

Automatic testing equivalence verification of spi calculus specifications

Original

Automatic testing equivalence verification of spi calculus specifications / Durante, L; Sisto, Riccardo; Valenzano, A.. - In: ACM TRANSACTIONS ON SOFTWARE ENGINEERING AND METHODOLOGY. - ISSN 1049-331X. - 12:2(2003), pp. 222-284. [10.1145/941566.941570]

Availability:

This version is available at: 11583/1406236 since: 2024-07-04T08:32:05Z

Publisher:

ACM

Published

DOI:10.1145/941566.941570

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

ACM postprint/Author's Accepted Manuscript

© ACM 2003. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in ACM TRANSACTIONS ON SOFTWARE ENGINEERING AND METHODOLOGY, <http://dx.doi.org/10.1145/941566.941570>.

(Article begins on next page)

Automatic testing equivalence verification of spi calculus specifications

LUCA DURANTE

IEIIT - CNR

and

RICCARDO SISTO

Politecnico di Torino

and

ADRIANO VALENZANO

IEIIT - CNR

Testing equivalence is a powerful means for expressing the security properties of cryptographic protocols, but its formal verification is a difficult task because of the quantification over contexts on which it is based. Previous papers have provided insights on using theorem proving for the verification of testing equivalence of spi calculus specifications. This paper addresses the same verification problem, but uses a state exploration approach. The verification technique is based on the definition of an environment-sensitive, labeled transition system representing a spi calculus specification. Trace equivalence defined on such a transition system coincides with testing equivalence. Symbolic techniques are used to keep the set of traces finite. If a difference in the traces of two spi descriptions (typically a specification and the corresponding implementation of a protocol) is found, it can be used to automatically build the spi calculus description of an intruder process that can exploit the difference.

Categories and Subject Descriptors: C.2.2 [Computer-Communication Networks]: Network Protocols—*protocol verification*; D.2.4 [Software Engineering]: Software/Program Verification—*formal methods*; D.4.6 [Operating Systems]: Security and Protection—*verification*

General Terms: Languages, Security, Verification

Additional Key Words and Phrases: Cryptographic protocols, equivalence verification, state space exploration

This work was partially supported by the Italian National Council of Research in the framework of the research programme “Agenzia 2000” and by the Center for Multimedia Radio Communications of Politecnico di Torino.

Authors’ addresses: L. Durante and A. Valenzano, Istituto di Elettronica e Ingegneria dell’Informazione e delle Telecomunicazioni - CNR c/o Politecnico di Torino, corso Duca degli Abruzzi 24, I-10129 Torino (Italy) fax: +39 011 564 5429 e-mail: {luca.durante, adriano.valenzano}@polito.it; R. Sisto, Politecnico di Torino, corso Duca degli Abruzzi 24, I-10129 Torino (Italy) fax: +39 011 564 7099 e-mail: riccardo.sisto@polito.it

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.

© 2003 ACM 0000-0000/2003/0000-0001 \$5.00

1. INTRODUCTION

Due to the increasing importance of secure distributed applications, such as electronic commerce, formal verification of cryptographic protocols is being extensively studied by several researchers. Some have investigated proof techniques based on various proof systems and description formalisms [Abadi and Gordon 1998; Boreale et al. 2002; Paulson 1998; Schneider 1998]. Although partial automation of the proofs is possible using theorem provers, such an approach is generally very time consuming and requires considerable expertise. An alternative approach, which is simpler and quicker, is to use state exploration methods such as model checking (e.g., [Clarke et al. 2000; Lowe 1996; 1997; Lowe and Roscoe 1997; Millen et al. 1987]). This requires modeling of the protocol behavior as a reasonably sized finite state system, which generally entails introducing simplifying assumptions that can reduce the accuracy of the analysis. Nevertheless, this form of verification has the invaluable advantage of being fully automatic.

Both theorem proving and state exploration have been used for several description formalisms. In this paper, attention is focused on spi calculus [Abadi and Gordon 1999], a process algebra derived from π -calculus [Milner et al. 1992] with some simplification and the addition of cryptographic operations. The strength of spi calculus, compared to similar formalisms, is predominantly in its simplicity and accuracy in describing cryptographic protocols and their security requirements. In particular, in [Abadi and Gordon 1999], it is shown how authentication and secrecy can be expressed easily using testing equivalence. For example, if $P(M)$ is the description of a cryptographic protocol to exchange a message M , a strong secrecy requirement can be simply expressed by saying that for any M' , $P(M)$ and $P(M')$ must be testing equivalent, i.e., any tester process specified in spi must be unable to distinguish their behaviors (the testing equivalence we refer to in this paper is the *may-testing* equivalence defined in [De Nicola and Hennessy 1984]). This approach enjoys two important properties.

First, there is no need to develop a specification of the attacker, because an intruder definition is implicitly contained in the testing equivalence concept (tester processes actually represent all the intruder behaviors that can be specified in spi calculus). It is worth noting that the explicit intruder specification that is required by some other methods [Schneider 1998], is not only extra work, but also a potential weak point because it is a possible source of errors. From this point of view, tools such as Brutus [Clarke et al. 2000] and Athena [Song 1999], which do not require an explicit specification of the attackers, are certainly preferred.

The second important point in favor of the spi calculus approach is its accuracy in specifying both the protocol and its properties. Unlike most other specification formalisms, which only provide the means for specifying the exchanged messages and make implicit assumptions for checking and decoding the messages, spi calculus enables a precise description of all the operations performed by each protocol principal. It is also very powerful for describing the messages, because for example, it admits structured keys in addition to atomic keys. Moreover, the testing equivalence formulation of security properties is more accurate than alternative formulations based on intruder knowledge. For example, the secrecy expression mentioned above does not only require that the intruder will never know M , but it

requires that the intruder cannot infer anything about M . Naturally, weaker secrecy specifications can also be expressed by means of testing equivalence, if required.

The main open problem that remains with the spi calculus approach is to check testing equivalence in an efficient and easy manner. This is difficult because of universal quantification over testers: checking equivalence means checking that two processes are indistinguishable for *any* tester process, and there is an infinite number of such processes. This problem has been addressed in [Abadi and Gordon 1998] and [Boreale et al. 2002], where tractable proof methods aimed at checking the testing equivalence of spi calculus processes are introduced. In [Abadi and Gordon 1998], the proof method is based on a bisimulation relation that is a sufficient, but not necessary, condition for testing equivalence. In contrast, the proof method proposed in [Boreale et al. 2002] is based on a trace equivalence that is a necessary and sufficient condition for testing equivalence. Such a trace equivalence is defined over a contextual labeled transition system representing the protocol behavior constrained by the environment knowledge of names and keys. No attempt to implement such proof methods has been documented in the literature.

In contrast to the two approaches above, this paper presents a method of checking the spi calculus testing equivalence using exhaustive state exploration instead of proof techniques. The quantification over contexts problem is solved in a similar manner to that reported in [Boreale et al. 2002], i.e., by defining an environment-sensitive labeled transition system with a trace equivalence that is a necessary and sufficient condition for testing equivalence. However, to make state exploration possible and effective, a new problem has to be solved, that is, how the size of the trace sets to be explored can be kept within finite and reasonable bounds. This goal is achieved in two different ways.

First, because of the presence of replication expressions of the form $!P$, which are interpreted in spi calculus as an infinite number of copies of P running in parallel, and which make testing equivalence undecidable, only spi calculus processes having a finite user-selectable number of parallel instances are considered. In practice, our approach is to substitute any replication expression of the form $!P$ with a finite number n of parallel copies of P . This is equivalent to considering a finite and acyclic version of spi calculus. Note that the resulting language is not a finite control language because infinite execution paths are not possible because spi calculus has no recursion. Because the replication operator is generally used to represent parallel sessions of a cryptographic protocol and because each session has only finite execution paths, this restriction is equivalent to considering up to n parallel runs of the protocol. Consequently, attacks that are possible only with more than n parallel sessions cannot be detected in this way. A similar restriction is adopted in the literature whenever state exploration methods are used and this is generally considered acceptable, because bugs tend to appear with few parallel sessions. This has been proved for some special cases in [Lowe 1999].

The limitation on the maximum number of parallel processes is not sufficient to guarantee finite models: because the environment can in principle send, at any time, any data that can be produced by its knowledge as well as any fresh name or integer, the number of transitions that correspond to inputs from the environment is potentially infinite. Previous attempts to remove this problem were based mainly

on limitations on the length of messages an intruder can build and send [Clarke et al. 2000; Lowe 1997]. Instead, we propose a more powerful solution, based on a symbolic technique, that avoids the explicit representation of transitions but virtually considers the whole set of messages potentially produced by the intruder. More precisely, our approach is to represent an infinite set of transitions (corresponding to different input values from the environment) symbolically as a single transition with an abstract label called *generic term*. As we prove in the paper, this form of reduction does not imply any loss in accuracy; it is a key means of making traces enumerable and limits their number.

A preliminary version of the techniques reported in this paper has been presented in the conference literature [Durante et al. 2000].

The paper is organized as follows. Section 2 introduces the spi calculus language and introduces two sample cryptographic protocol specifications, which are used throughout the paper to illustrate our method. Then, Section 3 presents our environment-sensitive LTS model, the traces of which describe all the possible evolutions of the protocol and of the corresponding intruder knowledge. Here the derivation rules to build the set of traces for each spi process are rigorously defined, and the symbolic technique is proved to be loss-free. Section 4 shows that the trace semantics defined above are sound and complete with respect to testing equivalence. The completeness proof is particularly interesting because it shows how a trace difference between two spi processes can be used to automatically build the definition of a spi tester process that can distinguish the two processes. In practice, this is useful to automatically derive the spi definition of an attacker whenever a trace difference is found between specification and implementation. Section 5 compares our approach with related methods, and Section 6 concludes. A table of the notation used in the paper is provided in the Appendix.

