

# A Spatial-Epistemic Logic and Tool for Reasoning about Security Protocols

Bernardo Toninho and Luís Caires

CITI and Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa

**Abstract.** Reasoning about security properties involves reasoning about *where* the knowledge of the several principals of a system is located, and how it evolves over time. Building on this observation, we introduce a framework for security protocol analysis based on dynamic spatial logic specifications. Our underlying computational model is a variant of the applied pi-calculus, while specifications are expressed in a dynamic spatial logic extended with an epistemic operator. We formally present the syntax and semantics of the model and logic, and discuss the expressiveness of the approach. We also prove that general attackers may be determined for any finite protocol in our framework, and discuss how this result may be used to reason about security properties of open systems. We also present a model-checking algorithm for our logic, which has been implemented as an extension to the SLMC system.

## 1 Introduction

Among the several artifacts in the field of computer security, security protocols are indubitably a fundamental subject of study and research [1–3] due to their importance in the security of systems. Security protocols serve a variety of purposes, ranging from secrecy and authentication to forward secrecy and deniable encryption, but despite their varied goals, a common trait of security protocols is their notoriously difficult design and analysis, ultimately leading to security vulnerabilities [4].

In light of this issue, it becomes essential to develop techniques that ensure the correctness of protocols, with respect to some specification of the properties they aim to establish. Thus, a wide range of language-based techniques have been proposed to analyze and verify protocols and their correctness, such as type systems, process calculi or static analysis [5–8]. These techniques allow one to verify that a model of a protocol conforms to some abstract specification, with varying degrees of expressiveness.

In this paper we propose a novel framework for protocol analysis based on process calculus models and logic specifications. While the usage of process calculi and logic in this context is not new [9, 10, 8], our approach stems from the fact that, when reasoning about security, we often reason about the knowledge of the several principals of a system (sub-systems), and to the best of our knowledge, this type of reasoning has yet to fully make its way to the verification landscape in a precise manner.

We propose a novel dynamic spatial epistemic logic that allows reasoning about systems at three levels: the *dynamics* of systems and subsystems, the *spatial* arrangement of systems and subsystems, and the *knowledge* (obtainable information) of systems and subsystems. To motivate our approach, consider the example of Fig. 1.

We have a system composed by two processes,  $P$  and  $Q$ , sharing a symmetric key  $k$ .  $P$  wants to send value  $v$  to  $Q$ . It first sends a fresh nonce  $N$  to ensure the freshness

```

P(k) = new N in c!(N).c?(x).let y = dec(x,k) in
      [N = y].c!(enc(pair(v,N),k));

Q(k) = c?(x).c!(enc(x,k)).c?(y).let m = dec(y,k) in
      let v = fst(m) in let n = snd(m) in
      [n = x].ok!();

Sys = new k in (P(k) | Q(k));
World = Sys | Attacker(Sys);

prop attackerK = not eventually inside
  (2 | (1 and @attacker and knows v));
prop pqK = eventually inside
  ((2 and knows v) | (1 and @attacker and not (knows v)));

check World |= attackerK;

- Time elapsed: 0.420781 secs - Number of state visits: 14866 -
* Process World satisfies the formula attackerK *

check World |= pqK;

- Time elapsed: 0.001366 secs - Number of state visits: 20 -
* Process World satisfies the formula pqK *

```

**Fig. 1.** A Motivating Example

of the run, which  $Q$  receives and sends back to  $P$ , encrypted with  $k$ . If  $P$  receives the correct value, it will send  $v$  and  $N$  encrypted with  $k$ .  $Q$  will receive the encrypted pair, test the previously received nonce against the value in the pair and finally, obtain  $v$ .

When considering this protocol between  $P$  and  $Q$ , its main goal is to allow the secure communication of  $v$  from  $P$  to  $Q$ . Clearly, for the protocol to be correct, it must be the case that a malicious agent running with the system never knows value  $v$  and that  $P$  and  $Q$  eventually both end up knowing  $v$ . It is precisely this type of reasoning that we enable with our approach. Using our dynamic spatial epistemic logic, it is possible to precisely and directly state the correctness properties that appear naturally when reasoning about security, such as “an attacker can never know  $v$ ” or “ $P$  and  $Q$  eventually know  $v$ ”.

In our example, property *attackerK* states that the world (the system with an attacker) never reaches a configuration in which the subsystem corresponding to the attacker can know  $v$ . Property *pqK* from our example states that the world eventually reaches a configuration where subsystems  $P$  and  $Q$  know  $v$ , while the attacker does not. Notice how both properties correspond very clearly to the previously stated informal correctness properties, but now in a formal and precise setting. This is so due to the combination of spatial, dynamic and epistemic connectives. Spatial connectives give the ability to mention subsystems ( $P$ ,  $Q$ , the attacker), dynamic connectives give the ability to range over evolving configurations (“eventually”, “always”) and epistemic connectives provide the ability of reasoning about knowledge (“knows  $v$ ”).

By combining these types of reasoning, we tap the potential of reasoning about security by reasoning precisely about the knowledge of principals, which up to this point had not been fully achieved. To achieve this, we use a process calculus model for security similar to existing  $\pi$ -calculi [5, 8], where we can model principals and attackers as processes running in a distributed setting. We then combine the fundamental traits of dynamic spatial logic with epistemic connectives to produce a logic that can reason

about the knowledge of principals, allowing us to state properties such as “principal P can know the value  $v$ ” or “an attacker never knows the key  $K$ ”.

Our framework supports the explicit modeling of attackers. For automatic verification purposes, the need to explicitly model attackers is not so convenient, since it often requires the modeler to guess *a priori* what kind of attacks may be feasible or interesting. Notice however, that our technique for modeling attackers does not require a complete specification of the attacker behavior, since we provide primitives, namely the attacker output action, that greatly simplifies the task. In fact, we prove that it is possible to *automatically* generate a general attacker for any finite protocol; the generated attacker behaves as an arbitrary Dolev-Yao adversarial environment. To fully automate our technique, further work is needed (as discussed in Section 4.1); the focus of the current paper being essentially on the expressiveness issues. In any case, we already provide tool support for arbitrary passive attackers and for *bounded* Dolev-Yao attackers (where the bound concerns the size of generated messages); this technique can already be used to automatically find attacks, eg., as illustrated in the example of Section 4.2.

In particular, we also show how we can use our framework to verify correspondence assertions (an established technique for verifying authenticity). We also establish decidability of model-checking for our logic for a relevant class of processes (and cryptographic operations). This result enabled an implementation of our framework as an extension to the SLMC tool, which provides, to our best knowledge, the first model-checker for a variant of the applied- $\pi$  calculus against an expressive spatial-behavioral (and epistemic) logic.

The technical contributions of this work are the following ones. We develop a process calculus model for security protocols (Section 2.2), inspired in existing  $\pi$ -calculi, supporting explicit modeling of adversarial agents, at an adequate level of abstraction. We also introduce a new dynamic spatial epistemic logic (Section 3), oriented towards reasoning about spatial distribution of knowledge. We develop a logic based formal theory of knowledge deduction (Section 3.2) for our  $\pi$ -calculus models, proved sound, complete and decidable. This presentation was used in our model-checking algorithms. We discuss our attacker representations (Section 4.1), and how it is possible to produce a generic Dolev-Yao attacker for finite protocols. We also show how we can model and verify correspondence assertions (Section 4.2) in our framework. Finally, we implemented of a model-checking algorithm for the logic as an extension to the SLMC tool, producing the first tool aimed at security protocol analysis using spatial logic model checking. The proofs of our technical results are detailed in [11].

## 2 Process Model

In this section we introduce our process model, starting with some preliminary notions on terms and equational theory and then introducing our process calculus.

## 2.1 Terms and Equational Theories

Data exchanged by processes is modeled by terms of a term algebra, as usual. In order to capture cryptographic operations, and data structuring, we will consider term algebras with equations, that is, equational theories (cf. [8]).

We assume an infinite set of variables ranged over by  $x, y, z$ , an infinite set of names  $\Lambda$  ranged over by  $m, n$  and range over terms with  $s, t, v$ . Terms are defined from names and variables by applying function symbols. Function symbols come equipped with an arity. We thus consider a given term algebra to be defined from a signature  $\Sigma$  and an equational theory  $E$  that defines the “semantics” of the function symbols in  $\Sigma$ . An equational theory is a congruence relation defined by a set of equations of form  $t = s$ .

In certain circumstances, an equational theory may give rise to a set of rewrite rules by orienting each equation to produce the rule  $t \rightarrow s$ , in such a way that two terms are equal modulo  $E$  whenever that have a common reduct under rewriting. This is the case of subterm convergent equational theories [12], which are the ones that we will consider here. A subterm convergent system is a convergent rewrite system in which in every rewrite rule the right-hand side is a proper subterm of the left-hand side. In this paper, we will assume a general rewrite theory  $\mathcal{R}$  subject to the conditions above. Given a rewrite rule  $t \rightarrow s$ , we call the outermost function symbol in  $t$  a *destructor*, since the application of the rule may open the internal structure of inner terms in  $t$  to produce the term  $s$ . We classify the remaining function symbols, that never occur as a destructor, as *constructors*. For example, for signature  $\Sigma \triangleq \{\text{enc}/2; \text{dec}/2\}$  and equational theory  $E \triangleq \{\text{dec}(\text{enc}(x, y), y) = x\}$ ,  $\text{dec}$  is a destructor and  $\text{enc}$  a constructor. We range over constructors with  $f$  and destructors with  $\delta$ .

We denote the set of names of a term  $T$  by  $\text{names}(T)$  and the depth of a term as  $|T|$  (the depth is the length of the longest path in the tree representation of the term). We state that a term is closed if it does not contain variables. We denote by  $=_E$  the reflexive transitive closure of single-step rewrite. We write  $\mathfrak{F}(\psi)$  for the DY (Dolev-Yao) equational closure of a set of terms  $\psi$ , that is, the set of all values generated by terms of  $\psi$  through function application, modulo the equational theory. This closure represents all possible information that may be produced from a set of terms while following the rules of the equational theory, which if we interpret a set of terms as a set of messages, is the usual notion of knowledge from the Dolev-Yao model.

**Definition 1 (Equational Closure).** *Given a rewrite theory  $\mathcal{R}$ , the DY equational closure of a set of terms  $\psi$ , noted  $\mathfrak{F}(\psi)$ , is the least set of terms such that:*

1.  $\psi \subseteq \mathfrak{F}(\psi)$
2.  $\forall f \in \Sigma$ . if  $f$  a constructor and  $t_1, \dots, t_k \in \mathfrak{F}(\psi)$  then  $f(t_1, \dots, t_k) \in \mathfrak{F}(\psi)$
3.  $\forall \delta \in \Sigma$ . if  $\delta$  a destructor and  $t_1, \dots, t_k \in \mathfrak{F}(\psi)$  and  $\delta(t_1, \dots, t_k) \rightarrow t'$  then  $t' \in \mathfrak{F}(\psi)$

When interpreting the DY equational closure of a set of terms as obtainable knowledge, we can state knowledge derivation through term derivation.

