Authentication via Localized Names *

Chiara Bodei, Pierpaolo Degano
Dipartimento di Informatica, Università di Pisa
Corso Italia, 40, I-56125 Pisa, Italy
{chiara,degano}@di.unipi.it

Riccardo Focardi
Dipartimento di Informatica, Università Ca' Foscari di Venezia
Via Torino 155, I-30173 Venezia, Italy
focardi@dsi.unive.it

Corrado Priami
Dipartimento Scientifico Tecnologico, Università di Verona
Ca' Vignal 2, Strada Le Grazie, 1, I-37134 Verona, Italy
priami@sci.univr.it

Abstract

We address the problem of message authentication using the π-calculus, which has been given an operational semantics in [2] that provides each sequential process of a system with its own local space of names. We exploit here that semantics and its localized names to guarantee by construction that a message has been generated by a given entity. Therefore, our proposal can be seen as a reference for the analysis of "real" protocols. As an example, we study the way authentication is ensured by encrypting messages in the spi-calculus [1].

1. Introduction

Authentication is one of the main issues in security. It can have different purposes depending on the specific application considered. For example, *entity authentication* is related to the verification of an entity's claimed identity [8], while *message authentication* should make it possible for the receiver of a message to ascertain its origin [20]. In recent years there have been some formalizations of these different aspects of authentication (see, e.g., [3, 1, 11, 19, 5]). These formalizations are crucial for proofs of authentication properties, that sometimes have been automatized (see, e.g., [7, 14, 10, 9, 4]). A typical approach presented in the literature is the following. First, a protocol is specified in a certain formal model. Then the protocol is shown to enjoy the desired properties, regardless of its operating environment, that can be unreliable, and can even harbour an hostile intruder.

We use here a basic calculus for modelling concurrent and mobile agents, whose semantics offers a built-in mechanism that guarantees message authentication. This is the main contribution of our paper, because it enables us to abstract from the various implementations/specifications of authentication mechanisms, and to obtain idealized protocols which are "secure by construction". Our protocols, or rather their specifications can then be seen as a reference for proving the correctness of "real protocols". The actual proofs can be carried out following the approach and using the tools mentioned above. Additionally, our built-in authentication mechanism suggests a further proof technique, that we shall briefly discuss in the conclusions.

The essence of concurrent and mobile computation can be studied in a pure form using the π-calculus [12], a foundational calculus based on the notion of naming. Systems are specified as expressions of the calculus, called *processes*. These are obtained by combining via a few operators (parallel composition, nondeterministic choice, declarations) the basic actions of sending and of receiving names between processes along channels. Names represent values, or messages, and also channels. Since processes exchange names in communications, the interconnection structure of

*This work has been partially supported by by the MURST Progetto *Tecniche Formali per la Specifica, l'Analisi, la Verifica, la Sintesi e la Trasformazione di Sistemi Software*. The first two authors have been also partially supported by the CNR Progetto Strategico *Modelli e Metodi per la Matematica e l'Ingegneria*.

a network can vary dynamically. Recently, Abadi and Gordon [1] defined the *spi*-calculus by enriching the π-calculus with a primitive of encryption. The resulting calculus is particularly suited to security issues, among which authentication.

In [2] the π-calculus has been equipped with a structural operational semantics which endows each sequential process P in the whole system with its own *local* environment, i.e., P has its *local* space of names and its *local* name manager that generates a fresh name, whenever necessary. In this proposal, when a name is communicated, say from P to Q, it arrives at destination equipped with the so-called *relative address*: it represents the path between P and Q in (an abstract view of) the network (think to it as an abstract version of a logic IP address). The relative address encodes the information needed to point back to the local environment of P where the name has been installed as fresh, in the style of static chains that are used to solve non local references in sequential languages with nested environments [17]. The names enriched with relative addresses are called *localized*. The reference to the generator of a name is maintained during all the communications by an operator of *address composition*, which updates the relative addresses of the names transmitted. Names generated by different environments are thus certainly kept distinct. Note that address composition is *not* available to the users of the π-calculus, i.e. it *cannot* occur in a process. Rather it is used by the abstract machine of the π-calculus only, defined by its semantics.

We propose here to use this semantics, and its straightforward extension to the *spi*-calculus, to study authentication. As a matter of fact, the semantics itself provides us with a built-in primitive that enables the receiver of a message to ascertain its origin. The key point is that the relative address of a message uniquely identifies its originator, even after the message has been maliciously intercepted and forwarded.

We write the primitive for authentication as $[x \overset{@}{=} A]P$, where x is a name and A is a process. Intuitively, the execution of the process P starts only when the check $[x \overset{@}{=} A]$ succeeds, and this happens if and only if x has been originated by A. [1] In other words, the interpreter of the calculus (i.e. the semantic rules, *not* the user!) checks whether the local environment to which x belongs is the one of A.

A communication protocol that guarantees authentication by construction is now easy to define, by using the plain handshaking communication of the π-calculus and the primitive sketched above. Suppose that a process A sends a (fresh) message M to B along a public channel c. In the π-calculus syntax they have the following form:

$$A = (\nu M)\overline{c}M.A'$$

and

$$B = c(x).[x \overset{@}{=} A]B'(x),$$

where the operator (νM) declares M to be a fresh name, different from all the others in the whole system; $\overline{c}M$ stands for sending M along c; $c(x)$ is an input action that paired in a communication with an output $\overline{c}z$ has the effect of binding x to z within its residual yielding $[z \overset{@}{=} A]B'(z)$. If we put A, B and also an intruder E in parallel, $A \mid B \mid E$, the effect is to guarantee that the residual of B is indeed $B'(M)$. Note that the intruder might as well intercept the message M originated by A, but it cannot modify any of its parts, because relative addresses are manipulated by the semantics only. Also, there is no need for c to be a channel private to A and B.

The definition of the semantics of [2] is very detailed, and quite long. We omit large parts of it, referring the interested reader to its original presentation. We only describe it intuitively in Sect. 3, and recall in Sect. 4 the basic definitions of localized names and of their management. The SOS rules for the π-calculus and a few auxiliary notions are in the Appendix. The example above is worked out in detail in Sect. 3 and any intrusion shown to fail.

Our solution assumes that the implementation of the communication primitive has a reliable mechanism to control and manage relative addresses. In some real cases this is possible, e.g., if the network management system filters every access of a user to the network. This may be not the case in many other situations. Yet, our solution may help checking the correctness of different implementations, e.g. those based on cryptography. A remarkable example of a process algebra with cryptographic features is given by the *spi*-calculus [1], over which the mechanism of handling names locally carries easily. We can then verify if a cryptography-based protocol ensures authentication, by checking a version of it containing also the primitive $[x \overset{@}{=} A]$: the original formulation and the extended one should exhibit the same behaviour. This check of specifications against implementations, is much in the style of the congruence-based technique proposed in [1]. We shall not present it in detail. We only show in Sect. 5 the needed extensions to model in our framework the primitives for encryption and decryption and we show on an example how an attack is revealed.

2. The π-calculus

In this section we briefly recall the π-calculus [12], a model of concurrent communicating processes based on the notion of *naming*. We give only its intuitive semantics; the formal semantics of the calculus is in Appendix A.

[1] For the sake of readability we write here the operands in a simplified form; in the full notation, the name x is equipped with a relative address, that is checked for equality against the address of A (for details, see below).