2. SPI CALCULUS

Spi calculus is a process algebraic language defined in [Abadi and Gordon 1999] as an extension of π -calculus [Milner et al. 1992], and is specifically designed for the specification of cryptographic protocols. A comprehensive description of the language can be found in [Abadi and Gordon 1999]. The reader is assumed to be familiar with basic cryptographic techniques.

2.1 Syntax and informal semantics

Spi calculus includes two basic language elements: *terms*, which represent data (e.g., messages, channel identifiers, keys, key pairs, integers), and *processes*, which represent behaviors. Terms can be either atomic elements, classified as constants or variables, or structured terms built using term composition operators. The distinction between constants and variables was introduced by [Abadi and Gordon 1999], but was not present in π -calculus, where both constants and variables are simply referred to as names. As in π -calculus, and for the sake of simplicity, we use a formalization of spi calculus with a single set of atomic terms, called names. In this way, a variable is a name that can be replaced by any other term. A special name identifies the 0 integer constant.

This paper uses the following naming conventions:

$\sigma, \rho, \theta ::=$	terms	$P, Q, R ::=$	processes
m	name	$\bar{\sigma}(\rho).P$	output
0	name (the zero constant)	$\sigma(x).P$	input
x	name (a variable)	$P \mid Q$	composition
y	name (a variable)	$(\nu m) P$	restriction
(σ, ρ)	pair	0	nil
$suc(\sigma)$	successor	$[\sigma \text{ is } \rho] P$	match
$H(\sigma)$	hashing	$let (x, y) = \sigma \text{ in } P$	pair splitting
$\{\sigma\}_\rho$	shared-key encryption	$case \sigma \text{ of } 0 : P \text{ suc}(x) : Q$	integer case
σ^+	public part	$case \sigma \text{ of } \{x\}_\rho \text{ in } P$	shared-key decr.
σ^-	private part	$case \sigma \text{ of } \{[x]\}_\rho \text{ in } P$	decryption
$\{[\sigma]\}_\rho$	public-key encryption	$case \sigma \text{ of } [\{x\}]_\rho \text{ in } P$	signature check
$[\{\sigma\}]_\rho$	private-key signature		

Fig. 1. Syntax of spi calculus

- σ, ρ , and θ range over terms;
- P, Q , and R range over processes;
- m, x , and y range over names; x and y are preferably used for variables.

Figure 1 shows the syntax of the language available to the user. The left hand side of the table shows the term grammar. Terms can be combined by pairing successor, hashing and encryption operators, with the following informal meanings:

- (σ, ρ) is a *pair*, made up of the two components σ and ρ . For simplicity, the spi calculus does not have tuples, which, however, can be represented as nested pairs.
- $suc(\sigma)$ is the *successor* of σ . This operator has been introduced mainly to represent integer successors, but it can be used more generally as the abstract representation of an invertible term function.
- $H(\sigma)$ is the *hashing* of σ . The hashing function is assumed to be perfect and non-invertible.
- $\{\sigma\}_\rho$ is the encrypted message obtained by encrypting σ under key ρ using a shared-key cryptosystem. It can be decrypted using the same key.
- σ^+ and σ^- represent the public half and private half of a key pair σ , respectively. While it is possible to extract the public and private halves of a key pair, it is impossible to build the key pair from one of two halves.
- $\{[\sigma]\}_\rho$ is the result of the encryption of σ with public key ρ . It can be decrypted using the corresponding private key.
- $[\{\sigma\}]_\rho$ is the result of the signature (private-key encryption) of σ with private key ρ . It can be decrypted using the corresponding public key.

Terms are not typed: any term can be used in any context, without restriction. While this feature is seldom required to specify protocols, it is important because it enables the spi representation of attacks on the protocols based on type mismatches, where, for example, the attack is made possible because a malicious agent has led an honest agent to encrypt a given message using a nonce instead of a key.

The right hand side of Figure 1 shows the process algebraic operators used to build behavior expressions. Their informal meaning is:

— $\bar{\sigma}(\rho).P$ is an *output process*, ready to output term ρ on channel σ when synchronization with a corresponding input process occurs. The behavior after the synchronization is described by P .

— $\sigma(x).P$ is an *input process*, ready to perform an input from channel σ when synchronization with a corresponding output process occurs. After the synchronization, the message produced by the output process is assigned to variable x , and the subsequent behavior is described by P .

— $P \mid Q$ is a *parallel composition* where P and Q run in parallel. They may either synchronize with each other or separately with the external environment. This operator is commutative and associative.

— $(\nu m)P$ is a *restriction* process which produces a fresh, private name m and then behaves as described by P .

— $[\sigma \text{ is } \rho]P$ is a *match* process which behaves as described by P if terms σ and ρ are the same, and otherwise is stuck.

— 0 is the *nil* process: it is a stuck process.

— $\text{let } (x, y) = \sigma \text{ in } P$ is a *pair-splitting* process, where x and y are two distinct variables. If σ is a pair, its two components are assigned to x and y and the process continues as specified by P , otherwise the process is stuck.

— $\text{case } \sigma \text{ of } 0 : P \text{ suc}(x) : Q$ is an *integer case* process, which is used to invert the successor function. If σ is 0 , it behaves as specified by P ; if σ is $\text{suc}(\rho)$ for some ρ , it assigns ρ to x and then it behaves as specified by Q . Otherwise it is stuck.

— $\text{case } \sigma \text{ of } \{x\}_\rho \text{ in } P$ is a *shared-key decryption* process. If σ is a cyphertext taking the form $\{\theta\}_\rho$, it computes θ , assigns it to x and behaves as specified by P . Otherwise it is stuck.

— $\text{case } \sigma \text{ of } \{[x]\}_\rho \text{ in } P$ is a *decryption* process. If σ is the result of encrypting a message θ under a public key whose corresponding private key is ρ , the process assigns θ to x and behaves as specified by P . Otherwise it is stuck.

— $\text{case } \sigma \text{ of } [\{x\}]_\rho \text{ in } P$ is a *signature-check* process. If σ represents a message θ signed under a private key whose corresponding public key is ρ , it assigns θ to x and behaves as specified by P . Otherwise it is stuck.

It should be noted that, with respect to the syntax of the original spi calculus [Abadi and Gordon 1999], the replication operator ($!P$), which is interpreted as an unbounded number of copies of P running in parallel, was omitted. This has been done to have finite models. As explained below, a bounded version of replication can be introduced as a syntactical shortcut.

The following implicit assumptions of perfect encryption apply in spi as in other similar cryptographic protocol specification languages:

- an encrypted message can only be decrypted by means of the corresponding key;
- the encryption key cannot be deduced from the encrypted message;
- an encrypted message is sufficiently redundant so that the decryption algorithm can detect whether or not it has succeeded in its task;
- the attacker cannot guess and/or forge any secret protocol data.

$$\begin{array}{l|l}
\begin{array}{l}
1) A \rightarrow B : \{M\}_k \\
2) B \rightarrow A : H(M)
\end{array} &
\begin{array}{l}
P_A(M) \triangleq \overline{c}(\{M\}_k). c(x). [x \text{ is } H(M)] F(M) \\
P_B \triangleq c(y_1). \text{case } y_1 \text{ of } \{y_2\}_k \text{ in } \overline{c}(H(y_2)). 0 \\
P_{sample}(M) \triangleq (\nu k)(P_A(M) \mid P_B)
\end{array}
\end{array}$$

Fig. 2. A simple spi calculus specification

$$\begin{array}{l|l}
\begin{array}{l}
1) A \rightarrow S : \{k_{AB}\}_{k_{AS}} \\
2) S \rightarrow B : \{k_{AB}\}_{k_{SB}} \\
3) A \rightarrow B : \{M\}_{k_{AB}}
\end{array} &
\begin{array}{l}
P_A(M) \triangleq (\nu k_{AB}) (\overline{c_{AS}} \langle \{k_{AB}\}_{k_{AS}} \rangle . \overline{c_{AB}} \langle \{M\}_{k_{AB}} \rangle . 0) \\
P_S \triangleq c_{AS}(x_1) . \text{case } x_1 \text{ of } \{x_2\}_{k_{AS}} \text{ in } \overline{c_{SB}} \langle \{x_2\}_{k_{SB}} \rangle . 0 \\
P_B \triangleq c_{SB}(y_1) . \text{case } y_1 \text{ of } \{y_2\}_{k_{SB}} \text{ in } c_{AB}(y_3) . \\
\quad \text{case } y_3 \text{ of } \{y_4\}_{y_2} \text{ in } F(y_4) \\
P_{wmf}(M) \triangleq (\nu k_{AS}) (\nu k_{SB}) (P_A(M) \mid P_S \mid P_B)
\end{array}
\end{array}$$

Fig. 3. The wide-mouthed frog protocol

Figure 2 shows a simple protocol, inspired by [Fiore and Abadi 2001], where two agents A and B share a secret key k and exchange two simple messages. By adopting the informal, intuitive notation that is frequently used in the literature dealing with security protocols, the left hand side of Figure 2 shows the exchanges of messages involved in the protocol that proceed. In contrast, the right hand side shows the spi calculus specification of the protocol. Processes P_A and P_B represent the role of agents A and B , respectively. P_{sample} puts the two behaviors in parallel, thereby allowing their synchronization on channel c . Because c is public, it can also be accessed by a potential intruder.

A sends B a message M encoded by means of k over public channel c . B tries to decode it, and in the case of success, acknowledges A by sending back the hashed cleartext. Finally, A checks that the received message matches with $H(M)$ and then proceeds with further operations. $F(M)$ is an unspecified process, representing the behavior of A after the successful termination of the protocol session.