**Definition 2. (Knowledge Derivation).** *Given sets of terms  $\psi$  and  $\phi$ , we say that  $\phi$  may be derived from  $\psi$  (written  $\psi \models \phi$ ) if and only if  $\phi \subseteq \mathfrak{F}(\psi)$ .*

The general idea is that one may can derive a piece of information if it can be generated by combining pieces of information using the rules of the equational theory. Given these

$P, Q ::= \mathbf{0}$	(Null Process)	$\alpha ::= m(x)$	(Input)
$P \mid Q$	(Parallel Composition)	$m\langle T \rangle$	(Output)
$(\nu n)P$	(Name Restriction)	$m\langle * \rangle$	(Attacker Output)
$\alpha.P$	(Action Prefix)	$[T_1 = T_2]$	(Test)
$P + Q$	(Choice)	$T ::= n$	(Name)
$\text{let } x = T \text{ in } P$ (Let Construct)		$x$	(Variable)
		$f(T_1, \dots, T_n)$	(Function)

**Fig. 2.** Process Calculus Syntax

basic notions relative to terms, equational theories, and knowledge derivation, we may now present our process calculus model.

## 2.2 Process Calculus

It is known that the high level of abstraction of the  $\pi$ -calculus, convenient from a foundational perspective, is not so adequate for modeling cryptographic techniques as necessary for analyzing security protocols.

We adopt an extension to the  $\pi$ -calculus that extends the base values of the language with functional terms (cf. Section 2.1), that can be seen as a fragment of the Applied  $\pi$ -calculus [8]. We choose this calculus over the applied  $\pi$ -calculus mainly for simplicity reasons, not requiring active substitutions nor frames given that we may use our logic to observe terms.

We model cryptographic operations by defining such operations in a term algebra. The calculus is thus aimed at the explicit modeling of agents involved in security protocols, both principals and adversaries. Principals are modeled standardly, using terms to model cryptographic terms. Adversaries are modeled as processes (cf. Section 4.1) using the attacker output prefix - a non-deterministic output of terms that can be generated from known values, which enables reasoning directly about attacker knowledge using our logic.

**Definition 3 (Processes).** *Given a signature  $\Sigma$ , an infinite set of names ranged over by  $m, n$ , and an infinite set of variables ranged over by  $x, y, z$ , the set of processes  $(P, Q)$ , of actions  $\alpha$  and of terms  $T$  are defined in Fig. 2.*

Before introducing the semantics of our calculus, we present some definitions that pertain to obtaining the *relevant terms* of a process that are necessary for the semantics of our attacker output and for our logic. To obtain the values of a process, we define a relation  $\vdash_k$  that extracts the set of values  $\psi$  that occur in a process ( $P \vdash_k \psi$ ). However, some care is needed in the definition of  $\vdash_k$  since a term may contain bound names or variables. For instance, in the process  $a(x).a(\text{dec}(x, k)).\mathbf{0}$ , the term  $\text{dec}(x, k)$  is not a proper value since it contains the variable  $x$ . In these situations, our extraction has to be such that it will produce a set containing  $k$  but not  $x$  (nor  $\text{dec}(x, k)$ ). Similarly, when we consider the terms that are to be the object of our attacker output, while it is true that outputting a term containing a variable would be senseless, it is correct to output a term that contains a restricted name, even though the attacker may not be able to actually use

$$\begin{array}{c}
\frac{}{sub(\delta(t_1, \dots, t_n)) \triangleq sub(t_1) \cup \dots \cup sub(t_n)} \quad \frac{n \text{ is a name}}{sub(n) \triangleq n} \\
\\
\frac{\nexists t_i \text{ with a variable or a destructor}}{sub(f(t_1, \dots, t_n)) \triangleq f(t_1, \dots, t_n)} \quad \frac{x \text{ is a variable}}{sub(x) \triangleq \emptyset} \\
\\
\frac{\exists t_i \text{ with a variable or a destructor}}{sub(f(t_1, \dots, t_n)) \triangleq sub(t_1) \cup \dots \cup sub(t_n)}
\end{array}$$

**Fig. 3.** Relevant Subterms

$$\begin{array}{c}
\frac{P \vdash_k \varphi \quad Q \vdash_k \psi}{P + Q \vdash_k (\varphi \cup \psi)} \quad \frac{P \vdash_k \varphi \quad Q \vdash_k \psi}{P | Q \vdash_k (\varphi \cup \psi)} \quad \frac{P \vdash_k \varphi}{n(x).P \vdash_k \varphi} \\
\\
\frac{}{\mathbf{0} \vdash_k \emptyset} \quad \frac{P \vdash_k \varphi}{x(M).P \vdash_k \varphi \cup sub(M)} \quad \frac{P \vdash_k \varphi}{(vn)P \vdash_k \varphi \uparrow n} \\
\\
\frac{P\{n \leftarrow M\} \vdash_k \varphi}{\text{let } n = M \text{ in } P \vdash_k \varphi \cup sub(M)} \quad \frac{P \vdash_k \varphi}{[M = N].P \vdash_k \varphi \cup sub(M) \cup sub(N)}
\end{array}$$

**Fig. 4.** Relevant Term Extraction

the name in other messages. To take all this into account, we define a procedure  $sub$  that given a term, extracts its *relevant subterms* (not containing variables or destructors) and a procedure  $\uparrow$  that removes terms with restricted names.

**Definition 4 (Relevant Subterms).** Given a term  $M$  we define the set of its relevant subterms, written  $sub(M)$ , by the rules of Fig. 3.

**Definition 5 (Name Occurrence Term Removal).** We define the removal of terms from a set  $\psi$  in which the name  $x$  occurs,  $\psi \uparrow x$ , as:  $\psi \uparrow x \triangleq \{t \mid t \in \psi : x \notin \text{names}(t)\}$ .

**Definition 6 (Relevant Term Extraction).** Given a process  $P$ , the set  $\psi$  of relevant terms of  $P$ , written  $P \vdash_k \psi$ , is defined by the rules of Fig. 4.

For our attacker output we need to collect all closed terms that occur in the process, which we denote by  $ct(P)$ .

The semantics of our calculus are defined standardly, modulo  $\alpha$ -conversion of bound names and variables, by a structural congruence relation, labelled transition and reduction, as follows. We denote by  $fn(P)$  and  $fv(P)$  the set of free-names and free-variables of process  $P$ , respectively. We consider *closed* processes to be processes without free variables.

**Definition 7 (Structural Congruence).** Structural congruence  $\equiv$  is the least congruence relation on processes defined by the rules of Fig. 5.

We augment the standard structural congruence laws of the  $\pi$ -calculus with rules that equate processes modulo the equality  $=_E$  of the equational theory. These laws are essential in our semantics because they allow us to block processes performing actions that use terms that are not values (i.e. terms that contain destructors). This restriction is sensible since such a term represents a “stuck” computation and shouldn’t be emitted by a process (e.g  $\text{dec}(\text{enc}(m, k_1), k_2)$ ).

$$\begin{array}{ll}
n \notin \text{fn}(P) \cup \text{fv}(P) \Rightarrow P \mid (\nu n)Q \equiv (\nu n)(P \mid Q) & P \mid \mathbf{0} \equiv P \\
(\nu n)\mathbf{0} \equiv \mathbf{0} & P \mid Q \equiv Q \mid P \\
(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P & P \mid (Q \mid R) \equiv (P \mid Q) \mid R \\
M =_E M' \Rightarrow \text{let } x = M \text{ in } P \equiv \text{let } x = M' \text{ in } P & P + Q \equiv Q + P \\
M =_E M' \Rightarrow m\langle M \rangle.P \equiv m\langle M' \rangle.P & P + (Q + R) \equiv (P + Q) + R \\
M_1 =_E M'_1 \Rightarrow [M_1 = M_2].P \equiv [M'_1 = M_2].P & [M_1 = M_2].P \equiv [M_2 = M_1].P
\end{array}$$

**Fig. 5.** Structural Congruence

$$\begin{array}{c}
\frac{M \text{ is destructor-free}}{\text{let } x = M \text{ in } P \longrightarrow P\{x \leftarrow M\}} (\text{Let}) \quad \frac{M \text{ is destructor-free}}{n\langle M \rangle.P + R \mid n(x).Q + S \longrightarrow P \mid Q\{x \leftarrow M\}} (\text{Sync}) \\
\frac{M_1 \text{ and } M_2 \text{ are destructor-free} \quad M_1 =_E M_2}{[M_1 = M_2].P \longrightarrow P} (\text{Test}) \quad \frac{P \longrightarrow Q}{P \mid R \longrightarrow Q \mid R} (\text{Par}) \\
\frac{P \longrightarrow Q}{(\nu n)P \longrightarrow (\nu n)Q} (\text{Scope}) \quad \frac{P \equiv P' \quad P' \longrightarrow Q' \quad Q' \equiv Q}{P \longrightarrow Q} (\text{Cong}) \\
\frac{M \in \mathfrak{F}(ct(Q) \cup \bar{n}) \quad \bar{n} \text{ fresh}}{c(x).P + R \mid c(*).Q + S \longrightarrow (\nu \bar{n})(P\{x \leftarrow M\} \mid Q)} (\text{Attacker})
\end{array}$$

**Fig. 6.** Reduction Semantics

We now present our reduction semantics, with the aforementioned destructor freedom conditions. If a process is attempting to use a term that contains a destructor, we use structural congruence to rewrite the term destructor-free and reduction proceeds, or the term cannot be rewritten destructor-free and reduction halts. These conditions ensure that all received values are valid computations. Note the semantics of our attacker output, expressed in the *Attacker* rule, that enable the output to emit any message that can be generated by the process, given its closed terms and some fresh values, as would be expected.

**Definition 8 (Reduction Semantics).** *The reduction relation  $P \longrightarrow Q$  over closed processes is defined as the least relation closed under the rules of Fig. 6.*

We now present our labelled transition semantics.

**Definition 9 (Labelled Transition Semantics).** *The labelled transition relation  $P \xrightarrow{\alpha} Q$  is the least relation on closed processes closed under the rules of Fig. 7.*

This semantics is not intended to characterize a complete notion of behavioral equivalence as could be expected, but rather to allow the observation of actions in our logic. Despite not belonging to the scope of this work, we can point out that our labelled semantics do not allow for a complete characterization of behavioral equivalence, in the sense that our rules reveal information in a way that they induce a higher discriminative power than that of behavioral equivalence.