Definition 2.1 (syntax) *Let $\mathcal{N}$ be a countable, infinite set of* names *ranged over by $a, b, \ldots, x, y, \ldots$, and let τ be a distinguished element such that $\mathcal{N} \cap \{\tau\} = \emptyset$.* Processes *(denoted by $P, Q, R, \ldots \in \mathcal{P}$), are built from names according to the syntax*

$$\begin{array}{rcl} P & ::= & \mathbf{0} \mid \pi.P \mid P + P \mid P|P \mid (\nu x)P \mid \\ & & [x = y]P \mid A(y_1, \ldots, y_k) \end{array}$$

where π may either be $x(y)$ for input*, or $\overline{x}y$ for* output *or τ for* silent *moves and A is an agent identifier. Hereafter, the trailing $\mathbf{0}$ will be omitted (i.e. π will stand for $\pi.\mathbf{0}$).*

Intuitively, $\mathbf{0}$ represents the null process which can do nothing. The prefix π is the first atomic action that the process $\pi.P$ can perform. After the execution of π the process $\pi.P$ behaves like P. The input prefix $x(y)$ binds the name y in the prefixed process as follows: when a name z is received along the link named x, all the (free) occurrences of y in the prefixed process P are substituted with z. This models an input on link x. [2] The output prefix $\overline{x}z$ does not bind the name z which is sent along x. The silent prefix τ denotes an action which is invisible to an external observer of the system. Summation denotes nondeterministic choice. The process $P_1 + P_2$ behaves like either P_1 or P_2. The operator $|$ describes parallel composition of processes. The components of $P_1|P_2$ may act independently; also, an output action of P_1 (resp. P_2) at any output port $\overline{x}$ may synchronize with an input action of P_2 (resp. P_1) at x to create a silent action denoting the communication. The value sent by P_1 replaces the relevant occurrences of the placeholder y in P_2 (see the comment above on input). The operator (νx) acts as a static declaration (i.e. a binder for) the name x in the process P that it prefixes. In other words, x is a unique name in P which is different from all the external names. The agent $(\nu x)P$ behaves as P except that actions at ports $\overline{x}$ and x are prohibited. However communications along link x of components *within* P are allowed since the result is a silent action. When a private name x is sent as (bound) output on a public channel y the effect is to vanish the (νx) operator, as in $Q = (\nu x)\overline{y}x.P \xrightarrow{\overline{y}(x)} P$. The intuition behind this operation is to make the name x, which is private to Q, available to the environment of P. Since the operator (νx) can be interpreted as a declaration, the bound output enlarges its scope and guarantees that the name x is fresh. (For this reason a bound output is sometimes referred to as *scope extrusion* in the literature.) For example, the generation of a session key can be modelled by an action of this kind. Matching $[x = y]P$ is an `if-then` operator: process P is activated only if $x = y$. Finally, each agent identifier A has a unique defining equation of the form $A(y_1, \ldots, y_k) = P$. The equation $A(y_1, \ldots, y_k) = P$ is akin to the definition of function: it is invocated by $A(z_1, \ldots, z_k)$ and triggers the substitution of the actual parameter z_i for the corresponding formal parameter y_i.

The operational semantics of the calculus is defined in a logical style by inducing on the syntax of the processes. This approach is usually called structural operational semantics (SOS, for short). There are also static rules that make equivalent syntactical different terms, like commutativity of choices or avoidance of declaration of unused names. These rules define a structural congruence $\equiv_P$ that simplifies the semantic rules. E.g. the processes $[x = x]P$ and P behave exactly in the same way, and this can be expressed through the static structural rule $[x = x]P \equiv_P P$. For more details see the Appendix A.

3. The Authentication Primitive

We recall here the basic ideas of [2] which extend the standard semantics of the π-calculus of [12], in that it handles names locally. The resulting operational semantics provides us with an authentication primitive that can be used to specify protocols which guarantee authentication "by construction".

Consider for a while the binary parallel composition as the main operator of the π-calculus, stipulating that it is neither associative nor commutative. Then, build abstract syntax trees of processes as binary trees, called *trees of (sequential) processes* as follows. Given a process P, the nodes of its tree correspond to the occurrences of the parallel operator in P, and its leaves are the sequential components of P. For an example, see Fig. 1.

Assume now that the left (resp. right) branches of a tree of sequential processes denote the left (resp. right) component of parallel compositions, and label their arcs with tag $||_0$ (resp. $||_1$). Therefore, any sequential component of a process is uniquely identified by a string ϑ over $\{||_0, ||_1\}^*$. The string corresponds to a path from the root, the top-level $|$ of the whole process, to a leaf. Intuitively, ϑ is the address of the sequential component relative to the root of the binary tree. The space of names of a whole process is then partitioned into local environments associated each with its sequential sub-process. Consider two different sequential processes, say G and R, that have generated two syntactically equal names, say n. Suppose now that G sends its n to R. It is necessary to distinguish between the two different instances of n. In order to do this, in the local environment of R, the name generated by G will be enriched with the *address* of G *relative* to R, which points back from R to the local environment of G. The relative address can be decomposed into two parts according to the minimal common predecessor P of G and R in the tree. A relative address

[2] This is the so-called *late* semantics, because the binding of y takes place at communication time, opposed to the *early* semantics [13] that we do not consider here.

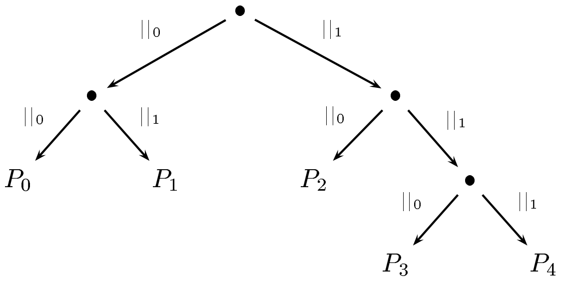

Figure 1. The tree of (sequential) processes of $(P_0|P_1)|(P_2|(P_3|P_4))$.

is then a string $\vartheta\bullet\vartheta' \in \{\|_0, \|_1\}^*\bullet\{\|_0, \|_1\}^*$, where ϑ represents the path from P to R, and ϑ' the path from P to G. Let G and R respectively be the processes P_3 and P_1 of Fig. 1. The address of P_3 relative to P_1 is $\|_0\|_1\bullet\|_1\|_1\|_0$ (read the path upwards from P_1 to the root and reverse, then downwards to P_3). Relative addresses are inductively built while deducing transitions.

A slightly more complex situation arises when a process receives a name and sends it to another process. The name must arrive at the new receiver with the address of the generator (*not* of the sender) relative to the new receiver (think to an intruder that simply forwards an intercepted message). As an example assume that, in Fig. 1, P_1 sends to P_2 a name generated by P_3 (i.e. with relative address $\|_0\|_1\bullet\|_1\|_1\|_0$). The rules for communication determine the address of P_1 relative to P_2, i.e. $\|_1\|_0\bullet\|_0\|_1$, and compose it with the address of the communicated name in order to derive the new relative address. This is done through the address composition operation $\star$ which is defined in the next section: $\|_1\|_0\bullet\|_0\|_1 \star \|_0\|_1\bullet\|_1\|_1\|_0 = \|_1\|_0\bullet\|_0$.

We can now intuitively present our authentication primitive $[x \stackrel{@}{=} A]$. It is akin to the matching operator of the π-calculus. The process $[x \stackrel{@}{=} A]P$ enables P only if the relative addresses of the two localized names x and A are exactly the same, i.e. only if x has been generated by A. [3] It is now simple to ascertain the origin of a message.

Consider again the example presented in the Introduction, on which we show in detail how our authentication primitive works. The protocol P is:

$$\begin{array}{rcl} P & = & A \mid (B \mid E) \\ A & = & (\nu M)\overline{c}M.A' \\ B & = & c(x).[x \stackrel{@}{=} A]B'(x) \end{array}$$

Now we show the role that localized names play, and how we guarantee by construction that B' is executed only if the message bound to x has been originated by A, i.e., if the message received on channel c is indeed M. Assume, for simplicity, that the name of A used by B in the authentication check is indeed the relative address $\|_1\|_0\bullet\|_0$ pointing from B to A. As said above, this relative address encodes the "site" hosting A, thus it gives the identity of the process from which B is expecting to receive a message on channel c. In order to analyze the behaviour of the protocol in an hostile environment, we consider a generic enemy E, as powerful as possible. We now consider the following two possible message exchanges:

$$\begin{array}{lcl} \text{Message } 1 & : & A \rightarrow B : M \\ \text{Message } 2 & : & E \rightarrow B : N \end{array}$$

The first message represents the correct exchange of message M from A to B. The second one is an attempt of E to send a different message N to B. The enemy E could actually be of the following form:

$$E = (\nu N)\overline{c}N.E' + \langle \text{other evil actions} \rangle.$$

The names M and N are received by B prefixed by the relative address corresponding to the respective originators.