Figure 3 shows a simplified version of the wide-mouthed frog protocol which appeared in [Abadi and Gordon 1999], where two agents A and B establish a secure session to exchange information with the aid of a server S . A shared-key encryption scheme is adopted and agent A shares the key k_{AS} with the server, while k_{SB} is used for communications between S and B . Processes P_A , P_B , and P_S represent the role of agents A , B , and S , respectively, and they communicate on public channels c_{AS} , c_{SB} , and c_{AB} . To establish a secure connection with B , A picks up and sends a new session key k_{AB} encrypted under k_{AS} to S on channel c_{AS} . The server decrypts the received message with k_{AS} and forwards k_{AB} to B on channel c_{SB} by encrypting it under k_{SB} . Agent B , after receiving the session key from S , gets the cyphertext $\{M\}_{k_{AB}}$ from A on channel c_{AB} , extracts the cleartext M using k_{AB} , and then proceeds with further operations, represented by $F(y_4)$.

Note that the spi processes fully specify the behaviors, as described concisely on the left hand side of the figures. In particular, the generation of new keys, the encryption/decryption activities, and the messages' fields verification are explicitly expressed by means of the spi calculus constructions.

We introduce some further notation and definitions for the spi calculus, which

will be used throughout the paper.

A name occurrence is said to be *bound* in a process P if P contains an operator that binds it. The restriction operator (νm) binds name m , and any operator that implies an assignment of variable x binds x . Therefore, all the occurrences of x in processes taking the form $\sigma(x).P$ are bound by the input operator $\sigma(x)$. Similarly, all occurrences of x in expressions $(\text{case } \sigma \text{ of } \{x\}_\rho \text{ in } P)$, $(\text{case } \sigma \text{ of } \{[x]\}_\rho \text{ in } P)$, $(\text{case } \sigma \text{ of } [\{x\}]_\rho \text{ in } P)$, in the $\text{suc}(x) : Q$ part of an integer case, and the occurrences of x and y in expressions $(\text{let } (x, y) = \sigma \text{ in } P)$, are bound. A name occurrence is *free* in a process P if it is not bound. The set of names that occur free in P is called the set of free names of P and is denoted $fn(P)$. Similarly, $bn(P)$ is the set of bound names of P , and $n(P) = fn(P) \cup bn(P)$ the whole set of names of P .

For simplicity, it is assumed that different occurrences of binding operators always refer to different names. This is not a limitation, because processes that differ only by a renaming of bound names are equivalent ([Milner et al. 1992]). This assumption simplifies our formalization because in this way all names bound by assignment operators actually represent write-once, uniquely identified variables, and different nonces are always identified by different names. This also means that name overloading is not possible. In the following we use the notation $!P$ as a syntactical shortcut for indicating n copies of P running in parallel (i.e., $!P = \underbrace{P|P|\dots|P}_{n \text{ times}}$), where n is a predetermined finite natural number that can be specified

by the user. In this case, we also assume that all the bound names occurring in P are implicitly renamed so that the bound name uniqueness is preserved.

In what follows, \mathcal{A} denotes the set of spi calculus names, and $\mathcal{M}(\mathcal{A})$ the set of all spi calculus terms that can be built starting from \mathcal{A} .

If a term σ occurs as a sub-expression of another term ρ , then σ is called a *subterm* of ρ (a term is also considered a subterm of itself). This is also written $\sigma \preceq \rho$. A *term substitution list* (called a substitution list for short) is a list of term pairs taking the form $\lambda = \langle \sigma_1/m_1, \dots, \sigma_n/m_n \rangle$, where $m_i \in \mathcal{A}$ are pairwise different names and $\sigma_i \in \mathcal{M}(\mathcal{A})$ are terms. The postfix term operator $[\lambda]$ literally and simultaneously applies all the substitutions in λ , i.e., $\theta[\lambda]$ is θ with each occurrence of m_i replaced by σ_i . For simplicity, the notation $[\sigma_1/m_1, \dots, \sigma_n/m_n]$ denotes $[\langle \sigma_1/m_1, \dots, \sigma_n/m_n \rangle]$. Because the semantics of a substitution list λ does not depend on the order of its elements, λ can also be formally regarded as a *set* of substitution items. Each substitution list λ identifies a corresponding *term substitution function* from $\mathcal{M}(\mathcal{A})$ to $\mathcal{M}(\mathcal{A})$, and it maps any term θ onto $\theta[\lambda]$. In the remainder of this paper, substitution lists and their corresponding substitution functions are denoted in the same way and simply called *substitutions*. The substitution that does not replace any name is denoted \top , and is represented by an empty list. The composition of λ_1 and λ_2 , corresponding to the postfix operator $[\lambda_1][\lambda_2]$, is simply denoted $\lambda_1\lambda_2$, (i.e., $[\lambda_1\lambda_2] = [\lambda_1][\lambda_2]$). The postfix operator $[\lambda]$ and the corresponding substitution function can be extended to any domain of objects containing spi calculus terms in the obvious manner: if E is an object containing spi calculus terms (e.g., a process or a set of terms), $E[\lambda]$ is E with each term θ occurring in E replaced by $\theta[\lambda]$.

Substitutions are useful, for example, to formalize the assignment of variables: an

input process $\sigma(x).P$, which interacts with a corresponding output process $\bar{\sigma}(\rho).Q$, evolves into $P[\rho/x]$, i.e., P with all the occurrences of x in P replaced by ρ ¹.

In the remainder of the paper, λ ranges over substitutions and Σ ranges over sets of terms.

2.2 Reaction operational semantics

The spi calculus operational semantics was originally defined in [Abadi and Gordon 1999] by means of the reduction relation $>$, the structural equivalence \equiv , and the reaction relation \rightarrow .

The reduction relation $>$ is a binary relation on processes, such that $P > Q$ means that P is a process that can only reduce to Q by performing a successful internal operation such as a check, pair-splitting, or decryption operation. It is defined by the following axioms:

$$[\sigma \text{ is } \sigma] P > P \quad (1)$$

$$\text{let } (x, y) = (\sigma, \rho) \text{ in } P > P[\sigma/x, \rho/y] \quad (2)$$

$$\text{case } 0 \text{ of } 0 : P \text{ suc}(x) : Q > P \quad (3)$$

$$\text{case suc}(\sigma) \text{ of } 0 : P \text{ suc}(x) : Q > Q[\sigma/x] \quad (4)$$

$$\text{case } \{\sigma\}_\rho \text{ of } \{x\}_\rho \text{ in } P > P[\sigma/x] \quad (5)$$

$$\text{case } \{[\sigma]\}_{\rho^+} \text{ of } \{[x]\}_{\rho^-} \text{ in } P > P[\sigma/x] \quad (6)$$

$$\text{case } \{[\sigma]\}_{\rho^-} \text{ of } \{[x]\}_{\rho^+} \text{ in } P > P[\sigma/x] \quad (7)$$

Structural equivalence \equiv is an equivalence on processes based on simple and intuitive operator properties (e.g., commutative and associative properties), such that processes bound by the reduction relation are also considered equivalent. It is defined as the least relation that satisfies the axioms:

$$P \equiv P \quad (8) \quad P \mid (Q \mid R) \equiv (P \mid Q) \mid R \quad (11)$$

$$P \mid 0 \equiv P \quad (9) \quad (\nu m_1)(\nu m_2)P \equiv (\nu m_2)(\nu m_1)P \quad (12)$$

$$P \mid Q \equiv Q \mid P \quad (10) \quad (\nu b)0 \equiv 0 \quad (13)$$

and the rules:

$$\frac{m \notin \text{fn}(P)}{(\nu m)(P \mid Q) \equiv P \mid (\nu m)Q} \quad (14) \quad \frac{P \equiv Q \quad Q \equiv R}{P \equiv R} \quad (17)$$

$$\frac{P > Q}{P \equiv Q} \quad (15) \quad \frac{P \equiv P'}{P \mid Q \equiv P' \mid Q} \quad (18)$$

$$\frac{P \equiv Q}{Q \equiv P} \quad (16) \quad \frac{P \equiv P'}{(\nu m)P \equiv (\nu m)P'} \quad (19)$$

Finally, the reaction relation \rightarrow is a binary relation on processes such that $P \rightarrow P'$ means that process P can evolve into P' by performing an internal synchronization,

¹In the original spi calculus definition, only the occurrences of x that are free in P are substituted. Because we assume that each variable has a unique binding operator, this distinction is not required in our setting.

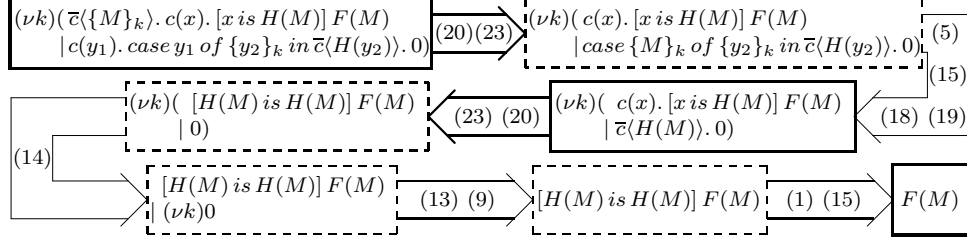


Fig. 4. Reaction operational semantics applied to the specification of Figure 2

i.e., a synchronization between any two (sub)-processes of P . It is the least relation on processes that satisfies the axiom,

$$\bar{\sigma}(\rho).P|\sigma(x).Q \rightarrow P|Q[\rho/x] \quad (20)$$

and the rules:

$$\frac{P \equiv P' \quad P' \rightarrow Q' \quad Q' \equiv Q}{P \rightarrow Q} \quad (21) \quad \frac{P \rightarrow P'}{P|Q \rightarrow P'|Q} \quad (22) \quad \frac{P \rightarrow P'}{(\nu m)P \rightarrow (\nu m)P'} \quad (23)$$

Figure 4 shows the application of reaction operational semantics to the process $P_{\text{sample}}(M)$ of Figure 2. In particular, thick boxes represent processes ready to evolve by reactions, and thick arrows indicate the reaction rules involved. Dashed boxes represent processes that can be simplified by means of structural equivalence, and thin arrows indicate the corresponding rules.