### 3 Logic

Having introduced the intended process model, we now introduce our logic. Considering it is common to reason about security by reasoning about the knowledge of principals, we explore key aspects of dynamic spatial logics, such as local reasoning, to

$$\begin{array}{c}
\frac{P \longrightarrow Q}{P \xrightarrow{\tau} Q} \text{ (Tau)} \quad \frac{M \text{ is destructor-free}}{n\langle M \rangle.P \xrightarrow{n(M)} P} \text{ (Out)} \\
\frac{M \text{ is destructor-free}}{n(x).P \xrightarrow{n(M)} P} \text{ (Inp)} \quad \frac{P \xrightarrow{\alpha} Q \quad \forall n \in \bar{u}: n \notin \text{names}(\alpha)}{(\nu \bar{u})P \xrightarrow{\alpha} (\nu \bar{u})Q} \text{ (Res)} \\
\frac{P \xrightarrow{n(M)} P' \quad \bar{s} \subseteq \text{names}(M) \text{ and } \bar{s} \subseteq \bar{u} \quad \bar{u}' = \bar{u} \setminus \bar{s}}{(\nu \bar{u})P \xrightarrow{v\bar{s}.n(M)} (\nu \bar{u}')P'} \text{ (BoundOut)} \\
\frac{P \equiv P' \quad P' \xrightarrow{\alpha} Q' \quad Q' \equiv Q}{P \xrightarrow{\alpha} Q} \text{ (Cong)} \quad \frac{M \in \mathfrak{F}(ct(P) \cup \bar{s}) \quad \bar{s} \text{ fresh}}{n\langle * \rangle.P \xrightarrow{v\bar{s}.n(M)} P} \text{ (AttackerOut)}
\end{array}$$

Fig. 7. Labelled Transition Semantics

$A, B ::= \mathbf{T}$ (True)	$P \models \mathbf{T} \triangleq \text{True}$
$\neg A$ (Negation)	$P \models \neg A \triangleq \text{not } P \models A$
$A \wedge B$ (Conjunction)	$P \models A \wedge B \triangleq P \models A \text{ and } P \models B$
$\mathbf{0}$ (Void)	$P \models \mathbf{0} \triangleq P \equiv \mathbf{0}$
$A \mid B$ (Composition)	$P \models A \mid B \triangleq \exists Q, R. P \equiv Q \mid R \text{ and } Q \models A \text{ and } R \models A$
$Hx.A$ (Hidden quantification)	$P \models Hx.A \triangleq \exists Q. P \equiv (\nu n)Q \text{ and } Q \models A\{x \leftarrow n\}$
$\alpha.A$ (Action)	$P \models \alpha.A \triangleq \exists Q. P \xrightarrow{\alpha} Q \text{ and } Q \models A$
$\square A$ (Always)	$P \models \square A \triangleq \forall Q. P \xrightarrow{\tau^*} Q \text{ and } Q \models A$
$\diamond A$ (Eventually)	$P \models \diamond A \triangleq \exists Q. P \xrightarrow{\tau^*} Q \text{ and } Q \models A$
$\mathbb{K}\phi$ (Knowledge)	$P \models \mathbb{K}\phi \triangleq P \vdash_k \psi \text{ and } \psi \models \phi$
$Sx.A$ (Secret quantification)	$P \models Sx.A \triangleq \exists Q, t. P \equiv (\nu k)Q \text{ and } Q \models A\{x \leftarrow t\}$ and $Q \vdash_k \phi$ such that $t \in \phi$ and $k \in \text{names}(t)$
$\phi, \psi ::= \varphi \wedge \psi$ (Conjunction)	
$t$ (Term)	
$\top$ (True)	

Fig. 8. Logic Syntax and Semantics

develop a novel logic that can reason about epistemic, dynamic and spatial properties of agents.

We propose an extension to a dynamic spatial logic [13] to enable reasoning at the term level. Our extension consists in adding two epistemic modalities:  $\mathbb{K}\phi$  denotes the ability of an agent to derive  $\phi$  from its knowledge, and  $Sx.A$  allows us to mention values that are only known by an agent (e.g. secrets). These modalities give the ability to reason about the information an agent can derive by applying functions and reductions to terms, coupling the ability to reason about properties of space and behavior to the ability to reason about derivable information modulo the equational theory.

### 3.1 Syntax and Semantics

The syntax and semantics of our logic are presented in Fig. 8. We refer to  $\phi, \psi$  as knowledge formulas and ambivalently use  $\phi, \psi$  to denote both knowledge formulas and finite sets of terms. The boolean connectives are standard.  $\mathbf{0}$  denotes the empty process;  $A \mid B$  denotes a process that can be partitioned in two components, one satisfying  $A$  and the

other satisfying  $B$ ;  $Hx.A$  allows us to mention restricted names of processes in formulas;  $\alpha.A$  denotes a process can perform action  $\alpha$  and continue as a process satisfying  $A$ ;  $\Box A$  and  $\Diamond A$  denote “always in the future” and “sometime in the future”, respectively. These are the standard connectives of dynamic spatial logic [13].  $\mathbb{K}\phi$  holds of a process that has the ability to derive the terms denoted by  $\phi$ , that is, the ability to know  $\phi$ ;  $Sx.A$  holds of a process that satisfies property  $A$  that depends on a value that is secret to a process, that is, a term containing a restricted name.

With this logic, we can state properties about the knowledge of principals (subsystems) over time, such as “it is never the case that the secret key is known by 3 subsystems”:  $\neg\Diamond H \text{key}.\mathbb{K} \text{key} \mid \mathbb{K} \text{key} \mid \mathbb{K} \text{key}$ ; or “it is always the case that 2 agents know the key and one does not”:  $\Box H \text{key}.\mathbb{K} \text{key} \mid \mathbb{K} \text{key} \mid \neg\mathbb{K} \text{key}$ . Notice how the expressiveness of the logic arises from the ability to combine the three types of modalities: dynamic ( $\Box, \Diamond$ ), spatial ( $H, \mid$ ) and epistemic ( $\mathbb{K}$ ). The dynamic connectives allow us to range over a specific execution or all possible executions, the spatial connectives allow us to mention restricted names (usually used to model keys and nonces) and to refer subsystems, and the epistemic connectives allow us to analyze the knowledge of a process.

However, the semantics for  $\mathbb{K}\phi$  pose a challenge in the sense that they use the notion of knowledge derivation from Section 2.1. While this definition is adequate from a semantic perspective, it makes use of the DY equational closure of a set which is not stable by reduction of terms, and therefore doesn’t provide a clear way of algorithmically determining if  $\psi \models \phi$ . We approach the problem with a purely logical approach and characterize knowledge derivation with a structural proof system for knowledge formulas, unlike the approach of [12]. Our proof system for knowledge derivation and its properties are described in the following section.

### 3.2 Proof System for Knowledge Formulas

Our proof system, formulated as a sequent calculus, is equipped with rules from the equational theory in order to consider the ability to combine terms to generate new information. Each rule of our calculus represents a possible computational step that an agent can perform on terms to produce a new term. Intuitively, if a sequent  $\Gamma \vdash A$  is derivable, this means that  $A$  is deducible from the knowledge represented by  $\Gamma$ .

**Definition 10 (Proof System K for Knowledge Formulas).** *The sequent calculus formulation of our proof system K for knowledge formulas is defined by the rules of Fig. 9.*

The rules for identity, conjunction elimination and introduction are standard. Rule *fun-Right* expresses the ability to derive a constructor application if it is possible to derive the several necessary subterms, since any agent can apply functions to terms; rule *AtLeft* expresses that all that can be derived from a term  $f(t_1, \dots, t_n)$  can also be derived from terms  $t_1, \dots, t_n$ , for the same reason as in the previous rule; rule *DestrLeft* reflects the equalities of the equational theory: what can be deduced from  $s$  can also be deduced from  $f(t_1, \dots, t_n)$  since both terms are equal under the equational theory.

For the sequent calculus  $K$ , we establish the results of *soundness*, *completeness* and *decidability*.

$$\frac{}{\Gamma, A \vdash A} (\text{Id}) \quad \frac{\Gamma, A, B \vdash C}{\Gamma, A \wedge B \vdash C} (\wedge: \text{left}) \quad \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} (\wedge: \text{right})$$

For every constructor function symbol  $f$  with arity  $n$ , such that  $f \in \Sigma$ :

$$\frac{\Gamma \vdash t_1 \dots \Gamma \vdash t_n}{\Gamma \vdash f(t_1, \dots, t_n)} (\text{funRight}) \quad \frac{\Gamma, f(t_1, \dots, t_n) \vdash C}{\Gamma, t_1, \dots, t_n \vdash C} (\text{AttLeft})$$

For every equation  $f(t_1, \dots, t_n) = s \in E$ :

$$\frac{\Gamma, s \vdash C}{\Gamma, f(t_1, \dots, t_n) \vdash C} (\text{DestrLeft})$$

**Fig. 9.** Proof System for Knowledge Formulas

**Theorem 1 (Soundness of K).** *Given a set of terms  $S$ , if  $S \vdash A$  then  $S \models A$ .*

*Proof:* By induction on the derivation of  $S \vdash A$  ■

**Theorem 2 (Completeness of K).** *Given a set of terms  $S$ , if  $S \models A$  then  $S \vdash A$ .*

**Theorem 3 (Decidability of K).** *For any set of terms  $S$  and term  $A$ ,  $S \vdash A$  is decidable.*

The proofs of completeness and decidability rely on a finite approximation result for the DY equational closure of a set of terms. More concretely, for each finite set of terms  $S$  and equational theory, it is possible to build a finite set  $b(S)$  from which all terms in the DY equational closure of  $S$  may be determined.

**Proposition 1 (Approximation of  $\mathfrak{F}(S)$ ).** *Let  $S$  be a finite set of terms. We may construct in polynomial time an approximation to  $\mathfrak{F}(S)$ , named  $b(S)$ , a finite set with the following property:*

$$\forall M \in \mathfrak{F}(S), \exists C, \bar{t} \in b(S) \text{ such that } M = C[\bar{t}]$$

where  $C[-]$  is a functional context solely built out of constructors.

*Proof:* The finite approximation  $b(S)$  is built from the terms of  $S$  by interpreting the rewrite rules of the theory as contexts of a bounded size. Therefore, applying function symbols to terms of  $S$  up to the bound of the context produces a new term by then applying the rewrite rule. This procedure is iterated, eventually reaching a fix-point, due to the subterm convergency property of the equational theories (the idea is that each time we produce a new term, the term will be smaller than the terms used to generate it). The resulting computable set has the property that defines our approximation [14]. ■

The approximation  $b(S)$  is such that all terms of  $\mathfrak{F}(S)$  can be built from terms of  $b(S)$  just by applying constructors, no longer requiring the equations from the theory. Completeness follows from the fact that our proof system is able to emulate the computation steps required to generate the approximation. Given a set of terms  $S$ ,  $b(S)$  is generated by applying functions to terms of  $S$ , applying a rewrite rule to the resulting term and iterating. Thus, our proof system is complete since the computation steps of  $b(S)$  may be emulated by the rules of proof system K, and we may then apply function symbols to terms of  $b(S)$  to produce terms of  $\mathfrak{F}(S)$ . The latter is trivial due to *funRight* and *AttLeft*. The former we prove through the following lemma.