Fig. 2 shows the two message exchanges. In particular we see that M is received by B as $\|_1\|_0\bullet\|_0 M$ while N becomes $\|_0\bullet\|_1 N$. It is now immediate to see that only in the first case B will evolve to $B(M)$ while in the latter it will stop. This is so because only $\|_1\|_0\bullet\|_0 M$ matches with the address of A. Every attempt of the enemy of introducing new messages on c is filtered out by the authentication primitive.

A further interesting case arises when the enemy intercepts M and forwards it to B. We will show in the following section that our mechanism accepts the message, as it is authentic.

4. Address Composition

Here we recall the definition of relative addresses and show how these are updated when names are communi-

[3] We assume that a process knows which are the other processes around, and which are the relative addresses of their local environments. Note also that x is a name while A is a process: we do not distinguish them here for the sake of presentation.

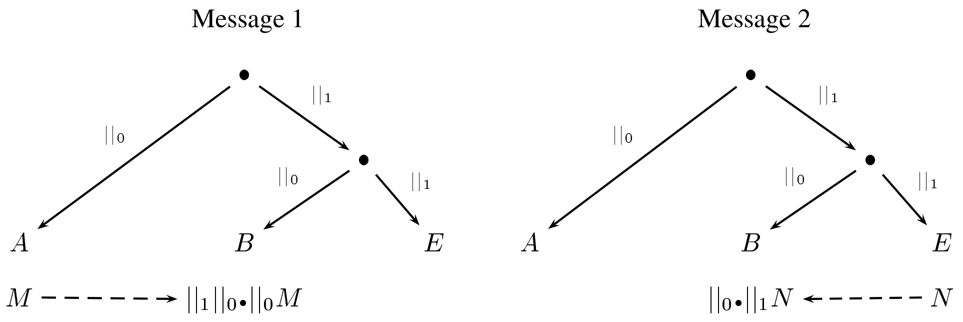

Figure 2. The process B detects that the message $||_1||_0\bullet||_0 M$ is authentic while the message $||_0\bullet||_1 N$ comes from the enemy E.

cated. More intuition, the full definitions and the statements of their correctness are in [2].

Definition 4.1 (relative addresses)
Let $\vartheta, \vartheta'_i, \vartheta_i \in \{||_0, ||_1\}^*$, let ϵ be the empty string, and let $\oplus$ be sum modulo 2. Then, the set of relative addresses *is*

$$\mathcal{A} = \{\vartheta_0 \bullet \vartheta_1 : \vartheta_i = ||_j \vartheta'_i \Rightarrow \vartheta_{i\oplus 1} = ||_{j\oplus 1}\vartheta'_{i\oplus 1}, i, j = 0, 1\}.$$

As we said in the previous section, we use relative addresses to encode paths between pairs of nodes of binary trees of processes, like the one in Fig. 1. Note that the condition $||_0\vartheta'_0\bullet||_1\vartheta'_1$ (and $||_1\vartheta'_0\bullet||_0\vartheta'_1$) makes it explicit that the two components of the relative address describe the two distinct paths going out from the *same* node in a binary tree. This node is the minimal common predecessor of the generator of a name and its user. We now define address composition. It is a partial operation, defined only when relative addresses can indeed be composed. We make sure that this is always the case when we apply it. Fig. 3 depicts the cases in which we will use the address composition $\star$, that discards the dashed paths.

Definition 4.2 (address composition) *Address composition $\star : (\mathcal{A} \times \mathcal{A}) \rightharpoonup \mathcal{A}$ is defined by the following three exhaustive cases:*

1. $\vartheta_0\bullet\vartheta \star \vartheta_2\vartheta\bullet\vartheta_3 = \vartheta_2\vartheta_0\bullet\vartheta_3$ *with* $\vartheta_2 \neq \epsilon$
2. $\vartheta_0\bullet\vartheta_1\vartheta \star \vartheta\bullet\vartheta_3 = \vartheta_0\bullet\vartheta_1\vartheta_3$ *with* $\vartheta_1 \neq \epsilon$
3. $\vartheta'\vartheta_0\bullet\vartheta \star \vartheta\bullet\vartheta'\vartheta_3 = \vartheta_0\bullet\vartheta_3$

Let G be the generator of a name, and S be the process which receives the name and sends it to the process R. Moreover, let $\vartheta_S\bullet\vartheta_G$ and $\vartheta_R\bullet\vartheta'_S$ be the addresses of G relative to S, and of S relative to R, respectively. Then, our address composition is correct because $\vartheta_R\bullet\vartheta'_S \star \vartheta_S\bullet\vartheta_G$ is the address of G relative to R (see Theorem 4.7 of [2]).

We extend below the syntax of the π-calculus with localized names. In the following definition we assume to have all the standard names of the π-calculus, including τ, and to represent a localized name as a natural number n, prefixed by a relative address, i.e. these names will have the form $\vartheta_0\bullet\vartheta_1 n$. Actually, there is an additional extension that provides each sequential process with its own name generator. For the sake of readability, we ignore name generators here, and we refer the reader to the Appendix B for their definition. We only mention that technically a name generator keeps track of the last local name n originated so far, and produces on demand $n+1$ as the new local name. (This may explain our use of natural numbers as local names, which however is not a limitation at all.) As a matter of fact, a new name is needed whenever the scope of a declaration (νx) is enlarged. This happens when a process P extrudes its local name x and makes it available to another process, say Q. In this case, the name x is replaced in (the residual of) P by the newly generated name $n + 1$, localized to the environment of P by prefixing it with the empty relative address $\epsilon\bullet\epsilon$. When communicated to Q, this name receives the correct relative address $\vartheta_0\bullet\vartheta_1$, as sketched above.

Definition 4.3 *Let $\mathcal{N}' = \mathcal{A} \cdot \mathbf{Z} \cup \mathcal{N} \cup \{\tau\}$ be the set of* localized names*, ranged over by $r, s, u, \ldots$, possibly indexed, where $\mathbf{Z}$ is the set of non-zero integer numbers, and "$\cdot$" is the operator of language concatenation.*
The new syntax of the π-calculus is obtained from the one in Def. 2.1, with r, s in place of x, y. In particular, π can either be $r(s)$ or $\overline{r}s$ or τ and a restricted name can be s.

Note that a free name, i.e., a name known to all the processes, is an element of $\mathcal{N}$ (the second set forming $\mathcal{N}'$) and carries no relative address.

We recall below how the operation of address composition is exploited to update the addresses of localized names when they are sent from one site to another (note that free

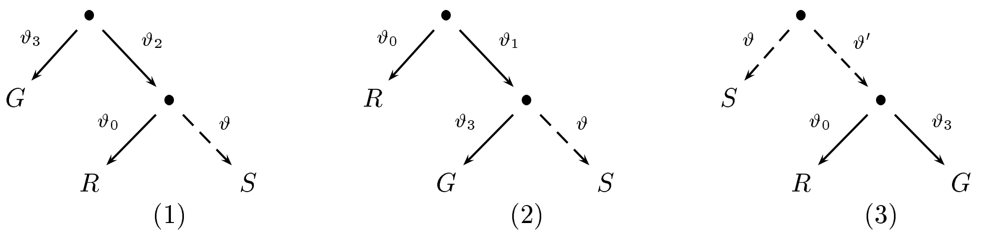

Figure 3. The three possible placements of the generator (G), the sender (S), and the receiver (R) of a name.

names, like a below, are not affected by the $\star$ operator).