Starting from the upper left box, and following the arrows, rule (20) allows the two (sub)processes A and B involved in the parallel composition to synchronize and to assign $\{M\}_k$ to y_1 , while rule (23) extends the reaction even in the presence of a restriction construction, leading to the behavior shown in the upper right box. Then rule (5) is applied to decrypt $\{M\}_k$ and to assign y_2 , while rule (15) joins the reduction relation with structural equivalence. Rules (18) and (19) extend the structural equivalence even in the presence of parallel composition and restriction, respectively. The behavior expressions are subsequently reduced by a further application of rules (20) and (23). Because the restricted name k does not belong to the free names of $[H(M) \text{ is } H(M)] F(M)$, by rule (14) the restriction can be moved inside the right hand side of the parallel composition, and rules (13) and (9) can be applied to definitely remove the parallel composition. Finally, $[H(M) \text{ is } H(M)] F(M)$ is reduced to $F(M)$ by means of rules (1) and (15).

It is worth noting that, exploiting rule (21), which joins reaction and structural equivalence, the dashed sections can be eliminated, and the whole evolution of the process can be described by the following sequence of reactions:

$$\begin{aligned} &(\nu k)(\bar{c}(\{M\}_k).c(x).[x \text{ is } H(M)] F(M) \mid c(y_1).case y_1 \text{ of } \{y_2\}_k \text{ in } \bar{c}(H(y_2)).0) \\ &\longrightarrow (\nu k)(c(x).[x \text{ is } H(M)] F(M) \mid \bar{c}(H(M)).0) \longrightarrow F(M) \end{aligned}$$

2.3 Testing equivalence

Testing equivalence is used to compare spi processes from the point of view of security. More precisely, we use the may-testing equivalence definition initially

introduced by De Nicola and Hennessy on CCS [De Nicola and Hennessy 1984], and we adapt it to the spi calculus, which is a CCS extension. As explained in [Abadi and Gordon 1999], the choice of may-testing stems from the fact that may-testing corresponds to safety, and typical security properties are safety properties.

A *test* R is a spi calculus process that, when run in parallel with a process under test, P , may signal that P has passed a test by means of a distinguished success action ω . Formally, the original spi calculus process grammar is extended by adding the new syntactic rule:

$$P, Q, R ::= \Omega$$

where Ω is a distinguished process that can perform only ω . It is assumed that Ω is reserved for the definition of tests only.

For a process P , the predicate P *exhibits* ω , written $P \downarrow \omega$, is defined by the axiom $\Omega \downarrow \omega$ and by the rules:

$$\frac{P \downarrow \omega}{(P|Q) \downarrow \omega} \quad (24) \quad \frac{P \downarrow \omega}{(\nu m)P \downarrow \omega} \quad (25) \quad \frac{P \equiv Q \quad Q \downarrow \omega}{P \downarrow \omega} \quad (26)$$

whereas the convergence predicate $P \Downarrow \omega$ is defined as:

$$P \Downarrow \omega \triangleq \exists Q \mid (P(\rightarrow) * Q) \wedge (Q \downarrow \omega) \quad (27)$$

where $(\rightarrow)^*$ is the reflexive and transitive closure of \rightarrow . A process P may pass a test R if and only if $(P|R) \Downarrow \omega$.

The testing preorder \sqsubseteq and the testing equivalence \simeq are then defined as:

$$P \sqsubseteq Q \triangleq \forall R \quad ((P|R) \Downarrow \omega \implies (Q|R) \Downarrow \omega) \quad (28)$$

$$P \simeq Q \triangleq (P \sqsubseteq Q) \wedge (Q \sqsubseteq P) \quad (29)$$

$P \simeq Q$ means that the tests that P and Q may pass are the same. In other words, $P \simeq Q$ means that no test can distinguish between P and Q .

If we consider the sample protocol introduced in Figure 2, it may be necessary to express a secrecy property, i.e., that an intruder cannot infer anything about M . Such a property can easily be expressed by requiring that, for each M' , $P(M) \simeq P(M')$ [Abadi and Gordon 1999].

Authenticity properties can also be expressed in terms of equivalence. We return, for example, to the protocol introduced in Figure 3. Authenticity, in this case, requires that agent B applies F to the message M sent by A, independently of what happens during the protocol session. This means that no attacker can force B to apply F to a cleartext that is different from M. For this reason we write the reference spi specification in Figure 5, where P_{BSpec} is a process that behaves like P_B in Figure 3 except for the fact that the (unspecified) process F is always applied to M, independent of the message received on c_{AB} , just as if agent B already knew M in some other way. Similarly, $P_{wmfSpec}$ is the *correct* specification of the protocol obtained by running P_A , P_S , and P_{BSpec} in parallel. To express the authenticity property in terms of equivalence, we can require that for each M , $P_{wmf}(M) \simeq P_{wmfSpec}(M)$ [Abadi and Gordon 1999].

$$\begin{aligned}
P_A(M) &\triangleq (\nu k_{AB}) (\overline{c_{AS}} \langle \{k_{AB}\}_{k_{AS}} \rangle . \overline{c_{AB}} \langle \{M\}_{k_{AB}} \rangle . 0) \\
P_S &\triangleq_{c_{AS}} (x_1) . \text{case } x_1 \text{ of } \{x_2\}_{k_{AS}} \text{ in } \overline{c_{SB}} \langle \{x_2\}_{k_{SB}} \rangle . 0 \\
P_{BSpec}(M) &\triangleq_{c_{SB}} (y_1) . \text{case } y_1 \text{ of } \{y_2\}_{k_{SB}} \text{ in } c_{AB}(y_3) . \text{case } y_3 \text{ of } \{y_4\}_{y_2} \text{ in } F(M) \\
P_{wmspec}(M) &\triangleq (\nu k_{AS}) (\nu k_{SB}) (P_A(M) \mid P_S \mid P_{BSpec}(M))
\end{aligned}$$

Fig. 5. Reference specification of the wide-mouthed frog protocol

3. THE ENVIRONMENT-SENSITIVE LTS MODEL

Our first goal is to define an *environment-sensitive labeled transition system* (ES-LTS) that describes all the possible interactions of a given spi calculus process with its environment. The ES-LTS concept is borrowed from [Boreale et al. 2002], although the labels of our specific ES-LTS are different from those proposed in [Boreale et al. 2002], as explained in Section 5.

The environment considered in building the ES-LTS is most powerful, power being measured by the knowledge the environment acquires by interacting with the spi process. In analogy with [Boreale et al. 2002], each state of our ES-LTS is made up of a spi calculus process P and an environment knowledge K , and is denoted $K \triangleright P$. As elucidated below, K incorporates all the knowledge that an intruder can have acquired during the preceding interactions with the spi process, and is represented in a finite and minimized form.

Transitions are described by the following syntactical form, already introduced in [Boreale et al. 2002]:

$$K \triangleright P \xrightarrow[\phi]{\mu} K' \triangleright P' \quad (30)$$

where label μ represents the action performed by process P , and ϕ represents the complementary action performed by the environment. The exact form of labels μ and ϕ , which is explained below, has been carefully defined so as to make our ES-LTS trace equivalence sound and complete with respect to testing equivalence. Attention has also been paid to avoid unnecessary replication of information, therefore minimizing label sizes and simplifying their comparison.

As described above, symbolic representations are used to make the ES-LTS finite. Both a *concrete ES-LTS*, where no symbolic representations are used, and a *symbolic ES-LTS*, which is the finite symbolic version of the previous representation, are defined. The form taken by the symbolic transitions is slightly different from that taken by the concrete transitions:

$$(K \triangleright P)_{\Upsilon, \Lambda} \xrightarrow[\phi]{\mu} (K' \triangleright P')_{\Upsilon', \Lambda'} \quad (31)$$

In this case, K , P , μ , and ϕ have the same meaning as in the concrete ES-LTS, but may be symbolic, i.e., they may represent infinite sets of corresponding concrete elements. The state subscripts Υ, Λ , whose exact meaning is explained in Sections 3.4 and 3.5, specify how the symbolic elements must be interpreted, i.e., which concrete values they can take. Moreover, to represent situations where a symbolic transition is only possible for some of the concrete states represented by its symbolic starting state, symbolic labels may also include additional elements specifying which concrete behaviors are allowed to take the transition.

In the remainder of this paper we adopt the usual notation for functions, where $f : A \rightarrow B$ is a (possibly partial) function from A to B that maps elements $a \in A$ onto $f(a) \in B$. According to the set-theoretic interpretation, f also denotes the set of pairs $f = \{(a, b) \mid a \in A, b \in B, b = f(a)\}$ that must satisfy the well-known function properties. We also use the notation $\text{dom}(f) = \{a \in A \mid (a, b) \in f \text{ for some } b \in B\}$ and $\text{im}(f) = \{b \in B \mid (a, b) \in f \text{ for some } a \in A\}$ for the domain and the image of f , and, if $S \subseteq A$, we denote the image of S by f as $f(S)$.

3.1 Knowledge representation

The simplest form of intruder knowledge representation, which is adopted by most cryptographic protocol analyzers (e.g., [Clarke et al. 2000; Song 1999]), consists of only the set of data that can be collected by the intruder. Such a representation is adequate provided the properties to be checked are sufficiently simple (e.g., *the intruder will never discover certain data*), but it is inadequate for checking testing equivalence. This can be understood by considering, for example, that the spi processes $P \triangleq \bar{c}(M).\bar{c}(N).\bar{c}(M).0$ and $Q \triangleq \bar{c}(M).\bar{c}(N).\bar{c}(N).0$ are not testing equivalent, although the set of data that the intruder can learn from each of these is the same at each step (after the first step, the intruder learns M , after the second step it learns N , and after the third step it learns no new data). Naturally, a simple test that can distinguish between P and Q is $R \triangleq c(x_1).c(x_2).c(x_3).[x_1 \text{ is } x_3]\Omega$, which checks if the data received third is equal to the first. To make this type of test possible in our intruder model, a more accurate representation of the intruder knowledge is needed, such that the order in which data terms are learned is also recorded.