**Lemma 1. (Completeness of  $\mathbb{K}$  i.r.t the Approximation).** Given a set of terms  $S$ , if  $t \in b(S)$  then  $S \vdash t$ .

*Proof:* Through instances of `AttLeft` it is possible to apply functions to terms of  $S$  up to the bound of the context used in  $b(S)$ . Through an instance of `DestrLeft` the corresponding rewrite rule can be applied, and through `Id` the new term is derived at the root of the proof tree [14]. ■

To emulate the iteration with the proof system, that is, to perform similar computations with  $S$  and the new term, the auxiliary result of reasoning with cuts is required.

**Lemma 2. (Cut Admissibility in  $\mathbb{K}$ ).** If  $\Gamma \vdash A$  and  $\Gamma, A \vdash C$  then  $\Gamma \vdash C$ .

*Proof:* See [14].

Using Cut, the proof system is able to emulate the iterative procedure by building the previously described proof tree that allows the derivation of a new term, and using the new term as the cut formula. This technique can then be applied to produce any term of  $b(S)$ , as required. Thus Lemma 1 holds and therefore Theorem 2 also holds. Since the computation of  $b(S)$  always terminates, Theorem 3 holds.

### 3.3 Model-Checking

We now discuss model-checking for our logic. We know that model-checking is decidable for the logic without epistemic modalities [13], for the class of bounded processes. Therefore, we need only show that our two modalities preserve the decidability result.

**Proposition 2 (Decidability of model-checking  $\mathbb{K}$ ).** Let  $\phi$  be a finite set of terms. Checking that  $P \models \mathbb{K}\phi$  is decidable.

The above proposition holds since for any process  $P$  it is possible to collect its set of relevant terms  $\psi$  ( $P \vdash_k \psi$ ), compute the finite approximation  $b(\psi)$  and check that each term in  $\phi$  can be constructed from terms of  $b(\psi)$  by application of constructors.

**Proposition 3 (Decidability of model-checking  $Sx.A$ ).** Checking that  $P \models Sx.A$  is decidable.

Decidability of  $Sx.A$  follows from the fact that if  $P \equiv (vn)Q$ , it is possible to collect the set  $\psi$  of relevant terms of process  $Q$ , pick some term  $t$  from  $\psi$  that contains the name  $n$  and check that  $Q \models A\{x \leftarrow t\}$ . Given that model-checking the core dynamic spatial logic with  $\mathbb{K}$  is decidable, it follows that checking  $P \models Sx.A$  is decidable and therefore model-checking for our logic is decidable for the class of bounded processes. We may then state and prove:

**Theorem 4 (Decidability of Model-Checking).** Checking that  $P \models A$  is decidable for the class of bounded processes.

## 4 Expressiveness and Extensions

Having presented our framework, we discuss some extensions to our work that provide additional features that can be used to model and analyze systems. In particular, we discuss the representation of attackers, and modeling and verification of correspondence assertions [15] in our framework.

$$\begin{aligned}
A(K) &\triangleq (\nu K_{ab}, N) c(\text{enc}(\text{pair}(K_{ab}, N), K)).c(x).[N - 1 = \text{dec}(x, K_{ab})] \\
B(K) &\triangleq c(x).\text{let } K_{ab} = \text{fst}(\text{dec}(x, K)) \text{ } N = \text{snd}(\text{dec}(x, K)) \text{ in } c(\text{enc}(N - 1, K_{ab})) \\
Sys &\triangleq (\nu K) (A(K) | B(K))
\end{aligned}$$

**Fig. 10.** Modeling the Example

$$\begin{aligned}
Attacker &\triangleq c(x).c(*).c(y).c(*).\text{mem}(x, y) \\
World &\triangleq (Sys | World) \\
World &\models \neg \diamond \text{Hk}.(2 | (@\text{mem} \wedge \mathbb{K}k))
\end{aligned}$$

**Fig. 11.** An Attacker for the Example

#### 4.1 Modeling Attackers

To analyze a security protocol one usually needs to consider how it behaves in any possible environment. While our logic focus on the analysis of closed systems, it is still possible to check properties of a system's behavior and knowledge in an arbitrary environment (representing an arbitrary attacker), by using a special technique to internalize an arbitrary attacker inside the system. The general idea is that, for any process  $P$ , we may determine a process  $Q$  (making essential use of the attacker prefix construct) such that  $P|Q$  reaches some state whenever  $P$  reaches an equivalent state when placed in an arbitrary environment. While in general the explicit specification of an attacker for a given protocol may not be easy, since it requires some *a priori* guessing of the kind of attack one is looking for, our approach to represent the attacking environment is quite different and general, and may indeed be used to find unanticipated attacks (see example in Section 4.2). In fact, we can generically model a Dolev-Yao [2] attacker in our framework by considering the number of message exchanges and the communication channels used in a protocol.

Considering an arbitrary protocol modeled as a process, the role of an attacker is to intercept all communications of the principals and be able to inject any message it can produce, given its knowledge at the time, at any point where a principal expects to receive a message (cf. our attacker output). Thus, a *general attacker* consists of a process that for all outputs of the protocol performs an input (storing the received message) and for all inputs performs an attacker output. For instance, consider the following protocol, where  $K$  is a shared key,  $N$  a fresh value and  $K_{ab}$  a session key generated by  $A$ :

$$\begin{aligned}
A &\rightarrow B : \{K_{ab}, N\}_K \\
B &\rightarrow A : \{N - 1\}_{K_{ab}}
\end{aligned}$$

In our process model, such a protocol would be represented as done in Fig. 10 (we omit the signature and equational theory for the sake of brevity). An attacker for this protocol, following our *attacker schema* is presented in Fig. 11 (*mem* can be any unused channel), where we state that it is never the case that the attacker can know one of the keys used in the protocol. This approach shows that while the additional effort of representing an attacker is necessary, we can easily represent a generic attacker for a protocol by following a pre-determined schema. We consider finite protocols, modeled as processes in our calculus that use a communication channel  $c$  as their communication medium (written  $P_c$ ). Our attacker schema is as follows: For each output on  $c$ , the at-

```

proc Attacker( $P, S$ )  $\equiv$ 
  if  $P \xrightarrow{\alpha} Q \wedge \alpha = \text{input on } c$  then  $c(*)$ .Attacker( $Q, S$ ) fi
  if  $P \xrightarrow{\alpha} Q \wedge \alpha = \text{output on } c$  then  $c(x)$ .Attacker( $Q, S \cup x$ ) fi
  if  $P \xrightarrow{\alpha}$  then  $m\langle x_1, \dots, x_n \rangle$  where  $x_i \in S$  fi.

```

**Fig. 12.** Attacker Generation

tacker performs an input on  $c$  (and stores the message). For each input on  $c$ , the attacker performs an attacker output.

**Definition 11 (Attacker Generation Procedure).** *Given a process  $P_c$  that models a finite protocol, the set  $S$  that tracks the attacker memory, an attacker for  $P$  can be generated by procedure  $\text{Attacker}(P, S)$  defined in Fig. 12 ( $x$  and  $m$  are fresh in  $P$  and the attacker).*

The generation procedure produces the necessary actions by inspection of the process dynamics: if an output can occur in the process, the attacker intercepts the message and memorizes it; if an input can occur in the process, the attacker injects any message it can produce from its knowledge; in the case where the protocol has no more actions, we represent the attacker memory with an output  $m\langle x_1, \dots, x_n \rangle$ , modeling the attacker's memory throughout the protocol run. We thus show how an attacker can be extracted by inspection of the considered protocol. We now show that this attacker is general in our framework, in the sense that it can simulate the behavior expected from an adversarial medium (c.f. Dolev-Yao attacker). Note that this result does not yet fully apply to our tool implementation, as we discuss later in this section.

**Definition 12 (K-Set).** *Given a process  $P$ , we define its K-Set  $K_P$ , the set of all terms known by  $P$  as:  $K_P \triangleq \{t \mid P \models \mathbb{K} t\}$*

**Lemma 3 (Monotonicity of  $\mathfrak{F}$  in Sub).** *Let  $M$  be a term,  $N$  a closed term without destructors and  $x$  a variable of  $M$ :  $\mathfrak{F}(\text{sub}(M\{x \leftarrow N\})) \subseteq \mathfrak{F}(\text{sub}(M) \cup N)$ .*

**Lemma 4 (Monotonicity of K-Sets Under Substitution).** *For any process  $P_c$ ,  $K_{P\{x \leftarrow M\}} \subseteq \mathfrak{F}(K_P \cup M)$  where  $x$  is a variable in  $P_c$  and  $M$  is a closed term without destructors.*

**Theorem 5 (Monotonicity of K-Sets under Synchronization).** *Let  $P_c$  and  $A$  be a processes such that  $P_c \xrightarrow{c(M)} P'_c$  and  $A \xrightarrow{c(x)} A'$ . We have that  $K_{A'} \subseteq \mathfrak{F}(K_A \cup M)$ .*

We begin with the K-Set of a process, the set of all terms known by the process that we observe in our logic, and we show that the evolution of arbitrary processes' K-Sets through synchronization is monotonic, in the sense that the resulting process' knowledge will be a subset of the initial process' knowledge, plus any received messages. We state a similar property of our generated attacker's K-Set. Over time, the attacker's K-Set captures all messages exchanged in the protocol.

**Theorem 6 (Monotonicity of Attacker Storage).** *Let  $P_c$  and  $At$  be processes such that  $At = \text{Attacker}(P_c, \{\}, c)$ ,  $P \xrightarrow{c(M)} P'$  and  $At \xrightarrow{c(x)} At'$ . We have that  $K_{At'} = \mathfrak{F}(K_{At} \cup M)$ .*

Our Attacker Simulation (Lemma 5) and Process Knowledge (Lemma 6) lemmas provide some insight on the expressiveness of our generated attacker. Lemma 5 shows that a generated attacker, through interactions with an arbitrary finite protocol, can obtain as much knowledge as an arbitrary process in the same circumstances. Lemma 6 states a similar property, regarding the knowledge a finite protocol may obtain while interacting with our attacker.

