc. of 19th IEEE Computer Security Foundations Workshop (CSFW'96)*, Volume 1055 of *Lecture Notes in Computer Science*, Springer, 1996, pp. 147–166.

[27] F. Martinelli, Encoding several security properties as properties of the intruder's knowledge, Dec. 20, Institute of Informatics and telematics – CNR, 2001.

[28] C.A. Meadows, The NRL protocol analyzer: an overview, in: *Proc. of the 2nd International Conference on the Practical Application of PROLOG*, 1994.

[29] D. Miller, Higher-order quantification and proof search, in: *Proceedings of the AMAST Conference*, LNCS, Springer, 2002.

[30] R. Milner, *Communication and Concurrency*, International Series in Computer Science. Prentice Hall, 1989.

[31] R. Milner, *Communicating and Mobile Systems: The π-Calculus*, Cambridge University Press, 2000.

[32] R. Milner, J. Parrow and D. Walker, A calculus of mobile processes, I and II, *Information and Computation* **100**(1) (1992), 1–40.

[33] R.M. Needham and M.D. Schroeder, Using encryption for authentication in large network of computer, *Communication of the ACM* **21**(12) (1978), 993–999.

[34] L.C. Paulson, Proving properties of security protocols by induction, in: *Proc. of The 10th Computer Security Foundations Workshop*, IEEE Computer Society, 1997.

[35] S. Schneider, Security properties and CSP, in: *Proc. of the IEEE Symposium on Research in Security and Privacy*, 1996, pp. 174–187.

[36] S. Schneider, Verifying authentication protocols in CSP, *IEEE Transaction on Sofware Engineering* **24**(8) (1998), 743–758.

[37] D. Song, Athena: a new efficient automatic checker for security protocol analysis, in: *Proceedings of the Twelfth IEEE Computer Security Foundations Workshop*, Mordano, Italy, 1999, IEEE Computer Society, pp. 192–202.

[38] T. Woo and S. Lam, A semantic model for authentication protocols, in: *Proc. of the IEEE Symposium on Research in Security and Privacy*, 1993.