Based on this consideration, the model for the intruder knowledge that we propose is a set of learned data terms with labeling of its elements that uniquely identifies them according to the order in which they have been added to the set. Such labels could be interpreted as the names of the variables used by the intruder to store the learned data. The set of learned data that we use is a *minimized* set, in the sense that it does not include redundant elements, but it includes only the minimum set of *elementary* terms by which the intruder can build all the possible messages that can be generated with all the data it has observed. For example, if the intruder receives $\{m\}_k$, and it does not know k , then $\{m\}_k$ is added to the set of learned data. If instead the intruder already knows k , then m is added to the set, but $\{m\}_k$ is not, because it can be built using m and k . Similarly, if the set of learned data is $\{m, \{m\}_k\}$, and the intruder receives k , then the new set will be $\{m, k\}$, because $\{m\}_k$ can now be built from m and k .

The formal definition of our intruder knowledge representation requires some preliminary definitions.

The closure of a set of terms $\Sigma \subseteq \mathcal{M}(\mathcal{A})$ is denoted $\widehat{\Sigma}$ and is defined as the set of all spi calculus terms that can be built by combining the elements of Σ by means of the term operators defined in spi (left hand side of Figure 1) and their inverses. Formally, $\widehat{\Sigma}$ is the least set of terms such that, for each σ, σ_1 , and $\sigma_2 \in \mathcal{M}(\mathcal{A})$, the following closure rules hold:

$$\sigma \in \Sigma \Rightarrow \sigma \in \widehat{\Sigma} \quad (32)$$

$$\sigma \in \widehat{\Sigma} \Rightarrow \text{suc}(\sigma) \in \widehat{\Sigma} \quad (\text{successor}) \quad (33)$$

$$\sigma_1 \in \widehat{\Sigma} \wedge \sigma_2 \in \widehat{\Sigma} \Rightarrow (\sigma_1, \sigma_2) \in \widehat{\Sigma} \quad (\text{pairing}) \quad (34)$$

$$\sigma_1 \in \widehat{\Sigma} \wedge \sigma_2 \in \widehat{\Sigma} \Rightarrow \{\sigma_1\}_{\sigma_2} \in \widehat{\Sigma} \quad (\text{shared-key encryption}) \quad (35)$$

$$\sigma \in \widehat{\Sigma} \Rightarrow H(\sigma) \in \widehat{\Sigma} \quad (\text{hashing}) \quad (36)$$

$$\sigma_1 \in \widehat{\Sigma} \wedge \sigma_2^+ \in \widehat{\Sigma} \Rightarrow \{[\sigma_1]\}_{\sigma_2^+} \in \widehat{\Sigma} \quad (\text{public-key encryption}) \quad (37)$$

$$\sigma_1 \in \widehat{\Sigma} \wedge \sigma_2^- \in \widehat{\Sigma} \Rightarrow \{[\sigma_1]\}_{\sigma_2^-} \in \widehat{\Sigma} \quad (\text{private-key signature}) \quad (38)$$

$$\sigma \in \widehat{\Sigma} \Rightarrow \sigma^+ \in \widehat{\Sigma} \wedge \sigma^- \in \widehat{\Sigma} \quad (\text{key-pair extraction}) \quad (39)$$

$$\text{suc}(\sigma) \in \widehat{\Sigma} \Rightarrow \sigma \in \widehat{\Sigma} \quad (\text{predecessor}) \quad (40)$$

$$(\sigma_1, \sigma_2) \in \widehat{\Sigma} \Rightarrow \sigma_1 \in \widehat{\Sigma} \wedge \sigma_2 \in \widehat{\Sigma} \quad (\text{projection}) \quad (41)$$

$$\{\sigma_1\}_{\sigma_2} \in \widehat{\Sigma} \wedge \sigma_2 \in \widehat{\Sigma} \Rightarrow \sigma_1 \in \widehat{\Sigma} \quad (\text{shared-key decryption}) \quad (42)$$

$$\{[\sigma_1]\}_{\sigma_2^+} \in \widehat{\Sigma} \wedge \sigma_2^+ \in \widehat{\Sigma} \Rightarrow \sigma_1 \in \widehat{\Sigma} \quad (\text{public-key decryption}) \quad (43)$$

$$\{[\sigma_1]\}_{\sigma_2^-} \in \widehat{\Sigma} \wedge \sigma_2^- \in \widehat{\Sigma} \Rightarrow \sigma_1 \in \widehat{\Sigma} \quad (\text{signature check}) \quad (44)$$

$$\sigma^+ \in \widehat{\Sigma} \wedge \sigma^- \in \widehat{\Sigma} \Rightarrow \sigma \in \widehat{\Sigma} \quad (\text{key-pair reconstruction}) \quad (45)$$

Rules (33)-(45) can be divided into two groups: rules (33)-(39) are *constructive* rules, related to term composition operators, whereas rules (40)-(45) are the corresponding *destructive* rules, related to term decomposition operations. Naturally, the hashing rule has no destructive counterpart.

We say that a set of terms is *finite* if it contains a finite number of finite length elements. Given a finite set of terms Σ , the *minimal closure seed* of Σ , denoted $\overline{\Sigma}$, is defined as the subset of $\widehat{\Sigma}$ that satisfies the following predicates for each $m \in \mathcal{A}$, and for each $\sigma, \sigma_1, \sigma_2 \in \mathcal{M}(\mathcal{A})$:

$$m \in \overline{\Sigma} \Leftrightarrow m \in \widehat{\Sigma} \quad (46)$$

$$\text{suc}(\sigma) \notin \overline{\Sigma} \quad (47)$$

$$(\sigma_1, \sigma_2) \notin \overline{\Sigma} \quad (48)$$

$$\{\sigma_1\}_{\sigma_2} \in \overline{\Sigma} \Leftrightarrow \sigma_2 \notin \widehat{\Sigma} \quad (49)$$

$$H(\sigma) \in \overline{\Sigma} \Leftrightarrow \sigma \notin \widehat{\Sigma} \quad (50)$$

$$\{[\sigma_1]\}_{\sigma_2^+} \in \overline{\Sigma} \Leftrightarrow \sigma_2^+ \notin \widehat{\Sigma} \vee \sigma_1 \notin \widehat{\Sigma} \quad (51)$$

$$\{[\sigma_1]\}_{\sigma_2^-} \in \overline{\Sigma} \Leftrightarrow \sigma_2^- \notin \widehat{\Sigma} \vee \sigma_1 \notin \widehat{\Sigma} \quad (52)$$

$$\sigma^+ \in \overline{\Sigma} \Leftrightarrow \sigma \notin \widehat{\Sigma} \quad (53)$$

$$\sigma^- \in \overline{\Sigma} \Leftrightarrow \sigma \notin \widehat{\Sigma} \quad (54)$$

In practice, $\overline{\Sigma}$ is the minimum subset of $\widehat{\Sigma}$ that is sufficient to regenerate $\widehat{\Sigma}$. The rationale behind rules (46)-(54) is that the minimal closure seed must contain only those elements of the closure that cannot be regenerated by simpler closure elements. For example, $\overline{\Sigma}$ contains all the atomic elements of $\widehat{\Sigma}$ (by (46)) because there is no way to regenerate them. However, it does not contain terms taking the forms $\text{suc}(\sigma)$ and (σ_1, σ_2) (by (47) and (48)), because such terms can be respectively generated by σ and by σ_1 and σ_2 , which necessarily also belong to the closure by

rules (40) and (41). An element of $\widehat{\Sigma}$ taking the form $\{[\sigma_1]\}_{\sigma_2^+}$ can be regenerated by simpler elements only if both σ_1 and σ_2^+ belong to $\widehat{\Sigma}$. This is why (by (51)) it belongs to $\overline{\Sigma}$ iff at least one of its components does not belong to $\widehat{\Sigma}$. Note that $\overline{\Sigma}$ is not necessarily a subset of Σ , because Σ could include terms that can be decomposed into simpler subterms not included in Σ . For example, if $\Sigma = \{\{[m]\}_{a^+}, a^-\}$, then we have that $\overline{\Sigma} = \{\{[m]\}_{a^+}, a^-, m\}$, because a^- can be used to decrypt $\{[m]\}_{a^+}$, therefore computing m , but $\{[m]\}_{a^+}$ cannot be regenerated using only a^- and m .

The minimal closure seed $\overline{\Sigma}$ is uniquely defined because $\widehat{\Sigma}$ is uniquely defined, and, for each $\sigma \in \widehat{\Sigma}$, there is exactly one of the rules (46)-(54) that uniquely determines if $\sigma \in \overline{\Sigma}$.

$\overline{\Sigma}$ has some useful properties that make it a suitable candidate for a finite and minimized representation of the term generation capabilities of an intruder that has had access to the set of terms Σ . First, if Σ is finite, then $\overline{\Sigma}$ is also finite and is the minimal set of terms having the same closure of Σ , where minimality means that any element in $\overline{\Sigma}$ cannot be built from the other elements, i.e., $\overline{\Sigma}$ does not include redundant terms. Moreover, $\overline{\Sigma}$ is computable and the question if a finite term σ belongs to the closure of Σ can be algorithmically decided using only $\overline{\Sigma}$. Such properties are now formally stated and proved.

We start with the finiteness and minimality of $\overline{\Sigma}$.