**Lemma 5 (Attacker Simulation).** *Let  $P_c$  and  $A$  be processes.*

*If  $(\nu\bar{n})(P_c \mid A) \xrightarrow{*} (\nu\bar{n})(P'_c \mid A')$  and  $At = \text{Attacker}(P_c, S)$  with  $K_A \subseteq K_{At}$  then  $\exists At', S'$  such that  $(\nu\bar{n})(P_c \mid A) \xrightarrow{*} (\nu\bar{n})(P'_c \mid At')$  and  $At' = \text{Attacker}(P'_c, S \cup S')$  and  $K_{At'} \subseteq K_{At}$ .*

**Lemma 6 (Process Knowledge).** *Let  $P_c, A$  be processes and  $\phi$  a knowledge formula.*

*If  $(\nu\bar{n})(P_c \mid A) \xrightarrow{*} (\nu\bar{n})(P'_c \mid A')$  and  $P'_c \models \mathbb{K}\phi$  and  $At = \text{Attacker}(P_c, S)$  with  $K_A \subseteq K_{At}$  then  $\exists At', S'$  such that  $(\nu\bar{n})(P_c \mid A) \xrightarrow{*} (\nu\bar{n})(P'_c \mid At')$  and  $P'_c \models \mathbb{K}\phi$  and  $At' = \text{Attacker}(P'_c, S \cup S')$ .*

Furthermore, from Lemma 6 follows that, in our logic, a finite protocol interacting with an arbitrary process is indistinguishable from one interacting with our attacker. From Lemma 6 we know that the knowledge resulting from interactions with an arbitrary process and our attacker is the same, therefore the result follows by induction on the structure of the formula  $A$ . By combining these results, we show that our attacker can behave as one would expect of an adversarial Dolev-Yao agent, given that it has the ability of collecting all the available information of a protocol and emitting any possible values generated from those obtained.

**Theorem 7 (Preservation of Satisfaction).** *Let  $P_c$  and  $A$  be processes and  $A$  any formula. If  $(\nu\bar{n})(P_c \mid A) \xrightarrow{*} (\nu\bar{n})(P'_c \mid A')$  and  $P'_c \models A$  and  $At = \text{Attacker}(P_c, S)$  with  $K_A \subseteq K_{At}$  then  $\exists At', S'$  such that  $(\nu\bar{n})(P_c \mid A) \xrightarrow{*} (\nu\bar{n})(P'_c \mid At')$  and  $P'_c \models A$  and  $At' = \text{Attacker}(P'_c, S \cup S')$ .*

Notice that this general result follows from the fact that messages generated by the attacker output prefix are unbounded. Our implementation bounds the generated message, to ensure tractability, and thus sacrificing completeness. We believe however that for any possible finite protocol a bound on the size of the generated messages may be effectively computed; this is a subject of ongoing work, building on symbolic interpretation techniques (cf. [16]). It is anyway important to note that our method is already sound and complete for passive attackers, even for the case of non finite processes (eg. we may consider any finite control system, or bounded in the sense of [13]).

## 4.2 Modeling Correspondence Assertions

Correspondence assertions are a technique for verifying authentication properties in protocols [15, 17]. The idea is that the model of each principal in a protocol is refined with begin/end events, named *correspondence assertions*, at each stage of an authentication procedure. Authentication will be established if, for every run of the protocol, all end events for each stage are preceded by a matching begin event.

$$A \rightarrow B : \{N\}_k$$

*B* asserts the reception of *N*

$$B \rightarrow A : \{h(N)\}_k$$

*A* asserts the reception of *h(N)*

**Fig. 13.** Protocol with Correspondence Assertions

```
parameter attacker_depth = 2;

defproc A(k) = new N in c!(enc(N,k)).c?(x).[dec(x,k)=h(N)].end!(h(N));
defproc B(k) = c?(x).(begin!(dec(x,k)) | c!(enc(h(dec(x,k)),k)));
defproc Sys = new k in (A(k) | B(k));
defproc Attacker = c?(v).c!(*).s!(v);
defproc World = (Sys | Attacker);

defprop begin = <begin!> true;
defprop end = <end!> true;
defprop corrsp = always (end => begin);
check World |= corrsp;
Processing...
* Process World satisfies the formula corrsp *
```

**Fig. 14.** Checking Correspondence in a Toy Protocol

To illustrate the idea, consider the example of Fig. 13, where *A* and *B* share a symmetric key *k*, *N* is a fresh value and *h* is a one-way hash function. when *B* receives  $\{N\}_k$  it asserts the beginning of the run of the protocol. *B* sends message  $\{h(N)\}_k$  so that *A* can verify the freshness of the run, by comparing the received value with its own hash of *N*. If the test succeeds, *A* asserts the reception of *h(N)* and the end of the run. To check correspondence, one has to check that every run of the protocol, in the presence of an adversary, would be such that *A*'s end assertion is always preceded by *B*'s begin assertion, that is, *A* only ends if *B* was involved in the protocol.

Using our framework, we can model correspondence assertions by representing the assertion as an output on a channel that is irrelevant to the protocol, and then inspecting the existence of such outputs with our logic. For instance, our example could be modeled as done in Fig. 14 (note the `attacker_depth` parameter set to 2 due to the size of the second message). We can also successfully handle the case where we consider a faulty system that leaks *k* to the attacker (and thus correspondence does not hold), as presented in Fig. 15. This shows that our framework has enough expressivity to model other verification techniques, such as correspondence assertions.

## 5 Concluding Remarks and Related Work

In this paper we have introduced a dynamic spatial epistemic logic for a variant of the applied  $\pi$ -calculus aimed at reasoning about security protocols. We explore the application of combining spatial and epistemic reasoning to the several agents involved in a security protocol, be they principals or adversaries. In our work, we can reason about the knowledge of the several agents of a protocol and how it can evolve over time. Model-checking for the logic is shown to be decidable for an interesting class of processes and cryptographic primitives.

Our framework allows an interesting degree of freedom in the analyses it can perform, not only allowing one to reason directly about knowledge but also enabling rea-

```

...
defproc Sys = new k in (c!(k).(A(k) | B(k)));
defproc Attacker = c?(u).c?(v).c!(*).s!(v,u);
defproc World = (Sys | Attacker);
...
check World |= corrsp;
Processing...
* Process World does not satisfy the formula corrsp *

```

**Fig. 15.** Checking Correspondence in a Broken Toy Protocol

soning with correspondence assertions, which is an important addition to the range of available techniques. Moreover, our explicit attacker representation, which does not require a complete behavioral specification, is able to accurately emulate the behavior of a Dolev-Yao attacker, enabling reasoning about the dynamics and knowledge of such an attacker.

Finally, the decidability result for our logic allowed us to implement a model-checking algorithm as an extension to the SLMC tool. The main difference between the tool and the theory is that our attacker outputs are parametrized with a maximum message size, to bound the state space. This is the main limitation of the current version of our tool, since it does not yet fully capture the expressiveness of our attacker modeling, given that our results employ a more powerful version of the attacker output.

Overall, we have produced an interesting formal framework and tool for protocol analysis, the first that employs dynamic spatial logics, enabling a very natural (yet precise) way of reasoning about security protocols, all the while allowing reasoning with previously established techniques.

While Kremer et al. [10] have proposed an epistemic logic for the applied  $\pi$ -calculus, their logic lacks the ability to reason about spatial properties, which is the key element that allows reasoning about principals. Furthermore, their epistemic modalities are interpreted over maximal traces of a protocol run that are indistinguishable through static equivalence. This is a substantially different approach in comparison with ours, given that we have a clear separation of epistemic and dynamic modalities, not relying on static equivalence. Mardare and Priami have also proposed a dynamic epistemic spatial logic [18], for CCS processes and without the problems of security in mind. Their logic is hence substantially different from ours, mainly due to their epistemic modality interpreting knowledge as the possibility of observing actions of other processes, resulting in an operator similar to the adjunct of the parallel operator of spatial logic.

For future work we wish to further study the problem of attacker representations, aiming at an identical expressiveness result that does not require the attacker to be able to produce a message of an arbitrary size. This result will be key in removing the previously discussed limitation of our tool.

*Acknowledgments* We would like to thank Mario Pires and Pedro Adão for their interesting comments on some version of this work.

## References

1. Diffie, W., Hellman, M.E.: New directions in cryptography (1976)
2. Dolev, D., Yao, A.C.: On the security of public key protocols. Technical report, Stanford, CA, USA (1981)

3. Denning, D.E., Sacco, G.M.: Timestamps in key distribution protocols. *Commun. ACM* **24**(8) (1981) 533–536
4. Lowe, G.: Breaking and fixing the needham-schroeder public-key protocol using *fd*. In: TACAs '96: Proceedings of the Second International Workshop on Tools and Algorithms for Construction and Analysis of Systems, London, UK, Springer-Verlag (1996) 147–166
5. Abadi, M., Gordon, A.D.: A calculus for cryptographic protocols: the spi calculus. In: CCS '97: Proceedings of the 4th ACM conference on Computer and communications security, New York, NY, USA, ACM (1997) 36–47
6. Burrows, M., Abadi, M., Needham, R.: A logic of authentication. *ACM Trans. Comput. Syst.* **8**(1) (1990) 18–36
7. Bodei, C., Degano, P., Nielson, F., Nielson, H.R.: Flow logic for dolev-yao secrecy in cryptographic processes (2002)
8. Abadi, M., Fournet, C.: Mobile values, new names, and secure communication. In: POPL '01: Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, New York, NY, USA, ACM (2001) 104–115
9. Étienne Lozes, Villard, J.: A spatial equational logic for the applied  $\Pi$ -calculus. In: CONCUR '08: Proceedings of the 19th international conference on Concurrency Theory, Berlin, Heidelberg, Springer-Verlag (2008) 387–401
10. Chadha, R., Delaune, S., Kremer, S.: Epistemic logic for the applied pi calculus. In Lee, D., Lopes, A., Poetsch-Heffter, A., eds.: Proceedings of IFIP International Conference on Formal Techniques for Distributed Systems (FMOODS/FORTE'09). Lecture Notes in Computer Science, Lisbon, Portugal, Springer (June 2009)
11. Toninho, B., Caires, L.: A spatial-epistemic logic and tool for reasoning about security protocols. Technical report, Departamento de Informática, FCT/UNL (2009)
12. Abadi, M., Cortier, V.: Deciding knowledge in security protocols under equational theories. *Theor. Comput. Sci.* **367**(1) (2006) 2–32
13. Caires, L.: Behavioral and spatial observations in a logic for the pi-calculus. *FoSSaCS 2004* (2004)
14. Toninho, B.: A logic and tool for local reasoning about security protocols. Master's thesis, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa (2009)
15. Woo, T.Y.C., Lam, S.S.: A semantic model for authentication protocols. In: SP '93: Proceedings of the 1993 IEEE Symposium on Security and Privacy, Washington, DC, USA, IEEE Computer Society (1993) 178
16. Cortier, V., Delaune, S.: A method for proving observational equivalence. In: CSF '09: Proceedings of the 2009 22nd IEEE Computer Security Foundations Symposium, Washington, DC, USA, IEEE Computer Society (2009) 266–276
17. Gordon, A.D., Jeffrey, A.: Typing correspondence assertions for communication protocols. *Theor. Comput. Sci.* **300**(1-3) (2003) 379–409
18. Mardare, R., Priami, C.: Dynamic epistemic spatial logic. Technical report, Center for Computational and Systems Biology, University of Trento (2006)

## A Proofs

**Proof of Lemma 2 (Cut Elimination)** If a sequent  $\Gamma \vdash A$  has a single cut derivation in  $K$  with cut, then it has a cut-free derivation.