Definition 4.4 *Let $r = \vartheta_r \bullet \vartheta'_r n$ (resp. a) be a name. A name r exported to the relative address $\vartheta \bullet \vartheta'$ is* $\vartheta \bullet \vartheta' \star r = (\vartheta \bullet \vartheta' \star \vartheta_r \bullet \vartheta'_r)n$ *(resp. $\vartheta \bullet \vartheta' \star a = a$).*

An example of address composition may help. We reconsider the protocol of the previous section and we analyze the case of a different enemy that, masquerading as B (written $E(B)$), intercepts M and forwards it to B:

$$\begin{array}{lcl} \text{Message 3} & : & A \rightarrow E(B) : M \\ \text{Message 4} & : & E \rightarrow B : M \end{array}$$

Since E does not modify M we would like to accept the message in B even if it has been forwarded by E. Address composition maintains the identity of messages, so we accept indeed this behaviour. In detail, $E(B)$ receives M as $||_1||_1 \bullet ||_0 M$. When E forwards it to B, the message is composed with the address of E relative to B, $||_0 \bullet ||_1$, yielding $||_0 \bullet ||_1 \star ||_1||_1 \bullet ||_0 M$. By applying the rule (1) of Def. 4.2 we see that the message is received by B as $||_1||_0 \bullet ||_0 M$ and that it is accepted as it is authentic.

5. Implementing Authentication

As we have shown above, our primitive allows processes to obtain authentication in a direct and "secure" way. It thus permits to completely abstract from the underlying mechanisms which are necessary to guarantee authentication. However, if we want to implement it, the interpreter itself of the calculus (or of a language based on it) must control and manage relative addresses in a highly reliable manner. E.g., this is the case for intranets.

Unfortunately, this may be not the case in many other situations. Typically, in a wide-area network setting, the network is completely insecure, i.e., users may have a complete access to the net management system and there is no reliable way of filtering these accesses. In our terminology, a malicious user could change the relative addresses of localized names making them unreliable, and our authentication primitive completely useless.

In these situations the "standard" approach to guarantee authentication (and security in general) is to use cryptography. The spi-calculus [1] is an extension of the π-calculus with cryptographic primitives. They make the formal definitions of protocols compact and elegant, and facilitate the proof that protocols guarantee authentication.

We use below our proposal to analyze the cryptographic "implementation" of authentication mechanisms specified in the spi-calculus. To do this, we first carry on the spi-calculus the semantics of [2] used so far. This step is quite easy, because the spi-calculus has the π-calculus as "core" language. We then observe if a particular authentication protocol is indeed able to "implement" our authentication primitive, i.e., if the cryptographic protocol is as strong in detecting names with an "incorrect" relative address as our secure-by-construction authentication protocol is.

Lack of space prevents us from giving all the definitions in detail; we only present them intuitively. We first extend our calculus with the shared-key cryptographic primitives of the spi-calculus: [4]

- $\{M\}_K$ represents the encryption of message M with key K;
- *case x of $\{y\}_K$ in P* represents the decryption of message x through key K; the result of the decryption is put in y. It can be read as follows: "if x is of the form $\{M\}_K$ then $P[M/y]$ is executed (i.e., P where M replaces all the free occurrences of y); otherwise it is stuck".

The management of localized names extends easily to the encryption and decryption primitives. The first raises no problems, because $\{M\}_K$ is a *new* name, local to the process that encrypts M, say P. Technically, one has only to

[4] In the following items, both M and K can in fact be localized names; for better supporting intuition, we prefer to write M and K instead of following pedantically our conventions that impose to use names r and u.

make sure that this new name never clashes with those produced by the name generator. The two localized names M and K are frozen just like they were at encryption time, and their own relative addresses are not changed when the encrypted message is sent through the net; actually, these relative addresses *cannot* be updated. Since $\{M\}_K$ is a new name, its relative address, say $\vartheta_0 \bullet \vartheta_1$, will always point to the process P that made the encryption. In this way, when decrypting $\{M\}_K$ the semantic rules recover the correct addresses for M and K by simply composing the actual address of $\{M\}_K$, $\vartheta_0 \bullet \vartheta_0$, with the (frozen) relative addresses of M and K, respectively.

We do not give an operational rule for encryption since it constructs a local name in quite a standard way. Instead, a static definition of structural congruence suffices, that identifies two syntactically different, but semantically equivalent processes, just as it is the case with the rule for structural congruence $[x = x]P \equiv_P P$. For decryption we stipulate the following (again M should be a localized name r; see the preceding footnote):

$$\textit{case } \vartheta_0 \bullet \vartheta_1 \{M\}_u \textit{ of } \{y\}_{\vartheta_0 \bullet \vartheta_1 \star u} \textit{ in } P \equiv_P P[\vartheta_0 \bullet \vartheta_1 \star M/y].$$

The intuition is that we can decrypt a message $\{M\}_u$ only if we use the key $s = \vartheta_0 \bullet \vartheta_1 \star u$ which is exactly how the frozen key u should appear to the receiver of the encrypted message. When the two names s and u, although starting from different sites, do refer to the same key, the semantic rules decrypt the message and update its relative address by composing $\vartheta_0 \bullet \vartheta_1$ with M. The congruence rule equates then the process at the left, where the match succeeds, and P, where $\vartheta_0 \bullet \vartheta_1 \star M$ replaces y.

We now show an example of a correct run of the Wide Mouthed Frog key exchange protocol. Consider its simplified version analyzed in [1]. The two processes A and B share keys K_{AS} and K_{BS} with a trusted server S. In order to establish a secure channel with B, A sends a fresh key K_{AB} encrypted with K_{AS} to the server S. Then, the server decrypts the key and forwards it to B, this time encrypted with K_{BS}. Now B has the key K_{AB} and A can send a message M encrypted with K_{AB} to B. The protocol should guarantee that when B receives M, such a message has been indeed originated by A.

The protocol is composed of the following three messages:

Message 1	$A \to S$	:	$\{K_{AB}\}_{K_{AS}}$
Message 2	$S \to B$	:	$\{K_{AB}\}_{K_{BS}}$
Message 3	$A \to B$	:	$\{M\}_{K_{AB}}$

Its specification in our calculus with localized names is:

$$\begin{array}{rcl} P & = & (\nu K_{AS})(\nu K_{BS})(\ (A|B) \mid S) \\ A & = & (\nu K_{AB})(\nu M)(\overline{c_{AS}}\{K_{AB}\}_{K_{AS}}.\overline{c_{AB}}\{M\}_{K_{AB}}) \\ S & = & c_{AS}(x).\textit{case } x \textit{ of } \{y\}_{K_{AS}} \textit{ in } \overline{c_{BS}}\{y\}_{K_{BS}} \\ B & = & c_{BS}(x).\textit{case } x \textit{ of } \{y\}_{K_{BS}} \textit{ in} \\ & & c_{AB}(z).\textit{case } z \textit{ of } \{r\}_y \textit{ in } B'(r) \end{array}$$

Not surprisingly, the specification is in the style of [1]. Indeed the localized name mechanism is not visible at this level, and we are not using yet our authentication primitive. Fig. 4 shows how the localized names are handled in a correct execution of the protocol P. Note that K_{AS} and K_{BS} are standard names, belonging to $\mathcal{N}$. Indeed, they are restricted in P but never extruded and therefore they never assume the form of localized names. So the keys have no relative address; of course, no problem would arise if they were localized names and carry a relative address, instead.