THEOREM 3.1. (Finiteness) *For each finite set of terms $\Sigma \subseteq \mathcal{M}(\mathcal{A})$, $\overline{\Sigma}$ is finite.*

PROOF. By inspection of rules (46)-(54) it is clear that any structured element of $\overline{\Sigma}$ has at least a subterm that does not belong to $\widehat{\Sigma}$, i.e., a subterm that cannot be built by combining the elements of Σ . This means that any element of $\overline{\Sigma}$ is necessarily a subterm of an element of Σ . However, because Σ is finite, the subterms of its elements are also finite. This implies that $\overline{\Sigma}$ is finite. \square

THEOREM 3.2. (Minimality) *Let $\Sigma \subseteq \mathcal{M}(\mathcal{A})$ be a finite set of terms, and $\sigma \in \overline{\Sigma}$. Then $(\overline{\Sigma} \setminus \{\sigma\}) \subset \widehat{\Sigma}$.*

PROOF. From rules (33)-(45) it is clear that $(\overline{\Sigma} \setminus \{\sigma\}) \subseteq \widehat{\Sigma}$. Therefore, it is sufficient to show that there is at least one element of $\widehat{\Sigma}$ that does not belong to $(\overline{\Sigma} \setminus \{\sigma\})$. Such an element is σ , which belongs to $\widehat{\Sigma}$ by rule (32), but it does not belong to $(\overline{\Sigma} \setminus \{\sigma\})$ because, by rules (46)-(54), it is either a name or a structured term with at least a subterm that cannot be built from the elements of $\widehat{\Sigma}$ (which includes $\widehat{\Sigma}$ by definition). \square

Before formulating and proving the other properties, we require more definitions.

Define $r(\sigma, \Sigma)$ as the boolean value that is obtained by executing the algorithm shown in Figure 6.

The value computed by $r(\sigma, \Sigma)$ is true if σ can be built from the elements of Σ using only constructive closure rules. The base of recursion simply says that $r(\sigma, \Sigma)$ is true for any $\sigma \in \Sigma$ and false for any atomic term not belonging to Σ . For non-atomic terms not belonging to Σ , $r(\sigma, \Sigma)$ is true if the components of σ can be built using only constructive rules. This interpretation implies that if $r(\sigma, \Sigma)$ is true, then $\sigma \in \widehat{\Sigma}$.


```

boolean  $r(\sigma, \Sigma)$  {
  if  $\sigma \in \Sigma$  then return TRUE;
  else if  $\sigma = \text{suc}(\sigma_1)$  then return  $r(\sigma_1, \Sigma)$ ;
  else if  $\sigma = (\sigma_1, \sigma_2)$  then return  $r(\sigma_1, \Sigma) \wedge r(\sigma_2, \Sigma)$ ;
  else if  $\sigma = \{\sigma_1\}_{\sigma_2}$  then return  $r(\sigma_1, \Sigma) \wedge r(\sigma_2, \Sigma)$ ;
  else if  $\sigma = H(\sigma_1)$  then return  $r(\sigma_1, \Sigma)$ ;
  else if  $\sigma = \{[\sigma_1]\}_{\sigma_2^+}$  then return  $r(\sigma_1, \Sigma) \wedge r(\sigma_2^+, \Sigma)$ ;
  else if  $\sigma = \{[\sigma_1]\}_{\sigma_2^-}$  then return  $r(\sigma_1, \Sigma) \wedge r(\sigma_2^-, \Sigma)$ ;
  else if  $\sigma = \sigma_1^+$  then return  $r(\sigma_1, \Sigma)$ ;
  else if  $\sigma = \sigma_1^-$  then return  $r(\sigma_1, \Sigma)$ ;
  else ( $\sigma \in \mathcal{A} \setminus \Sigma$ ) return FALSE;
}

```

Fig. 6. The recursive algorithm to compute $r(\sigma, \Sigma)$

The computation of $\overline{\Sigma}$ from Σ can be performed by a sequence of closure-preserving transformations. Such a sequence is called a *reduction* of Σ , and each transformation is called a *reduction step*. Each reduction step exploits one of closure rules (32)-(45) and transforms the current set of terms by applying a *reduction rule*. A *reduction rule* is formally defined as a triple $U = \langle \Sigma_I, C, \Sigma_O \rangle$, where C is the exploited closure rule, Σ_I (*premises* of U) is the set of terms that must be included in the current set of terms to enable the application of the rule, and Σ_O (*conclusions* of U) is the set of terms that replace the premises. Applying reduction rule U to a finite set of terms Σ means eliminating Σ_I from and adding Σ_O to Σ .

This is written $\Sigma \xrightarrow{U} \Sigma'$, where $\Sigma' = (\Sigma \setminus \Sigma_I) \cup \Sigma_O$ is the resulting set.

Given a finite set of terms Σ , a *reduction of Σ* is formally defined as an alternating sequence of finite sets of terms Σ_i and reduction rules U_i , denoted by:

$$\Sigma_0 \xrightarrow{U_0} \Sigma_1 \xrightarrow{U_1} \Sigma_2 \cdots \Sigma_{k-1} \xrightarrow{U_{k-1}} \Sigma_k$$

such that $\Sigma_0 = \Sigma$ and $U_i \in \mathcal{U}(\Sigma_i)$, where $\mathcal{U}(\Sigma_i)$ is the least set of reduction rules that satisfies the following logical implications for each $\sigma, \sigma_1, \sigma_2 \in \mathcal{M}(\mathcal{A})$:

$$\text{if } H(\sigma) \in \Sigma_i \wedge r(\sigma, \Sigma_i) \text{ then } \langle \{H(\sigma)\}, (36), \emptyset \rangle \in \mathcal{U}(\Sigma_i) \quad (55)$$

$$\begin{aligned} \text{if } \{[\sigma_1]\}_{\sigma_2^+} \in \Sigma_i \wedge r(\sigma_1, \Sigma_i) \wedge r(\sigma_2^+, \Sigma_i) \\ \text{then } \langle \{[\sigma_1]\}_{\sigma_2^+}, (37), \emptyset \rangle \in \mathcal{U}(\Sigma_i) \end{aligned} \quad (56)$$

$$\begin{aligned} \text{if } \{[\sigma_1]\}_{\sigma_2^-} \in \Sigma_i \wedge r(\sigma_1, \Sigma_i) \wedge r(\sigma_2^-, \Sigma_i) \\ \text{then } \langle \{[\sigma_1]\}_{\sigma_2^-}, (38), \emptyset \rangle \in \mathcal{U}(\Sigma_i) \end{aligned} \quad (57)$$

$$\text{if } \sigma^+ \in \Sigma_i \wedge r(\sigma, \Sigma_i) \text{ then } \langle \{\sigma^+\}, (39), \emptyset \rangle \in \mathcal{U}(\Sigma_i) \quad (58)$$

$$\text{if } \sigma^- \in \Sigma_i \wedge r(\sigma, \Sigma_i) \text{ then } \langle \{\sigma^-\}, (39), \emptyset \rangle \in \mathcal{U}(\Sigma_i) \quad (59)$$

$$\text{if } \text{suc}(\sigma) \in \Sigma_i \text{ then } \langle \{\text{suc}(\sigma)\}, (40), \{\sigma\} \rangle \in \mathcal{U}(\Sigma_i) \quad (60)$$

$$\text{if } (\sigma_1, \sigma_2) \in \Sigma_i \text{ then } \langle \{(\sigma_1, \sigma_2)\}, (41), \{\sigma_1, \sigma_2\} \rangle \in \mathcal{U}(\Sigma_i) \quad (61)$$

$$\text{if } \{\sigma_1\}_{\sigma_2} \in \Sigma_i \wedge r(\sigma_2, \Sigma_i) \text{ then } \langle \{\{\sigma_1\}_{\sigma_2}\}, (42), \{\sigma_1\} \rangle \in \mathcal{U}(\Sigma_i) \quad (62)$$

$$\begin{aligned} \text{if } \{[\sigma_1]\}_{\sigma_2^+} \in \Sigma_i \wedge r(\sigma_2^-, \Sigma_i) \wedge \neg r(\sigma_1, \Sigma_i) \\ \text{then } \langle \{[\sigma_1]\}_{\sigma_2^+}, (43), \{\sigma_1, \{[\sigma_1]\}_{\sigma_2^+}\} \rangle \in \mathcal{U}(\Sigma_i) \end{aligned} \quad (63)$$

$$\begin{aligned}
& \text{if } [\{\sigma_1\}]_{\sigma_2^-} \in \Sigma_i \wedge r(\sigma_2^+, \Sigma_i) \wedge \neg r(\sigma_1, \Sigma_i) \\
& \quad \text{then } \langle [\{\sigma_1\}]_{\sigma_2^-}, (44), \{\sigma_1, [\{\sigma_1\}]_{\sigma_2^-}\} \rangle \in \mathcal{U}(\Sigma_i) \quad (64) \\
& \text{if } \sigma^+ \in \Sigma_i \wedge \sigma^- \in \Sigma_i \text{ then } \langle \{\sigma^+, \sigma^-\}, (45), \{\sigma\} \rangle \in \mathcal{U}(\Sigma_i) \quad (65)
\end{aligned}$$

Reduction rules $U_i \in \mathcal{U}(\Sigma_i)$ associated with constructive rules (i.e., (55)-(59)) have as premises terms that can be regenerated using the other elements of Σ_i . For this reason, such rules simply remove the premises. Reduction rules $U_i \in \mathcal{U}(\Sigma_i)$ associated with destructive closure rules, instead, act on premises from which it is possible to extract new terms. They substitute the premises with a set of simpler subterms extracted from them, which is sufficient to regenerate them. Note that in reductions coming from equations (63) and (64), the premises are re-introduced in the conclusions, because they cannot be regenerated otherwise. From the above considerations, reductions preserve closures, i.e., the following proposition holds:

PROPOSITION 3.3. *if $\Sigma \xrightarrow{U} \Sigma'$ is a one-step reduction, then $\widehat{\Sigma} = \widehat{\Sigma}'$*

A reduction can be used to compute $\overline{\Sigma}$ from Σ as is expressed by the following proposition:

PROPOSITION 3.4. *Given a finite set of terms Σ , there exists a finite reduction of Σ*

$$\Sigma = \Sigma_0 \xrightarrow{U_0} \Sigma_1 \cdots \Sigma_{k-1} \xrightarrow{U_{k-1}} \Sigma_k$$

such that $\Sigma_k = \overline{\Sigma}$

PROOF. A reduction that leads from Σ to $\overline{\Sigma}$ can be found if we keep applying reduction rules $U_i \in \mathcal{U}(\Sigma_i)$ as long as they can be applied. It can be verified by inspection that reduction rules $U_i \in \mathcal{U}(\Sigma_i)$ always add subterms of terms that are already included in Σ_i , and that each $U_i \in \mathcal{U}(\Sigma_i)$ can occur only once in a given reduction, because, once applied, it is no longer included in $\mathcal{U}(\Sigma_{i+1}) \cdots \mathcal{U}(\Sigma_k)$, because the corresponding pre-condition in equations (55)-(64) becomes false. Consequently, because Σ is finite, it is guaranteed that in a finite number of steps we reach a Σ_k on which no reduction rule can be applied. When this happens, $\Sigma_k = \overline{\Sigma}$, because all the pre-conditions of equations (55)-(65) are false, which implies that Σ_k satisfies the minimal closure seed definition equations (46)-(54). \square

The results that have been proved so far allow us to state and prove the final theorems of closure preservation, computability, and decidability.