*Proof:* Induction on the structure of the derivation of  $\Gamma \vdash A$ . By the assumptions of the lemma we have that the derivation is of the form

$$\frac{\frac{\Pi(n)}{\Gamma \vdash A} \quad \frac{\Pi(m)}{\Gamma, A \vdash C}}{\Gamma \vdash C} \text{ (Cut)}$$

where  $\Pi(n)$  and  $\Pi(m)$  are cut-free derivations for  $\Gamma \vdash A$  and  $\Gamma, A \vdash C$ , respectively. The various forms of the premise derivations  $\Pi(n)$  and  $\Pi(m)$  fall under the following cases:

1. One of the premises is an instance of (Id).
2. One of the premises is an instance of a rule that doesn't introduce the cut formula.
3. Both premises introduce the cut formula

We now discuss the identified cases:

1. **Case (Cut)-(Id):** Suppose (Id) occurs on the right premise of the cut, the derivation must have the form

$$\frac{\frac{\Pi(n)}{\Gamma \vdash A} \quad \frac{}{\Gamma, A \vdash A} \text{ (Id)}}{\Gamma \vdash A} \text{ (Cut)}$$

and we can remove the cut since  $\Gamma \vdash A$  follows from  $\Pi(n)$ , which is cut free. The case where (Id) appears on the left premise of the cut is handled symmetrically.

2. **Case (Cut)-(LL):** The cases where the left premise of the cut is a left rule that doesn't introduce the cut formula.

- (a) Case (Cut)-( $\wedge$ L)

$$\frac{\frac{\frac{\Pi(n)}{\Gamma, A, B \vdash C}}{\Gamma, A \wedge B \vdash C} \text{ ( $\wedge$ L)} \quad \frac{\Pi(m)}{\Gamma, A \wedge B, C \vdash D}}{\Gamma, A \wedge B \vdash D} \text{ (Cut)}$$

Applying weakening on  $\Pi(n)$  and  $\Pi(m)$ , we construct  $\Pi'(n)$  and  $\Pi'(m)$ :

$$\frac{\frac{\Pi'(n)}{\Gamma, A, B, A \wedge B \vdash C} \quad \frac{\Pi'(m)}{\Gamma, A, B, A \wedge B, C \vdash D}}{\Gamma, A, B, A \wedge B \vdash D}$$

By the induction hypothesis we have a cut-free derivation for  $\Gamma, A, B, A \wedge B \vdash D$  and:

$$\frac{\vdots}{\frac{\Gamma, A, B, A \wedge B \vdash D}{\Gamma, A \wedge B \vdash D} \text{ ( $\wedge$ L)}}$$

since our sequents have formula sets (and not multi-sets) on the left-hand side, we have a derivation for  $\Gamma, A \wedge B \vdash D$  cut-free.

- (b) Case (Cut)-(DestrLeft)

$$\frac{\frac{\frac{\Pi(n)}{\Gamma, s \vdash C}}{\Gamma, f(\bar{i}) \vdash C} \text{ (DestrLeft)} \quad \frac{\Pi(m)}{\Gamma, f(\bar{i}), C \vdash D}}{\Gamma, f(\bar{i}) \vdash D} \text{ (Cut)}$$

Applying weakening on  $\Pi(n)$  and  $\Pi(m)$ , we construct  $\Pi'(n)$  and  $\Pi'(m)$ :

$$\frac{\frac{\Pi'(n)}{\Gamma, s, f(\bar{i}) \vdash C} \quad \frac{\Pi'(m)}{\Gamma, f(\bar{i}), s, C \vdash D}}{\Gamma, s, f(\bar{i}) \vdash D}$$

By the induction hypothesis we have a cut-free derivation for  $\Gamma, s, f(\bar{i}) \vdash D$  and:

$$\frac{\vdots}{\frac{\Gamma, s, f(\bar{i}) \vdash D}{\Gamma, f(\bar{i}) \vdash D} \text{ (DestrLeft)}}$$

since our sequents have formula sets (and not multi-sets) on the left-hand side, we have a derivation for  $\Gamma, f(\bar{i}) \vdash D$  cut-free.

(c) Case (Cut)-(AttLeft)

$$\frac{\frac{\frac{\Pi(n)}{\Gamma, f(\bar{i}) \vdash C} \quad \frac{\Pi(m)}{\Gamma, \bar{i}, C \vdash D}}{\Gamma, \bar{i} \vdash C} \text{ (AttLeft)} \quad \frac{\Pi(m)}{\Gamma, \bar{i}, C \vdash D} \text{ (Cut)}}{\Gamma, \bar{i} \vdash D}$$

Applying weakening on  $\Pi(n)$  and  $\Pi(m)$ , we construct  $\Pi'(n)$  and  $\Pi'(m)$ :

$$\frac{\frac{\Pi'(n)}{\Gamma, f(\bar{i}), \bar{i} \vdash C} \quad \frac{\Pi'(m)}{\Gamma, f(\bar{i}), \bar{i}, C \vdash D}}{\Gamma, f(\bar{i}), \bar{i} \vdash D}$$

By the induction hypothesis we have a cut-free derivation for  $\Gamma, f(\bar{i}), \bar{i} \vdash D$  and:

$$\frac{\vdots}{\frac{\Gamma, f(\bar{i}), \bar{i} \vdash D}{\Gamma, f(\bar{i}) \vdash D} \text{ (AttLeft)}}$$

since our sequents have formula sets (and not multi-sets) on the left-hand side, we have a derivation for  $\Gamma, f(\bar{i}) \vdash D$  cut-free.

(d) Case (Cut)-(RL): The cases where the right premise of the cut is a left rule that doesn't introduce the cut formula are handled symmetrically to Case 2.

3. Cases where the cut premisses introduce the cut formula. The last rule on the left premise is a right rule and the last rule on the right premise is a left rule.

(a) Case of  $\wedge$ :

$$\frac{\frac{\frac{\Pi(n)}{\Gamma \vdash A} \quad \frac{\Pi(m)}{\Gamma \vdash B}}{\Gamma \vdash A \wedge B} \text{ (\wedge R)} \quad \frac{\frac{\Pi(k)}{\Gamma, A, B \vdash C}}{\Gamma, A \wedge B \vdash C} \text{ (\wedge L)}}{\Gamma \vdash C} \text{ (Cut)}$$

From weakening we have:

$$\frac{\Pi'(n)}{\Gamma, B \vdash A} \quad \frac{\Pi'(m)}{\Gamma, A \vdash B}$$

We can thus build the derivations:

$$\frac{\frac{\frac{\Pi'(n)}{\Gamma, B \vdash A} \quad \frac{\Pi(k)}{\Gamma, A, B \vdash C}}{\Gamma, B \vdash C} \text{ (Cut)} \quad \frac{\frac{\Pi'(m)}{\Gamma, A \vdash B} \quad \frac{\Pi(k)}{\Gamma, A, B \vdash C}}{\Gamma, A \vdash C} \text{ (Cut)}}$$

and by the induction hypothesis we have cut-free derivations for  $\Gamma, B \vdash C$  and  $\Gamma, A \vdash C$ . We then use such cut-free derivations to construct:

$$\frac{\frac{\Pi(n)}{\Gamma \vdash A} \quad \vdots}{\Gamma \vdash C} \quad \text{or} \quad \frac{\frac{\Pi(m)}{\Gamma \vdash B} \quad \vdots}{\Gamma \vdash C}$$

And by the induction hypothesis, there exist cut-free derivations for both sequents.

(b) Term formulae (funRight + DestrLeft)

$$\frac{\frac{\frac{\Pi(m_1)}{\Gamma \vdash t_1} \quad \dots \quad \frac{\Pi(m_n)}{\Gamma \vdash t_n}}{\Gamma \vdash f(t_1, \dots, t_n)} \quad \frac{\frac{\Pi(k)}{\Gamma, t \vdash C}}{\Gamma, f(t_1, \dots, t_n) \vdash C}}{\Gamma \vdash C} \text{ (FunRight)} \quad \text{(AttLeft)} \quad \text{(Cut)}$$

From the  $\Pi(m_i)$  derivations we can build:

$$\frac{\frac{\frac{\Pi(m_1)}{\Gamma \vdash t_1} \quad \dots \quad \frac{\Pi(m_n)}{\Gamma \vdash t_n}}{\Gamma \vdash \bar{t}} \quad \frac{\frac{\Gamma, t \vdash t}{\Gamma, f(\bar{t}) \vdash t}}{\Gamma, \bar{t} \vdash t}}{\Gamma \vdash t} \text{ (Cut)}$$

From the induction hypothesis we can derive  $\Gamma \vdash t$  cut-free. Finally:

$$\frac{\vdots \quad \frac{\Pi(k)}{\Gamma, t \vdash C}}{\Gamma \vdash C} \text{ (Cut)}$$

And by application of the induction hypothesis there exists a cut-free derivation for  $\Gamma \vdash C$ . ■

**Proof of Theorem 1 (Soundness of K)** If  $\Gamma \vdash A$  is derivable in the proof system K, then  $\Gamma \models A$ .

*Proof:* Induction on the derivation of  $\varphi \vdash \psi$

We assume  $\varphi$  to be of the form  $v_1, \dots, v_k$ .

1. Cases (Id) and ( $\wedge$ : left)  
Trivial from the I.H.
2. Case ( $\wedge$ : right)

$$\frac{\varphi \vdash A \quad \varphi \vdash B}{\varphi \vdash A \wedge B}$$

From the I.H follows that  $\varphi \models A$  and  $\varphi \models B$ . From the definition of  $\models$  we have that  $\varphi \models A \wedge B$  if  $\varphi \models A$  and  $\varphi \models B$ , so we're done.

3. Case (funRight)

$$\frac{\varphi \vdash t_1 \quad \dots \quad \varphi \vdash t_n}{\varphi \vdash f(t_1, \dots, t_n)}$$

From the I.H follows that  $\varphi \models t_1$  through  $t_n$ . Since  $\models$  is closed by function application, then  $\varphi \models f(t_1, \dots, t_n)$ .

4. Case (AttLeft)

$$\frac{\Gamma, f(t_1, \dots, t_n) \vdash C}{\Gamma, t_1, \dots, t_n \vdash C}$$

From the I.H we have that  $\Gamma, f(t_1, \dots, t_n) \models C$ , but the set inclusion used in  $\models$  is closed by function application, we can apply  $f$  to  $t_1, \dots, t_n$  and therefore  $\Gamma, t_1, \dots, t_n \models C$ .

5. Case (DestrLeft)

$$\frac{\Gamma, s \vdash C}{\Gamma, f(t_1, \dots, t_n) \vdash C}$$

From the I.H we have that  $\Gamma, s \models C$ .  $\Gamma, f(t_1, \dots, t_n) \models C$  follows from the fact that  $f(t_1, \dots, t_n)$  reduces to  $s$  and the set inclusion used in  $\models$  is closed by reduction, therefore  $\Gamma, f(t_1, \dots, t_n) \models C$ . ■

**Proof of Proposition 1 (Termination of the approximation procedure of  $\mathfrak{F}(S)$ )** Given any finite term set  $S$ , the computation of  $b(S)$  always reaches a saturation point and hence always terminates.