After the reception of message 2, B decrypts the received message $||_0||_1 \bullet ||_1\{||_1 \bullet ||_0||_0 K_{AB}\}_{K_{BS}}$. This means computing

$$\begin{array}{r} \textit{case } ||_0||_1 \bullet ||_1\{||_1 \bullet ||_0||_0 K_{AB}\}_{K_{BS}} \textit{ of } \{y\}_{K_{BS}} \textit{ in } c_{AB}(z). \\ \textit{case } z \textit{ of } \{r\}_y \textit{ in } B'(r), \end{array}$$

where $||_0||_1 \bullet ||_1 K_{BS} = K_{BS}$ because the key is a standard name in $\mathcal{N}$ (see Def. 4.4). The variable y is then set to

$$||_0||_1 \bullet ||_1 \star ||_1 \bullet ||_0||_0 K_{AB},$$

that results in $||_1 \bullet ||_0 K_{AB}$, by the rule (3) of Def. 4.2. This is indeed the correct reference to the key K_{AB} generated by A and installed in its local environment. In the last message B receives $||_1 \bullet ||_0\{M\}_{K_{AB}}$, and succeeds in decrypting it with $||_1 \bullet ||_0 K_{AB}$, obtaining $||_1 \bullet ||_0 M$. The addresses of M and of A relative to B are equal, so M is indeed authentic. This characterizes a "correct" execution.

It is well-known that an attack can take place over two sessions of the protocol above. Basically, it occurs when the enemy replays some messages of the first session in the second one. We follow below the formalization of [1], where this attack is analyzed. Note that in the single session illustrated above no problem arises instead, even if an enemy intercepts the message sent by A and then forwards it to B. Indeed the message received is the right one, because no one can alterate it.

Now, we show that the attack above is immediately detected by an observer that can compare localized name, e.g. by using our authentication primitive $[x \overset{@}{=} A]$. For the sake of readability, we call A' and B' the two instances of A and B in the second session where A' is trying to send the

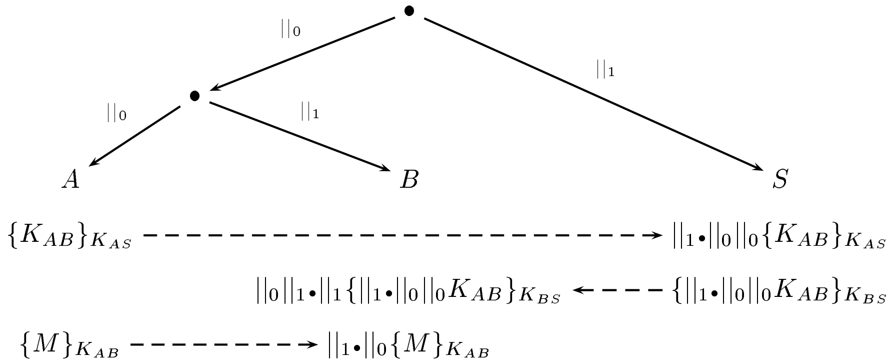

Figure 4. A correct execution of the Wide Mouthed Frog protocol

message M' to B' using the session key K'_{AB}:

$$\begin{array}{lll}
\text{Message } 1 & A \to S : & \{K_{AB}\}_{K_{AS}} \\
\text{Message } 2 & S \to B : & \{K_{AB}\}_{K_{BS}} \\
\multicolumn{3}{c}{\langle E \text{ eavesdrops Message } 2\rangle} \\
\text{Message } 3 & A \to B : & \{M\}_{K_{AB}} \\
\multicolumn{3}{c}{\langle E \text{ eavesdrops Message } 3\rangle} \\
\text{Message } 1' & A' \to S : & \{K'_{AB}\}_{K_{AS}} \\
\text{Message } 2' & E(S) \to B' : & \{K_{AB}\}_{K_{BS}} \\
\text{Message } 3' & E(A') \to B' : & \{M\}_{K_{AB}}
\end{array}$$

The enemy eavesdrops the first session and then replays messages 2 and 3 in the second session (messages $2'$ and $3'$). The result is that B' receives a copy of M instead of one of M'.

The specification of the protocol is:

$$P' \;=\; (\nu K_{AS})(\nu K_{BS})(\,(A \mid A) \mid (B \mid B)\,) \mid (S \mid E)$$

Note that A generates both M and K_{AB} as fresh names. So when we consider $A \mid A$, the two A's originate two different messages, say M and M', and two different keys, say K_{AB} and K'_{AB}. We need to modify S so that it can serve more sessions:

$$S \;=\; c_{AS}(x).\mathit{case}\ x\ \mathit{of}\ \{y\}_{K_{AS}}\ \mathit{in}\ \overline{c_{BS}}\{y\}_{K_{BS}}.S$$

and we specify the enemy as:

$$E \;=\; c_{BS}(x).c_{AB}(y).\overline{c_{BS}}x.\overline{c_{AB}}y.$$

Note that the names c_{AB}, c_{AS} and c_{BS} are free, i.e., known to all the processes of P'.

Now we can observe the attack sequence in Fig. 5. Note in particular that when B' decrypts $||_1||_1\bullet||_0||_0\{M\}_{K_{AB}}$, it obtains $||_1||_1\bullet||_0||_0 M$ which is a message originated from A and not from A', as it should be. Indeed, the address of A' relative to B' is $||_1||_1\bullet||_0||_1$, and the attack is detected.

6. Conclusions and Future Work

We defined a primitive that guarantees message authentication over public channels. It is based on a semantics of the π-calculus where names, representing messages or channels, are localized, whenever needed. Localized names have a reference to their creator and the semantic rules update them, after a communication. This reference is the relative address of a message M. The authentication primitive compares it with the address of a process A. The check succeeds and authenticates the message M only if the process A originated M.

The same mechanism can also be applied in case one is interested in making sure which principal encrypted the message received, under the hypothesis that the same entity also generated the message or has been delegated to send it. This is a somehow weaker form of authentication than the one considered in this paper, which assumes that a key can be used by its trusted owners, only. We make use of this restricted version of authentication in the discussion below.

Note that our authentication primitive does not transform a public channel into a private one. It neither guarantees that the received messages have not been intercepted nor eavesdropped, i.e., it does not guarantee secrecy. Indeed, our primitive corresponds neither to a private channel in the basic π-calculus, nor to a cryptographic one in the *spi*-calculus. These appear to be too strong to model authentication alone, as they guarantee also secrecy.

As discussed in the paper, our primitive may be not implementable directly. Indeed, one should have a low-level, highly reliable mechanism to manage localized names, which is quite unrealistic in many cases. Nevertheless, our proposal can help reasoning on authentication and security from an abstract point of view. We briefly exemplified this using the *spi*-calculus. We plan to apply the proof techniques presented in [1] to our model. The idea is to define a