THEOREM 3.5. (*Closure preservation*) *For each finite set of terms $\Sigma \subseteq \mathcal{M}(\mathcal{A})$, $\widehat{\overline{\Sigma}} = \widehat{\Sigma}$*

PROOF. The theorem directly descends from propositions 3.3 and 3.4 \square

THEOREM 3.6. (*Computability*) *For each finite set of terms $\Sigma \subseteq \mathcal{M}(\mathcal{A})$, $\overline{\Sigma}$ can be computed in a finite number of steps.*

PROOF. The theorem directly descends from propositions 3.3 and 3.4 and from the fact that the computation of $r(\sigma, \Sigma_i)$ takes a finite number of steps, because σ and Σ_i are finite. \square

THEOREM 3.7. (Decidability) *Let $\sigma \in \mathcal{M}(\mathcal{A})$ be any finite term and $\Sigma \subseteq \mathcal{M}(\mathcal{A})$ be a finite set of terms. Then, the question if $\sigma \in \widehat{\Sigma}$ is decidable.*

PROOF. We claim that $\sigma \in \widehat{\Sigma}$ iff $r(\sigma, \overline{\Sigma})$ is true. This claim can be proved by induction, proving directly the cases $\sigma \in \overline{\Sigma}$ and $\sigma \in \mathcal{A} \setminus \overline{\Sigma}$, and proving the other cases inductively. Once the claim is proved, it remains to be shown that the computation of $r(\sigma, \overline{\Sigma})$ takes a finite number of steps, but this descends directly from the fact that σ and $\overline{\Sigma}$ are finite and from theorem 3.6. \square

The results just given mean that if a new term ρ is added to a minimal closure seed $\overline{\Sigma}$, the new minimal closure seed $\overline{\Sigma} \cup \{\rho\}$ can be computed by a reduction that starts from $\overline{\Sigma} \cup \{\rho\}$. In general, the net effect of such a reduction is to eliminate some elements from the set of terms and to add some other new elements to it. We denote $\delta_{\overline{\Sigma}}^-(\rho)$ the set of eliminated elements and $\delta_{\overline{\Sigma}}^+(\rho)$ the set of added elements. Formally:

$$\delta_{\overline{\Sigma}}^-(\rho) = \overline{\Sigma} \setminus (\overline{\Sigma} \cup \{\rho\}) \quad (66)$$

$$\delta_{\overline{\Sigma}}^+(\rho) = (\overline{\Sigma} \cup \{\rho\}) \setminus \overline{\Sigma} \quad (67)$$

Another fact that may be found during a reduction is that *something more* becomes known about a term, which, however, is not removed from the set. This may happen only when a public-key encrypted term $\{[\sigma_1]\}_{\sigma_2^+}$ or a digitally signed term $\{[\sigma_1]\}_{\sigma_2^-}$ can be decoded by the appropriate decoding key, without knowing the corresponding encoding key. In this case, σ_1 becomes known, but the encoded term cannot be removed, because it cannot be regenerated. We denote as $\delta_{\overline{\Sigma}}^{\equiv}(\rho)$ the set of terms that become decipherable after the addition of ρ without being eliminated from the minimal closure seed. Formally, we first define $dec(\overline{\Sigma}, \sigma)$ as the predicate that is true if σ is an encoded term belonging to $\overline{\Sigma}$ that can be decoded with the elements of $\overline{\Sigma}$. We have that $dec(\overline{\Sigma}, \{[\sigma_1]\}_{\sigma_2^+}) = \{[\sigma_1]\}_{\sigma_2^+} \in \overline{\Sigma} \wedge r(\sigma_2^-, \overline{\Sigma})$, $dec(\overline{\Sigma}, \{[\sigma_1]\}_{\sigma_2^-}) = \{[\sigma_1]\}_{\sigma_2^-} \in \overline{\Sigma} \wedge r(\sigma_2^+, \overline{\Sigma})$, and $dec(\overline{\Sigma}, \sigma) = false$ if σ takes neither the form $\{[\sigma_1]\}_{\sigma_2^+}$ nor the form $\{[\sigma_1]\}_{\sigma_2^-}$. Then, $\delta_{\overline{\Sigma}}^{\equiv}(\rho)$ is defined as:

$$\delta_{\overline{\Sigma}}^{\equiv}(\rho) = \{\sigma \mid dec(\overline{\Sigma} \cup \{\rho\}, \sigma) \wedge \neg dec(\overline{\Sigma}, \sigma)\} \quad (68)$$

i.e., $\delta_{\overline{\Sigma}}^{\equiv}(\rho)$ contains just those terms of $\overline{\Sigma} \cup \{\rho\}$ that have become decipherable during the last reduction. We always have $\delta_{\overline{\Sigma}}^-(\rho) \cap \delta_{\overline{\Sigma}}^{\equiv}(\rho) = \emptyset$.

As an example consider Figure 7 where term $\rho = k_3^-$ is added to a minimal closure seed $\overline{\Sigma} = \{c, \{[\{k_1\}_{k_2}]\}_{k_3^+}, \{[m]\}_{k_1^+}, k_1^+, k_2\}$ leading to $\Sigma_0 = \{c, \{[\{k_1\}_{k_2}]\}_{k_3^+}, \{[m]\}_{k_1^+}, k_1^+, k_2, k_3^-\}$.

The column labeled Σ_i reports the set of terms at each reduction step; the column $\mathcal{U}(\Sigma_i)$ shows the set of reduction rules enabled in each Σ_i , with the rule that is applied first. Reduction starts from Σ_0 : term $\{k_1\}_{k_2}$ is added to Σ_0 after decoding term $\{[\{k_1\}_{k_2}]\}_{k_3^+}$ by means of key k_3^- . However, the encoded term remains in the set because the encoding key k_3^+ is not known. In Σ_1 , both $\{k_1\}_{k_2}$ and k_2 are available, therefore allowing the addition of k_1 to and the elimination of $\{k_1\}_{k_2}$ from Σ_1 . In Σ_2 , two reduction rules can be applied: m can be extracted from $\{[m]\}_{k_1^+}$

$\overline{\Sigma} = \{c, \{[\{k_1\}_{k_2}]\}_{k_3^+}, \{[m]\}_{k_1^+}, k_1^+, k_2\}$		$\rho = k_3^-$	$\Sigma_0 = \overline{\Sigma} \cup \rho$
i	Σ_i	$\mathcal{U}(\Sigma_i)$	
0	$\{c, \{[\{k_1\}_{k_2}]\}_{k_3^+}, \{[m]\}_{k_1^+}, k_1^+, k_2, k_3^-\}$	$\langle \{[\{k_1\}_{k_2}]\}_{k_3^+}, (43), \{k_1\}_{k_2}, \{[\{k_1\}_{k_2}]\}_{k_3^+} \rangle$	
1	$\{c, \{[\{k_1\}_{k_2}]\}_{k_3^+}, \{[m]\}_{k_1^+}, k_1^+, k_2, k_3^-, \{k_1\}_{k_2}\}$	$\langle \{k_1\}_{k_2}, (42), \{k_1\} \rangle$	
2	$\{c, \{[\{k_1\}_{k_2}]\}_{k_3^+}, \{[m]\}_{k_1^+}, k_1^+, k_2, k_3^-, k_1\}$	$\langle \{[m]\}_{k_1^+}, (43), \{m, \{[m]\}_{k_1^+}\}, \langle k_1^+, (39), \emptyset \rangle$	
3	$\{c, \{[\{k_1\}_{k_2}]\}_{k_3^+}, \{[m]\}_{k_1^+}, k_1^+, k_2, k_3^-, k_1, m\}$	$\langle \{[m]\}_{k_1^+}, (37), \emptyset \rangle, \langle \{k_1^+, (39), \emptyset \rangle$	
4	$\{c, \{[\{k_1\}_{k_2}]\}_{k_3^+}, k_1^+, k_2, k_3^-, k_1, m\}$	$\langle \{k_1^+, (39), \emptyset \rangle$	
5	$\{c, \{[\{k_1\}_{k_2}]\}_{k_3^+}, k_2, k_3^-, k_1, m\}$		
$\overline{\Sigma} \cup \{\rho\} = \{c, \{[\{k_1\}_{k_2}]\}_{k_3^+}, k_2, k_3^-, k_1, m\}$			
$\delta_{\overline{\Sigma}}^-(\rho) = \{\{[m]\}_{k_1^+}, k_1^+\}$		$\delta_{\overline{\Sigma}}^{\equiv}(\rho) = \{\{[\{k_1\}_{k_2}]\}_{k_3^+}\}$	$\delta_{\overline{\Sigma}}^+(\rho) = \{k_3^-, k_1, m\}$

Fig. 7. An example of reduction

because k_1^- can be computed from key pair k_1 , and k_1^+ can be removed because it can also be computed from k_1 . In our example, the first rule is applied, leading to Σ_3 . It can be noted that rule (39) is still applicable, along with a new rule that allows the elimination of $\{[m]\}_{k_1^+}$, because now both m and k_1^- are available. The two rules are applied sequentially, and we therefore obtain $\Sigma_5 = \overline{\Sigma} \cup \{\rho\}$. In conclusion, we have $\delta_{\overline{\Sigma}}^-(\rho) = \{\{[m]\}_{k_1^+}, k_1^+\}$, $\delta_{\overline{\Sigma}}^{\equiv}(\rho) = \{\{[\{k_1\}_{k_2}]\}_{k_3^+}\}$ and $\delta_{\overline{\Sigma}}^+(\rho) = \{k_3^-, k_1, m\}$.