*Proof:* We define a measure on the sets generated by  $\Rightarrow$  and show that this measure decreases monotonically in the sequence  $S \Rightarrow S_1 \Rightarrow \dots \Rightarrow S_n$ .

Given a signature  $\Sigma$ , a base set  $S$  and the maximum depth  $K$  which is the sum of the maximum nesting depth for all available rewrite rules with the maximum nesting depth of terms in  $S$ , we define the cardinality  $L$  of the set that contains all terms with maximum nesting smaller than  $K$  as:

$$L = \#\{t \in T(\Sigma, S) \mid |t| < K\}$$

where  $T(\Sigma, S)$  is the set of all terms generated from signature  $\Sigma$  using the names present in  $S$ . We can now define the measure  $S_{-i}$  as:  $S_{-i} = L - \#S_i$

Considering the sequence:  $S_0 \Rightarrow \dots \Rightarrow S_n$  we can prove that for any  $S_i$ ,  $S_{-i+1} < S_{-i}$  by looking at the way  $S_{i+1}$  is built from  $S_i$ .

In each step of  $\Rightarrow$ , a new term  $M$  is inserted in the previous set. Also, by the constraints imposed by  $\Rightarrow$ , we know that  $M < K$ . Seeing as the rule for the application of  $\Rightarrow$  is:

$$S, t_1, \dots, t_n \Rightarrow S, t_1, \dots, t_n, M$$

$$\text{if } \delta(g_1^{a_1}[t_1, \dots, t_k], \dots, g_i^{a_i}[t_k, \dots, t_n]) \rightarrow M, \forall i : a_i < K$$

and considering we impose that the right side of a reduction is a subterm of the left, and  $\forall i : a_i < K$ , we are sure that  $M < K$ .

In fact, the size  $L$  is the maximum possible size for any set  $S_i$ , since  $L$  is the size of the largest possible set of terms built from functions of  $\Sigma$ , whose nesting depth is smaller or equal to the maximum allowed in the rewrite rules for the functions in  $\Sigma$ .

Looking at  $S_{-i''}$ , we know the measure is strictly positive, because the existence of  $S_{i+1}$  implies that  $\#S_i < L$ , since  $L$  is the size of the maximum possible set of terms, and  $\#S_{i+1} > \#S_i$  (because  $S_{i+1}$  is obtained from  $S_i$  by adding one new term  $M$ ).

From the previous argument, it clearly follows that  $S_{-i+1''} < S_{-i''}$ , since  $S_{-i''} > 0$ ,  $S_{-i+1''} \geq 0$ ,  $L$  is fixed and  $\#S_{i+1} > \#S_i$ . Noting that it is only possible for  $S_{-i+1''} = 0$  if  $S_{i+1} = S_n$ , by the definition of  $L$ .

Given that  $S_{-0''} > S_{-1''} > \dots > S_{-n''}$ , the approximation procedure is proven to terminate. ■

**Proof of Proposition 1 (Approximation property)** Given a set of terms  $S$ , we have that:

$$\forall M \in \mathfrak{F}(S), \exists C, \bar{t} \in b(S) \text{ such that } M = C[\bar{t}]$$

Where  $C$  is an arbitrary function application context.

*Proof:* Induction on the generation of  $M \in \mathfrak{F}^{n+1}(S)$ . We use  $\mathfrak{F}^n(S)$  to denote the  $n$ th iteration over the saturation.

1. Case:  $M \in S$

Trivially satisfied by the empty context since for every  $M \in S$ ,  $M \in b(S)$ .

2. Case:  $M = f(M_1, \dots, M_n)$  such that  $\forall i : M_i \in \mathfrak{F}^n(S)$

From the I.H we have that  $\exists C_i, \bar{t}_i \in b(S) : M_i = C_i[\bar{t}_i]$  and thus we have the context  $C = f(C_1[-], \dots, C_n[-])$  such that  $M = C[\bar{t}_1, \dots, \bar{t}_n]$

3. Case:  $\delta(M_1, \dots, M_n) \rightarrow M$  such that  $\forall i : M_i \in \mathfrak{F}(S)$

From the I.H we have that  $\exists C_i, \bar{t}_i \in b(S) : M_i = C_i[\bar{t}_i]$ . Since  $\delta(M_1, \dots, M_n) \rightarrow M$  and we impose that the right-side of the rewrite rule to be a subterm of the left-side, we know that  $M$  is a subterm of some  $C_i[\bar{t}_i]$ . Therefore, the depth of any term  $M$  has to be such that:

$$C'[\bar{t}'] = M$$

where  $C'$  is a function context with nesting depth smaller than any  $C_i$ , and  $\bar{t}'$  is a subset of all the  $\bar{t}_i$ . This is so due to  $M$  being such that it can be constructed by a context sub-composed of the several contexts  $C_i$  that make up the left-side of the rewrite rule. ■

**Proof of Lemma 1 (Completeness of  $K$  i.r.t the approximation)** Given a set of terms  $S$ , if  $M \in b(S)$  then  $S \vdash M$ .

*Proof:* Let  $M \in b(S)$ . Then there is a sequence  $S_0 \Rightarrow S_1 \Rightarrow \dots \Rightarrow S_n$ , where  $M \in S_n$ . We proceed by induction on  $n$ .

– Case  $n = 0$ :

$$\frac{}{S \vdash M} \text{ (Id)}$$

– Case  $n + 1$ :

From  $\Rightarrow$  we have that  $S_{n+1}$  is generated through:

$$R, t_1, \dots, t_n \Rightarrow R, t_1, \dots, t_n, M \text{ with } S_n = R, \bar{t} \text{ and } S_{n+1} = R, \bar{t}, M$$

$$\text{if } \delta(g_1^{a_1}[t_1, \dots, t_k], \dots, g_i^{a_i}[t_{k'}, \dots, t_n]) \rightarrow M, \forall i : a_i < K$$

therefore, from the I.H we have that  $S \vdash t_1, \dots, S \vdash t_n$ . We can therefore build a derivation:

$$\frac{\frac{\frac{\vdots}{S \vdash \bar{t}}}{S, \bar{t} \vdash M} \text{(AttLeft)} \quad \frac{\frac{\vdots}{S, \bar{t} \vdash M} \text{(AttLeft)}}{S, \bar{t}, M \vdash M} \text{(DestrLeft)}}{S \vdash M} \text{(Cut)}$$

The schema for the derivation is a Cut, where the left premise follows from the I.H and the right premiss consists of a succession of applications of the (AttLeft) rule, which has the ability to apply arbitrary function symbols (of adequate arity) to terms on the left-side. With these applications, we reach a point where we have a term context that can match with the adequate rewrite rule that reduces to  $M$ . We then apply the (DestrLeft) rule to perform the reduction and conclude by using (Id). ■

**Lemma 3 (Monotonicity of  $\mathfrak{F}$  in Sub)** Let  $M$  be a term,  $N$  a closed term without destructors and  $x$  a variable of  $M$ :  $\mathfrak{F}(\text{sub}(M\{x \leftarrow N\})) \subseteq \mathfrak{F}(\text{sub}(M) \cup N)$

**Proof:** Induction on the length of the derivation of  $\text{sub}(M)$ .

– Case  $\text{sub}(\delta(t_1, \dots, t_n))$ :

Assuming some  $t_i$  contains  $x$ , we know that:

$$\text{sub}(\delta(t_1, \dots, t_n)\{x \leftarrow N\}) = \text{sub}(t_1\{x \leftarrow N\}) \cup \dots \cup \text{sub}(t_n\{x \leftarrow N\})$$

and, for any  $t_i$  containing  $x$  follows from the I.H that  $\mathfrak{F}(\text{sub}(t_i\{x \leftarrow N\})) \subseteq \mathfrak{F}(\text{sub}(t_i) \cup N)$ . Thus, since:

$$\mathfrak{F}(\text{sub}(\delta(t_1, \dots, t_n)\{x \leftarrow N\})) = \mathfrak{F}(\text{sub}(t_1\{x \leftarrow N\}) \cup \dots \cup \text{sub}(t_n\{x \leftarrow N\}))$$

we have that:  $\mathfrak{F}(\text{sub}(\delta(t_1, \dots, t_n)\{x \leftarrow N\})) \subseteq \mathfrak{F}(\text{sub}(\delta(t_1, \dots, t_n)) \cup N)$ .

– Case  $\text{sub}(f(t_1, \dots, t_n))$ :

In the case where  $f(\bar{t})$  does not contain any input-bound names, we're done. If not, then the reasoning is the same as in case  $\text{sub}(\delta(t_1, \dots, t_n))$ .

– Case  $\text{sub}(n)$ :

In the case where  $n$  is not a variable we're done. If  $n$  is a variable, then either  $n = x$  or  $n \neq x$ . If  $n = x$  then we have that  $\text{sub}(x) = \emptyset$  and  $\text{sub}(x\{x \leftarrow N\}) = N$ , from which we can conclude  $\mathfrak{F}(N) \subseteq \mathfrak{F}(\emptyset \cup N)$ . If  $n \neq x$  then  $\text{sub}(n\{x \leftarrow N\}) = \emptyset$  and trivially  $\mathfrak{F}(\emptyset) \subseteq \mathfrak{F}(\emptyset \cup N)$ . ■

**Lemma 4 (Monotonicity of K-Sets Under Substitution)** For any process  $P$ :  
 $K_{P\{x \leftarrow M\}} \subseteq \mathfrak{F}(K_P \cup M)$  where  $x$  is a variable in  $P$  and  $M$  is a closed term without destructors.

**Proof:** Induction on the structure of  $P$ .

- Case  $\mathbf{0}$  is trivial.
- Case  $P|Q$ :  
 If  $x$  occurs in  $P|Q$ , then it occurs in  $P$ , or  $Q$  or both. We know that:

$$K_{P|Q} = \{t \mid P|Q \vdash_k \phi_1 \cup \phi_2 \quad (\phi_1 \cup \phi_2) \models t\} = \mathfrak{F}(\phi_1 \cup \phi_2)$$

and

$$K_{(P|Q)\{x \leftarrow M\}} = \{t \mid (P|Q)\{x \leftarrow M\} \vdash_k \phi'_1 \cup \phi'_2 \quad (\phi'_1 \cup \phi'_2) \models t\} = \mathfrak{F}(\phi'_1 \cup \phi'_2)$$

from Lemma 3 follows that  $\mathfrak{F}(\phi'_1 \cup \phi'_2) \subseteq \mathfrak{F}(\phi_1 \cup \phi_2 \cup M)$ .