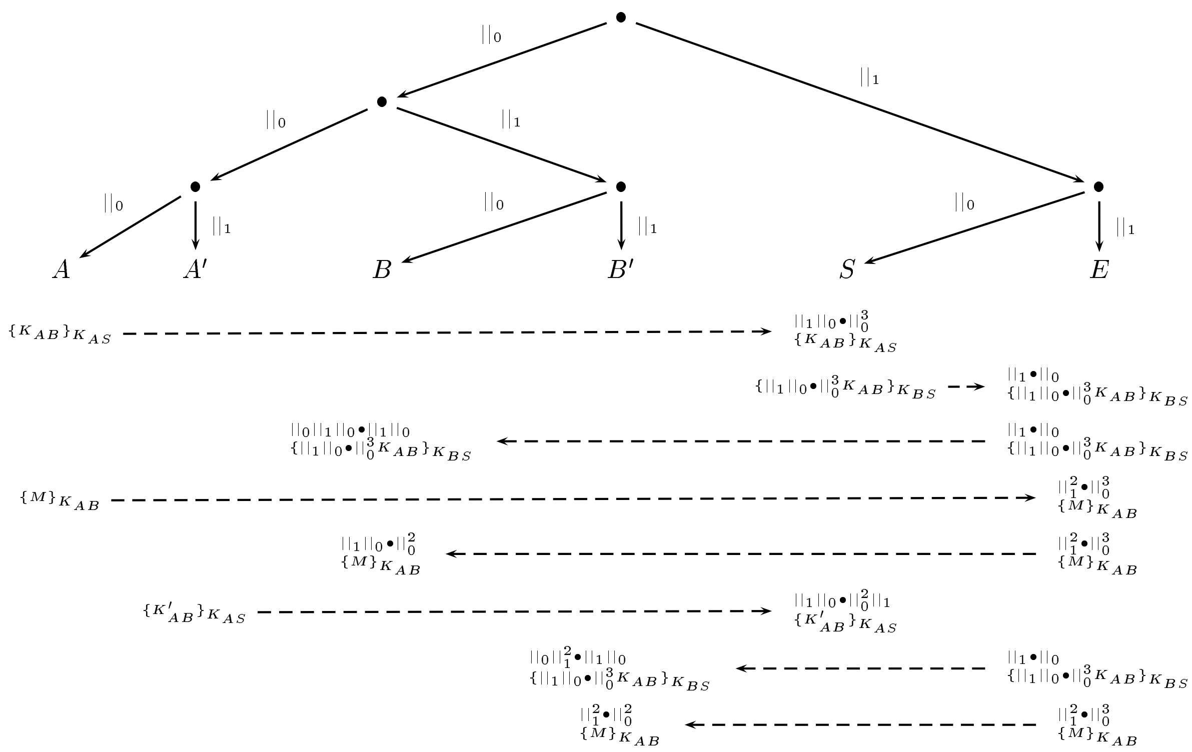

Figure 5. An attack on the Wide Mouthed Frog protocol; where $||_i^k$ stands for a sequence of k tags $||_i$.

suitable equivalence $\approx$ on our processes, and then check if a certain protocol guarantees authentication by comparing it with its corresponding "secure by construction" version. This secure version is easily got by adding the authentication primitive at the end of the protocol, i.e., where the message which should be authentic is derived. As an example, the process B in the Wide Mouthed Frog protocol becomes:

$$\begin{array}{rcl} B' & = & c_{BS}(x).\mathit{case}\ x\ \mathit{of}\ \{y\}_{K_{BS}}\ \mathit{in} \\ & & c_{AB}(z).\mathit{case}\ z\ \mathit{of}\ \{w\}_y\ \mathit{in}\ [w \overset{@}{=} A]B''(w) \end{array}$$

which makes the protocol secure by construction. So, if we establish that

$$(A|B)|S \approx (A|B')|S,$$

we are guaranteed that also the original protocol $(A|B)|S$ is secure.

The authentication property studied in [1] seems slightly stronger than ours: it requires that if A has to send a message to B, then this should be the only message that B accepts at the end of (a successful run of) the protocol. Our approach to authentication relaxes this requirement. As a matter of fact, we only ask that the message received by B has been indeed originated by the process A. We have shown on an example how this feature can be exploited to reveal replay attacks. Further investigation is needed for comparing Abadi's requirement and property with ours.

Besides the technique mentioned above, our authentication primitive could support proofs of correctness that are split in a sequence of "small" steps. A simple example follows. Consider the transmission of an encrypted message M on a public channel in the spi-calculus:

$$\begin{array}{rcl} P & = & (\nu K_{AB})(\ (\nu M)\overline{c}\{M\}_{K_{AB}}\ | \\ & & c(x).\mathit{case}\ x\ \mathit{of}\ \{y\}_{K_{AB}}\ \mathit{in}\ B'(y)\ ), \end{array}$$

that enables $B'(y)$ only if $y = M$. Its secure version is:

$$\begin{array}{rcl} P' & = & (\nu K_{AB})(\ (\nu M)\overline{c}\{M\}_{K_{AB}}\ | \\ & & c(x).\mathit{case}\ x\ \mathit{of}\ \{y\}_{K_{AB}}\ \mathit{in}\ [y \overset{@}{=} A]B'(y)\ ) \end{array}$$

where we assume A to be (the address of) the process $(\nu M)\overline{c}\{M\}_{K_{AB}}$. Instead of checking P against P', we could prove it is equivalent to

$$\begin{array}{rcl} P'' & = & (\nu K_{AB})\,(\nu M)(\overline{c}\{M\}_{K_{AB}}\ | \\ & & c(x).[x \overset{@}{=} A]\mathit{case}\ x\ \mathit{of}\ \{y\}_{K_{AB}}\ \mathit{in}\ B'(y)\ ) \end{array}$$

The check $P'' \approx P$ seems easier to prove than $P' \approx P$. Indeed, P'' makes explicit that P uses encryption to guarantee authentication: when B receives a message encrypted with K_{AB}, it knows that the encryption (not the encrypted message!) was originated by A. This constitutes the first step of the equivalence proof where we are still not interested in "what is encrypted". The next step consists of proving the equivalence $P'' \approx P'$, by observing that the only message which is encrypted with K_{AB} is actually originated by A. The steps above rely on the fact that K_{AB} is private to A and B, and exploit the weak notion of authentication mentioned above. Of course, devising the intermediate processes, like P'' above, could be non trivial, but we are confident that also their generation could be mechanized.

The idea of exploiting locations for the analysis of authentication comes from [6], where however entities are bound to physical addresses of the net. Here we relaxed the rigidity of that mapping, by introducing a sort of "identifiers of sites" represented by relative addresses. As a matter of fact, the actual placement of a process on a site can be recovered by composing our localized names (akin to the environment function of sequential languages) with allocation tables (similar to a store). Actually, [6] models a wider notion of authentication, that we plan to investigate next.

We feel that our authentication primitive also works when processes migrate while executing a communication protocol. Our claim is supported by [2], where the localized semantics of the (first order) π-calculus is extended to the Higher Order π-calculus [18], that describes explicitly code mobility, by permitting also processes to be exchanged in communications.

Finally, we feel confident that our proposal scales up, because many real languages for concurrency, like Facile [21], PICT [16], CML [15], are built on top of a core process calculus like the one we use here. Of course a great deal of work is still necessary to make our theoretical proposal applicable in real cases.

References

[1] M. Abadi and A. D. Gordon. "A Calculus for Cryptographic Protocols: The Spi Calculus". *Information and Computation 148*, 1:1–70, January 1999.

[2] C. Bodei, P. Degano, and C. Priami. "Names of Mobile Agents Handled Locally". To appear in *Theoretical Computer Science*. Also available at http://www.di.unipi.it/~chiara/publ-40/TCS.ps. Extended abstract in: *Proceedings of ICALP'96*, LNCS 1099, Springer-Verlag.

[3] M. Burrows, M. Abadi, and R. Needham. "A Logic of Authentication". *ACM Transactions on Computer Systems*, pages 18–36, Feb. 1990.

[4] A. Durante, R. Focardi, and R. Gorrieri. "CVS: A Compiler for the Analysis of Cryptographic Protocols". In *this volume*, 1999.

[5] F. J. T. Fabrega, J. C. Herzog, and J. D. Guttman. "Strand Spaces: Why is a Security Protocol Correct?". In *Proceedings of the 1998 Symposium on Security and Privacy*. IEEE Computer Society Press, May 1998.

[6] R. Focardi. "Using Entity Locations for the Analysis of Authentication Protocols". In *Proceedings of Sixth Italian Conference on Theoretical Computer Science (ICTCS'98)*, November 1998.

[7] R. Focardi, A. Ghelli, and R. Gorrieri. "Using Non Interference for the Analysis of Security Protocols ". In *Proceeding of the DIMACS Workshop on Design and Formal Verification of Security Protocols (H. Orman and C. Meadows Ed.)*, DIMACS Center, Rutgers University, September 1997.

[8] I. O. for Standardization. Information technology - Security techniques - Entity authentication machanism; Part 1: General model. ISO/IEC 9798-1, Second Edition, September 1991.

[9] R. Kemmerer, C. Meadows, and J. Millen. "Three systems for cryptographic protocol analysis". *J. Cryptology*, 7(2):79–130, 1994.

[10] G. Lowe. "Breaking and Fixing the Needham-Schroeder Public-key Protocol using FDR". In *Proceedings of Second International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, pages 146–166, Passau (Germany), March 1996. Springer-Verlag, LNCS 1055.