- Case  $P + Q$  is identical to  $P|Q$ .
- Case let  $n = T$  in  $P$ :  
 We know that the K-set of let  $n = T$  in  $P$  is  $\mathfrak{F}(\text{sub}(T) \cup \phi)$ , where  $P\{n \leftarrow T\} \vdash_k \phi$ ; and that  $x$  can occur in  $T$ , in the continuation  $P$ , or in both. Applying the substitution  $\{x \leftarrow M\}$  to the process, we end up with the K-set  $\mathfrak{F}(\text{sub}(T') \cup \phi')$ . From application of Lemma 3 follows that  $\mathfrak{F}(\text{sub}(T') \cup \phi') \subseteq \mathfrak{F}(\text{sub}(T) \cup \phi \cup M)$ .
- Case  $(\nu n)P$ :  
 We know that the K-set for  $(\nu n)P$  is  $\mathfrak{F}(\phi \uparrow n)$  where  $P \vdash_k \phi$  and the K-set for  $((\nu n)P)\{x \leftarrow M\}$  is  $\mathfrak{F}(\phi' \uparrow n)$ . Similarly, by Lemma 3 follows that  $\mathfrak{F}(\phi' \uparrow n) \subseteq \mathfrak{F}((\phi \cup M) \uparrow n)$ .
- Case  $n(x).P$  is trivial from the I.H since  $K_{n(x).P} = K_P$ .
- Case  $n(T).P$ :  
 From the I.H follows that  $K_{P\{x \leftarrow M\}} \subseteq \mathfrak{F}(K_P \cup M)$ , where  $P \vdash_k \phi$  and therefore  $K_P = \mathfrak{F}(\phi)$ . We know that  $K_{n(T).P} = \mathfrak{F}(\phi \cup \text{sub}(T))$ . Consider that  $P\{x \leftarrow M\} \vdash_k \phi'$  and  $T\{x \leftarrow M\} = T'$ , then from the I.H we have that  $\mathfrak{F}(\phi') \subseteq \mathfrak{F}(\phi \cup M)$ . Since:  $K_{(n(T).P)\{x \leftarrow M\}} = \mathfrak{F}(\phi' \cup \text{sub}(T'))$ , we conclude that  $K_{(n(T).P)\{x \leftarrow M\}} \subseteq \mathfrak{F}(K_{n(T).P} \cup M)$ .
- Case  $[T_1 = T_2].P$  is identical to the previous case.
- Case  $n(*).P$  is trivial from the I.H since  $K_{n(*).P} = K_P$ . ■

**Theorem 5 (Monotonicity of K-Sets under Synchronization)** Let  $P_c$  and  $A$  be a processes such that  $P_c \xrightarrow{c\langle M \rangle} P'_c$  and  $A \xrightarrow{c\langle x \rangle} A'$ . We have that  $K_{A'} \subseteq \mathfrak{F}(K_A \cup M)$ .

**Proof:** Induction on the structure of  $A'$ .

- Case  $\mathbf{0}$ :  
 We have  $A = c(x).\mathbf{0}$  and therefore  $K_{A'} = K_A$  since  $K_{\mathbf{0}} = \emptyset$ .
- Case  $R|S$ :  
 We have  $A = c(x).(R|S)$ . If  $x$  does not occur in  $(R|S)$ , then  $K_{A'} = K_A$ . If  $x$  occurs in  $(R|S)$ , by Lemma 4 we know that  $K_{(R|S)\{x \leftarrow M\}} \subseteq \mathfrak{F}(K_{(R|S)} \cup M)$ . Since  $K_{(R|S)} = K_{c(x).(R|S)}$ , we're done.

- Case  $R + S$  is similar to  $R|S$ .
- Case let  $n = T$  in  $R$ :  
We have  $A = c(x).let\ n = T\ in\ R$ . If  $x$  does not occur in let  $n = T$  in  $R$ , then  $K_{A'} = K_A$ . Otherwise, by Lemma 4 we know that  $K_{A'} \subseteq \mathfrak{F}(K_{let...} \cup M)$ . Since  $K_{let...} = K_{c(x).let...}$ , we're done.
- Case  $(\nu n)R$ :  
We have  $A = c(x).(\nu n)R$ . If  $x$  does not occur in  $(\nu n)R$  then  $K_A = K_{A'}$ . Otherwise, by Lemma 4 we know that  $K_{A'} \subseteq \mathfrak{F}(K_{(\nu n)R} \cup M)$ . Since  $K_{(\nu n)R} = K_{c(x).(\nu n)R}$ , we're done.
- Case  $n(y).R$ :  
We have  $A = c(x).n(y).R$ . If  $x$  does not occur in  $R$  then  $K_A = K_{A'}$ . Otherwise, by Lemma 4 we know that  $K_{A'} \subseteq \mathfrak{F}(K_{n(y).R} \cup M)$ . Since  $K_{n(y).R} = K_{c(x).n(y).R}$ , we're done.
- Case  $n\langle T \rangle.R$ :  
We have  $A = c(x).n\langle T \rangle.R$ . If  $x$  does not occur in  $n\langle T \rangle.R$  then  $K_A = K_{A'}$ . Otherwise, by Lemma 4 we know that  $K_{A'} \subseteq \mathfrak{F}(K_{n\langle T \rangle.R} \cup M)$ . Since  $K_{n\langle T \rangle.R} = K_{c(x).n\langle T \rangle.R}$ , we're done.
- Case  $[T_1 = T_2].R$ :  
Similar to the previous case, following from Lemma 4.
- Case  $n\langle * \rangle.R$ :  
Since  $K_{n\langle * \rangle.R} = K_R$ , trivial from Lemma 4.

■

**Lemma 5 (Attacker Simulation)** Let  $P_c$  and  $A$  be processes.

If  $(\nu \bar{n})(P_c \mid A) \longrightarrow (\nu \bar{n})(P'_c \mid A')$  and  $At = \text{Attacker}(P_c, S)$  with  $K_A \subseteq K_{A'}$  then  $\exists At', S'$  such that  $(\nu \bar{n})(P_c \mid A) \xrightarrow{*} (\nu \bar{n})(P'_c \mid At')$  and  $At' = \text{Attacker}(P'_c, S \cup S')$  and  $K_{A'} \subseteq K_{At'}$ .

**Proof:** By induction on  $(\nu \bar{n})(P_c \mid A) \longrightarrow (\nu \bar{n})(P'_c \mid A')$ :

- Cases where  $P_c \longrightarrow P'_c$  and  $A' \equiv A$ :  
Since  $A' \equiv A$  then we have  $S' = \emptyset$ ,  $At' = At$  and  $K_{A'} \subseteq K_{At'}$  follows trivially from the hypothesis.
- Cases where  $A \longrightarrow A'$  and  $P'_c \equiv P_c$ :  
Since  $A \longrightarrow A'$  then it is the case that  $K_A = K_{A'}$ . Given that  $P'_c \equiv P_c$  then  $At' = At$ , therefore trivially  $K_{A'} \subseteq K_{At'}$ .
- Case where  $c\langle M \rangle.P'_c \mid c(x).A'$ :  
We have that  $At \equiv c(x).At'$  and  $A \equiv c(x).A'$ . Given that  $c\langle M \rangle.P'_c \mid c(x).At' \longrightarrow P'_c \mid At'\{x \leftarrow M\}$ , we have  $S' = S \cup \{M\}$  and  $At'\{x \leftarrow M\} = \text{Attacker}(P'_c, S \cup S')$ . Given that  $K_A \subseteq K_{At'}$ , from Theorem 1 we have  $K_{A'} \subseteq \mathfrak{F}(K_A \cup M)$ , and from Theorem 2  $K_{At'\{x \leftarrow M\}} = \mathfrak{F}(K_{At'} \cup M)$ , then  $K_{A'} \subseteq K_{At'\{x \leftarrow M\}}$ .
- Case where  $c(x).P'_c \mid c\langle M \rangle.A'$ :  
We have that  $c(x).P'_c \mid c\langle M \rangle.A' \longrightarrow P'_c\{x \leftarrow M\} \mid A'$ . By definition we have that  $At \equiv c\langle * \rangle.At'$ . Therefore,  $c(x).P'_c \mid c\langle * \rangle.At' \longrightarrow P'_c\{x \leftarrow M\} \mid At'$ , where  $At' = \text{Attacker}(P'_c\{x \leftarrow M\}, S \cup S')$  with  $S' = \emptyset$ . We know that  $K_{At'} = K_{A'}$ . Since  $K_{A'} \subseteq K_A$  and from the hypothesis  $K_A \subseteq K_{At'}$ , we have that  $K_{A'} \subseteq K_{At'}$ .

■

**Lemma 6 (Process Knowledge)** Let  $P_c$  and  $A$  be processes and  $\phi$  a knowledge formula.

If  $(\nu \bar{n})(P_c \mid A) \xrightarrow{*} (\nu \bar{n})(P'_c \mid A')$  and  $P'_c \models \mathbb{K}\phi$  and  $At = \text{Attacker}(P_c, S)$  with

$K_A \subseteq K_{At}$  then  $\exists At', S'$  such that  $(\nu\bar{n})(P_c \mid At) \xrightarrow{*} (\nu\bar{n})(P'_c \mid At')$  and  $P'_c \models \mathbb{K}\phi$  and  $At' = \text{Attacker}(P'_c, S \cup S')$ .

**Proof:** By induction on  $(\nu\bar{n})(P_c \mid A) \xrightarrow{*} (\nu\bar{n})(P'_c \mid A')$ :

- Cases where  $A \longrightarrow A'$  and  $P' \equiv P$ , where  $P \longrightarrow P'$  and  $A' \equiv A$  or where  $c\langle M \rangle.P'_c \mid c(x).A'$  are trivial.
- Case where  $c(x).P'_c \mid c\langle M \rangle.A'$ :

We know  $P'_c\{x \leftarrow M\} \models \mathbb{K}\phi$  holds, therefore  $\phi \subseteq K_{P'_c\{x \leftarrow M\}}$ . From Lemma 4 follows that  $K_{P'_c\{x \leftarrow M\}} \subseteq \mathfrak{F}(K_{P'} \cup M)$ , where  $K_{P'} = K_P$ , hence  $\phi \subseteq \mathfrak{F}(K_P \cup M)$ .

By definition we have that  $At \equiv c\langle * \rangle.At'$ , and we have  $c\langle * \rangle.At' \xrightarrow{c\langle M' \rangle} At'$  such that  $M' \in \mathfrak{F}(K_{At} \cup \bar{x})$ .

Similarly, for  $P'_c\{x \leftarrow M'\} \models \mathbb{K}\phi$  to hold, it must be the case that  $\phi \subseteq \mathfrak{F}(K_P \cup M')$ . Since  $M \in K_A$  and  $K_A \subseteq K_{At}$ , it must be that  $M \in \mathfrak{F}(K_{At} \cup \bar{x})$ , therefore the process  $At$  can always send some  $M'$  such that  $M' = M$ , and we're done. ■