[11] G. Lowe. "A Hierarchy of Authentication Specification". In *Proceedings of the 10th Computer Security Foundation Workshop*. IEEE press, 1997.

[12] R. Milner, J. Parrow, and D. Walker. "A Calculus of Mobile Processes (I and II)". *Information and Computation*, 100(1):1–77, 1992.

[13] R. Milner, J. Parrow, and D. Walker. "Modal Logics for Mobile Processes". *Theoretical Computer Science*, 114:149–171, 1993.

[14] J. C. Mitchell, M. Mitchell, and U. Stern. "Automated Analysis of Cryptographic Protocols Using Murϕ". In *Proceedings of the 1997 IEEE Symposium on Research in Security and Privacy*, pages 141–153. IEEE Computer Society Press, 1997.

[15] F. Nielson and H. R. Nielson. "From CML to its Process Algebra". *Theoretical Computer Science*, 155:179–219, 1996.

[16] B. Pierce and D. Turner. "PICT: A programming language based on the pi-calculus". Technical report available electronically, 1995.

[17] T. Pratt. *Programming Languages: Design and Implementation*. Prentice Hall, 1984.

[18] D. Sangiorgi. *"Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms"*. PhD thesis, University of Edinburgh, 1992.

[19] S. Schneider. "Verifying authentication protocols in CSP". *IEEE Transactions on Software Engineering*, 24(9), September 1998.

[20] B. Schneier. *Applied Cryptography*. John Wiley & Sons, Inc., 1996. second edition.

[21] B. Thomsen, L. Leth, S. Prasad, T.-M. Kuo, A. Kramer, F. Knabe, and A. Giacalone. "Facile Antigua Release Programming Guide". Technical Report ECRC-93-20, European Computer-Industry Research Centre, 1993.

A. The π-calculus

The late operational semantics of the π-calculus is defined in SOS (Structural Operational Semantics) style, and the labels of transitions are τ for silent actions, $x(y)$ for input, $\overline{x}y$ for free output, and $\overline{x}(y)$ for bound output. We will use μ as a metavariable for the labels of transitions (it is distinct from π, the metavariable for prefixes, though it coincides in three cases).

We recall the notion of free names *fn*(μ), bound names *bn*(μ), and names *names*$(\mu) =$ *fn*$(\mu) \cup$ *bn*(μ) of a label μ. Two functions, *sbj* and *obj*, are also defined giving the subject and the object of input and output actions, respectively.

μ	$Kind$	*fn*(μ)	*bn*(μ)	*sbj*(μ)	*obj*(μ)
τ	Silent move	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$\overline{x}y$	Free output	$\{x, y\}$	$\emptyset$	$\{x\}$	$\{y\}$
$\overline{x}(y)$	Bound output	$\{x\}$	$\{y\}$	$\{x\}$	$\{y\}$
$x(y)$	Input	$\{x\}$	$\{y\}$	$\{x\}$	$\{y\}$

Functions *fn*, *bn* and *names* are extended in the obvious way to processes.

Below we assume the *structural congruence* $\equiv_P$ on processes, defined as the least congruence satisfying the following clauses:

- $P \equiv_P Q$ if P and Q are α-equivalent,
- $(\mathcal{P}/_{\equiv_P}, +, \mathbf{0})$ and $(\mathcal{P}/_{\equiv_P}, |, \mathbf{0})$ are commutative monoids,
- $[x = x]P \equiv_P P$,
- $(\nu\, x)(\nu\, y)P \equiv_P (\nu\, y)(\nu\, x)P$, $(\nu\, x)P \equiv_P P$ if $x \notin$ *fn*(P), $(\nu\, x)(P \,|\, Q) \equiv_P (\nu\, x)P \,|\, Q$ if $x \notin$ *fn*(Q), and
- $A(a_1, \ldots, a_k) \equiv_P P\{a_1/y_1, \ldots, a_k/y_k\}$, if $A(y_1, \ldots, y_k) = P$.

A *variant* of $P \xrightarrow{\mu}_L Q$ is a transition which only differs in that P and Q have been replaced by structurally congruent processes, and μ has been α-converted, where a name bound in μ includes Q in its scope.

Table 1 shows the late transition system of the π-calculus. The transition in the conclusion of each rule, as well as in the axiom, stands for all its variants.

B. Handling names locally

We recall the late operational semantics of the π-calculus of [2], which creates and handles names locally.

Definition B.1 *Let* $\mathcal{N}' = \mathcal{A} \cdot \mathbf{Z} \cup \mathcal{N} \cup \{\tau\}$ *be the set of* localized names, *ranged over by* $r, s, u, \ldots$, *possibly indexed, where* $\mathbf{Z}$ *is the set of non-zero integer numbers, and "$\cdot$" is the operator of language concatenation.*
The new syntax of the π-calculus is obtained from the one in Def. 2.1, with r, s in place of x, y. In particular, π can either be $r(s)$ or $\overline{r}s$ or τ and a restricted name can be s. Furthermore, we only assume on the new processes the least congruence induced by the monoidal laws for $+$, and the laws for matching and constant invocation.

$$act : \mu.P \xrightarrow{\mu}_L P$$

$$par : \frac{P \xrightarrow{\mu}_L P'}{P|Q \xrightarrow{\mu}_L P'|Q}, bn(\mu) \cap fn(Q) = \emptyset \qquad sum : \frac{P \xrightarrow{\mu}_L P'}{P + Q \xrightarrow{\mu}_L P'}$$

$$res : \frac{P \xrightarrow{\mu}_L P'}{(\nu x)P \xrightarrow{\mu}_L (\nu x)P'}, x \notin names(\mu) \qquad open : \frac{P \xrightarrow{\overline{x}y}_L P'}{(\nu y)P \xrightarrow{\overline{x}(y)}_L P'}, y \neq x$$

$$close : \frac{P \xrightarrow{\overline{x}(y)}_L P', Q \xrightarrow{x(y)}_L Q'}{P|Q \xrightarrow{\tau}_L (\nu y)(P'|Q')} \qquad com : \frac{P \xrightarrow{\overline{x}y}_L P', Q \xrightarrow{x(z)}_L Q'}{P|Q \xrightarrow{\tau}_L P'|Q'\{y/z\}}$$

Table 1. Late proved transition system of π-calculus.

We now lift the composition of addresses defined in Sect. 4 to exportation of names.

Definition B.2 *Let $r = \vartheta_r \bullet \vartheta'_r n$ (resp. a) be a name. A name r exported to the relative address $\vartheta \bullet \vartheta'$ is $\vartheta \bullet \vartheta' \star r = (\vartheta \bullet \vartheta' \star \vartheta_r \bullet \vartheta'_r)n$ (resp. $\vartheta \bullet \vartheta' \star a = a$).*

Below, we extend the needed notions to processes with localized names implemented as a family of operators $n \Rightarrow$ (with $n \geq 1$).

Definition B.3 *Let P be a process and n be a non-zero natural number. The* extended processes *(denoted by $t, t_i, t' \ldots \in \mathcal{T}$), are defined according to the syntax*

$$t ::= n \Rightarrow P \mid (\nu s)t \mid t|t$$

where P is as in Def. B.1. We assume on $\mathcal{T}$ the least congruence $\equiv$ which satisfies the following clauses

- $n \Rightarrow (\nu\, s)P \equiv (\nu\, s)n \Rightarrow P$ *and*
- $n \Rightarrow (P|Q) \equiv (n \Rightarrow P)|(n \Rightarrow Q)$.

The computations of a process $P \in \mathcal{P}$ start from the extended process $1 \Rightarrow P$.

The standard notions on names (free, bound) are carried over extended processes in the obvious way. We also need a new notion of substitution to correctly update localized names.

Definition B.4 *Let $\{-/-\}$ be the standard substitution. Then the* routed substitution $\{\!|-/-|\!\}$ *is defined by induction as follows:*

1. $(n \Rightarrow P)\{\!|u'/u|\!\} = n \Rightarrow P\{\!|u'/u|\!\}$
2. $((\nu\, s)t)\{\!|u'/u|\!\} = \begin{cases} (\nu\; -u)(t\{\!|-u/u'|\!\})\{\!|u'/u|\!\} & \\ & \textit{if } u' = s \\ (\nu\, s)t & \textit{if } u = s \\ (\nu\, s)(t\{\!|u'/u|\!\}) & \textit{otherwise} \end{cases}$
3. $(t_0|t_1)\{\!|u'/u|\!\} = t_0\{\!|u'_0/u_0|\!\}|t_1\{\!|u'_1/u_1|\!\}$, *where $u'_i = ||_i \bullet \star u'$ and $u_i = ||_i \bullet \star u$, for $i = 0, 1$*
4. $\mathbf{0}\{\!|u'/u|\!\} = \mathbf{0}$
5. $(\tau.P)\{\!|u'/u|\!\} = \tau.P\{\!|u'/u|\!\}$
6. $(\overline{r}s.P)\{\!|u'/u|\!\} = (\overline{r}s)\{u'/u\}.P\{\!|u'/u|\!\}$
7. $(r(s).P)\{\!|u'/u|\!\} = \begin{cases} r\{u'/u\}(-u).P\{\!|-u/u'|\!\}\{\!|u'/u|\!\} & \\ & \textit{if } u' = s \\ r\{u'/u\}(s).P & \textit{if } u = s \\ r\{u'/u\}(s).(P\{\!|u'/u|\!\}) & \textit{otherwise} \end{cases}$
8. $(P + Q)\{\!|u'/u|\!\} = P\{\!|u'/u|\!\} + Q\{\!|u'/u|\!\}$
9. $[r = s]P\{\!|u'/u|\!\} = [r\{u'/u\} = s\{u'/u\}]P\{\!|u'/u|\!\}$
10. $A(r_1, \ldots, r_k)\{\!|u'/u|\!\} = A(r_1\{u'/u\}, \ldots, r_k\{u'/u\})$

We use the above definition to implement a selective substitution that works on a given sub-process of a whole term.

Definition B.5 *Let t be an extended process and $\vartheta \in \{||_0, ||_1\}^*$. Then, the* selective rooted substitution $t\{\!|/|\!\}_{@\vartheta}$ *is defined by induction as*

- $(t_0|t_1)\{\!|r/s|\!\}_{@\vartheta} = \begin{cases} t_0\{\!|r/s|\!\}_{@\vartheta'}|t_1 & \textit{if } \vartheta = ||_0\vartheta' \\ t_0|t_1\{\!|r/s|\!\}_{@\vartheta'} & \textit{if } \vartheta = ||_1\vartheta' \end{cases}$
- $t\{\!|r/s|\!\}_{@\epsilon} = t\{\!|r/s|\!\}$

The selective increment *of counters $t\{n{+}1 \Rightarrow /n \Rightarrow\}_{@\vartheta}$ is defined as follows, by stipulating that $(n{+}1 \Rightarrow)$ and $(n \Rightarrow)$ are considered as tokens*

- $(t_0|t_1)\{n + 1 \Rightarrow /n \Rightarrow\}_{@||_i\vartheta} = t_i\{n + 1 \Rightarrow /n \Rightarrow\}_{@\vartheta}$
- $t\{n + 1 \Rightarrow /n \Rightarrow\}_{@\epsilon} = t\{n + 1 \Rightarrow /n \Rightarrow\}$.

The late transition system for the π-calculus with localized names is in Tab. 2. Note that our rules have *no global* conditions on names.

$$Out : n \Rightarrow \overline{r}s.P \xrightarrow{\overline{r}s} n \Rightarrow P \qquad In : n \Rightarrow r(s).P \xrightarrow{r(\bullet n)} (n+1) \Rightarrow P\{\!|\bullet n/s|\!\}$$

$$Tau : n \Rightarrow \tau.P \xrightarrow{\tau} n \Rightarrow P$$

$$Par_0 : \frac{t_0 \xrightarrow{\mu @ \vartheta} t_0'}{t_0|t_1 \xrightarrow{\mu @ ||_0 \vartheta} t_0'|t_1} \quad Par_1 : \frac{t_1 \xrightarrow{\mu @ \vartheta} t_1'}{t_0|t_1 \xrightarrow{\mu @ ||_1 \vartheta} t_0|t_1'} \quad Sum : \frac{n \Rightarrow P \xrightarrow{\mu @ \vartheta} t'}{n \Rightarrow (P+Q) \xrightarrow{\mu @ \vartheta} t'}$$

$$Res : \frac{t \xrightarrow{\mu @ \vartheta} t'}{(\nu u)t \xrightarrow{\mu @ \vartheta} (\nu u)t'},\ \mu \neq \tau \ \Rightarrow \begin{cases} u \notin \{r, s\} \\ r = \bullet\vartheta \star sbj(\mu) \\ s = \bullet\vartheta \star obj(\mu) \end{cases}$$

$$Com_0 : \frac{t_0 \xrightarrow{\overline{r_0}s_0 @ \vartheta_0} t_0', t_1 \xrightarrow{r_1(\bullet m) @ \vartheta_1} t_1'}{t_0|t_1 \xrightarrow{\tau} t_0'|t_1'\{\!|s_1/\bullet m|\!\}_{@\vartheta_1}}, \begin{cases} s_1 = ||_1\vartheta_1\bullet||_0\vartheta_0 \star s_0 \\ r_1 = ||_1\vartheta_1\bullet||_0\vartheta_0 \star r_0 \end{cases}$$

$$Com_1 : \frac{t_0 \xrightarrow{r_1(\bullet m) @ \vartheta_1} t_0', t_1 \xrightarrow{\overline{r_0}s_0 @ \vartheta_0} t_1'}{t_0|t_1 \xrightarrow{\tau} t_0'\{\!|s_1/\bullet m|\!\}_{@\vartheta_1}|t_1'}, \begin{cases} s_1 = ||_0\vartheta_1\bullet||_1\vartheta_0 \star s_0 \\ r_1 = ||_0\vartheta_1\bullet||_1\vartheta_0 \star r_0 \end{cases}$$

$$Open : \frac{t \xrightarrow{\overline{r}s @ \vartheta} t'}{(\nu u)t \xrightarrow{\overline{r}(\bullet n) @ \vartheta} (t'\{n+1 \Rightarrow / n \Rightarrow\}_{@\vartheta})\{\!|\bullet\vartheta n/u|\!\}}, \begin{cases} u = \bullet\vartheta \star s \\ r \neq s \end{cases}$$

$$Close_0 : \frac{t_0 \xrightarrow{\overline{r_0}(s_0) @ \vartheta_0} t_0', t_1 \xrightarrow{r_1(\bullet m) @ \vartheta_1} t_1'}{t_0|t_1 \xrightarrow{\tau} (\nu u)(t_0'|t_1'\{\!|s_1/\bullet m|\!\}_{@\vartheta_1})}, \begin{cases} s_1 = ||_1\vartheta_1\bullet||_0\vartheta_0 \star s_0 \\ r_1 = ||_1\vartheta_1\bullet||_0\vartheta_0 \star r_0 \\ u = \bullet||_0\vartheta_0 \star s_0 \end{cases}$$

$$Close_1 : \frac{t_0 \xrightarrow{r_1(\bullet m) @ \vartheta_1} t_0', t_1 \xrightarrow{\overline{r_0}(s_0) @ \vartheta_0} t_1'}{t_0|t_1 \xrightarrow{\tau} (\nu u)(t_0'\{\!|s_1/\bullet m|\!\}_{@\vartheta_1}|t_1')}, \begin{cases} s_1 = ||_0\vartheta_1\bullet||_1\vartheta_0 \star s_0 \\ r_1 = ||_0\vartheta_1\bullet||_1\vartheta_0 \star r_0 \\ u = \bullet||_1\vartheta_0 \star s_0 \end{cases}$$

Table 2. Late transition system of the π-calculus with localized names.