Finally, we define the intruder knowledge representation. Denote I as the infinite, countable, totally ordered set of names used to uniquely identify the data items learned by the intruder. Such names are called *indexes* and are assumed to be distinct from the spi calculus names (i.e., $I \cap \mathcal{A} = \emptyset$). We simply denote by $<$ the total ordering relation on I , and, for each element $l \in I$, we denote $next(l)$ the successor of l in I . We also denote as $next^n()$ the application of $next()$ n times.

In our framework, the intruder knowledge is formally represented by a bijective function $K : \overline{\Sigma} \rightarrow L$, where the domain $\overline{\Sigma}$ is the minimal closure seed of the set of terms that the intruder has learned (denoted Σ), and the image $L \subset I$ is the finite set of indexes uniquely identifying them. Naturally, the intruder term-generation capabilities depend on $\overline{\Sigma}$. If $\sigma \in \mathcal{M}(\mathcal{A})$ is a finite term, σ can be *produced* by K , written $K \vdash \sigma$, iff $\sigma \in \widehat{\Sigma}$. Similarly, σ can be produced by $\overline{\Sigma}$, written $\overline{\Sigma} \vdash \sigma$, iff $\sigma \in \widehat{\Sigma}$. The decidability of $K \vdash \sigma$ and of $\overline{\Sigma} \vdash \sigma$ comes from theorem 3.7.

The total order on I has been introduced to represent the order in which knowledge elements are added to the minimal closure seed of the set of terms known by the intruder, i.e., for any $l, l' \in im(K)$, $l < l'$ means that the element with index l has been added to $dom(K)$ before the element with index l' . More precisely, the elements added to $\overline{\Sigma}$ during the protocol run are labeled by consecutive indexes. If $Max(im(K))$ denotes the maximum element in $im(K)$, and K_i is the initial intruder knowledge, successive data terms added to the intruder knowledge domain during the protocol run are labeled $next(Max(im(K_i)))$, $next^2(Max(im(K_i)))$,

etc.

When the intruder receives a new data term ρ , its knowledge K is updated, and the new knowledge is denoted $f(\rho, K)$. If $\overline{\Sigma} = \text{dom}(K)$, $f(\rho, K)$ is the result of eliminating the elements of $\delta_{\overline{\Sigma}}^-(\rho)$ from and adding the elements of $\delta_{\overline{\Sigma}}^+(\rho)$ to the intruder knowledge domain. Because indexes are assigned to the new elements of the intruder knowledge domain in a predefined order, $f(\rho, K)$ is uniquely defined if we specify the order in which the elements of $\delta_{\overline{\Sigma}}^+(\rho)$ are added. Define the *normalized reduction* of a set of terms Σ as the reduction of Σ such that, at each step, in the case where $\mathcal{U}(\Sigma_i)$ contains two or more reduction rules, the one whose premises have been in Σ_i for a longer time is applied. Then, the order in which the elements of $\delta_{\overline{\Sigma}}^+(\rho)$ are added to $f(\rho, K)$ is the same as the order in which they are added during the normalized reduction of $(\overline{\Sigma} \cup \{\rho\})$. For example, consider the knowledge function:

$$K = \{\langle c, l_0 \rangle, \langle \{[k_1]_{k_2}\}_{k_3^+}, l_1 \rangle, \langle \{[m]\}_{k_1^+}, l_2 \rangle, \langle k_1^+, l_3 \rangle, \langle k_2, l_4 \rangle\} \quad (69)$$

where the labeling convention $l_i < l_j \Leftrightarrow i < j$ is used. Note that the domain of K is the set $\overline{\Sigma}$ of Figure 7, and, when k_3^- is added to the intruder knowledge, the reduction of $\overline{\Sigma} \cup \{k_3^-\}$ in Figure 7 is the normalized reduction. This can be verified, because in both reduction steps 2 and 3 we have two possible reduction rules, one with premise $\{[m]\}_{k_1^+}$ and the other with premise k_1^+ , whose indexes are respectively l_2 and l_3 , and the first index is applied in both cases. Then, we have that:

$$f(k_3^-, K) = \{\langle c, l_0 \rangle, \langle \{[k_1]_{k_2}\}_{k_3^+}, l_1 \rangle, \langle k_2, l_4 \rangle, \langle k_3^-, l_5 \rangle, \langle k_1, l_6 \rangle, \langle m, l_7 \rangle\} \quad (70)$$

because $\{[m]\}_{k_1^+}$ and k_1^+ belong to $\delta_{\overline{\Sigma}}^-(k_3^-)$ and are removed from $\overline{\Sigma}$, while k_3^- , k_1 and m are respectively added to the set in this order.

3.2 Canonical representations

When checking testing equivalence, it is necessary to abstract away from the exact value of the exchanged data, because only the manner such data is perceived by the intruder is important. For example, the spi processes $P \triangleq (\nu k)\overline{c}\langle\{M\}_k\rangle.0$ and $Q \triangleq (\nu k)\overline{c}\langle\{N\}_k\rangle.0$ are testing equivalent despite their outputs being different, because both the output of P and the output of Q are encrypted messages that the intruder cannot decrypt. Therefore, there is no test by which the intruder behaves differently if combined with P rather than with Q . To recognize this equivalence, the output of $\{M\}_k$ and the output of $\{N\}_k$ should produce the same ES-LTS transition label.

For this reason, we introduce the *canonical representation of a term σ with respect to an intruder knowledge K* , which expresses how σ is related to K (this is also how the intruder perceives and interprets σ).

To introduce this concept, we first extend the notion of substitution. The substitution lists originally introduced in Section 2 act only on names. If this constraint is relaxed, we have substitution lists $\lambda = \langle \sigma_1/\rho_1, \dots, \sigma_n/\rho_n \rangle$, where ρ_i can be non-atomic terms. If λ is one of such extended substitution lists, the postfix operator $[\lambda]$ simultaneously replaces each occurrence of ρ_i with σ_i for $1 \leq i \leq n$,

with the rule that if $\rho_i \preceq \rho_j$ and $i \neq j$, any occurrence of ρ_i inside an occurrence of ρ_j is not substituted. For example, $\{M\}_k[A/M, B/\{M\}_k] = B$, but $\{M\}_k[A/M, B/\{M\}_h] = \{A\}_k$.

The canonical representation of a term θ with respect to intruder knowledge K is a spi calculus term defined over the extended set of names $\mathcal{A} \cup I$, obtained by substituting each $\rho \in \text{dom}(K)$ occurring in θ by its corresponding unique identifier $K(\rho)$. Such a substitution is represented by a substitution list made up of an item $K(\rho)/\rho$ for each $\rho \in \text{dom}(K)$. This substitution is denoted K , and, consequently, the canonical representation is written $\theta[K]$. In a similar way, it is possible to define the inverse substitution, i.e., the substitution corresponding to function K^{-1} . Naturally, $[K][K^{-1}]$ gives the null substitution.

Because $[K]$ substitutes each occurrence of ρ with its corresponding index $K(\rho)$, $\theta[K]$ actually specifies how θ can be regenerated using the data items available in the intruder knowledge, each identified by its index. For example, if we consider knowledge functions K and $f(k_3^-, K)$ defined respectively in (69) and (70), we have that $\{[m]\}_{k_1^+}[K] = l_2$ and $\{[m]\}_{k_1^+}[f(k_3^-, K)] = \{[l_7]\}_{l_6^+}$, which means that $\{[m]\}_{k_1^+}$ is the element labeled l_2 in K , and it can be regenerated from $f(k_3^-, K)$ by encrypting the element labeled l_7 with the public part of the key pair labeled l_6 . It can be easily verified that if $K \vdash \theta$, then $\theta[K] \in \mathcal{M}(I)$, i.e., the canonical representation of a term that can be produced by K does not contain spi calculus names that are not indexes, because it can be built using only the elements of the intruder knowledge.

Canonical representations can be extended in the obvious way to any object containing terms. For example, the canonical representation of a substitution list $\lambda = \langle \sigma_1/\rho_1, \dots, \sigma_n/\rho_n \rangle$ with respect to K is $\lambda[K] = \langle \sigma_1[K]/\rho_1[K], \dots, \sigma_n[K]/\rho_n[K] \rangle$.

3.3 The concrete ES-LTS derivation system

As described above, transitions of the concrete ES-LTS take the syntactical form (30). Process action labels μ and complementary environment action labels ϕ may take three different forms, corresponding to three different types of transitions:

$$K \triangleright P \xrightarrow{\tau} K \triangleright P' \quad (71)$$

$$K \triangleright P \xrightarrow[\delta_K(\rho)]{\sigma[K']} K' \triangleright P' \quad (72)$$

$$K \triangleright P \xrightarrow[\rho[K]]{\sigma[K]} K \triangleright P' \quad (73)$$

Transitions taking the form (71) are related to synchronization events occurring inside the spi process, and for this reason they leave the environment unaware of what has happened, and do not involve any *knowledge migration* (K does not change). In this case, the process action has the special symbol τ , which represents an internal synchronization, and the complementary environment action is missing. Because such transitions actually represent spi calculus reactions, they are referred to as *reaction transitions*.

Transitions taking the second and third forms are related to synchronization events between the spi process and the environment.

A transition taking the form (72) is referred to as an *output transition* and rep-

resents a protocol output on channel σ . It implies a data transfer from the process to the environment. The process action (seen by the environment) is summarized by label $\overline{\sigma}[K']$, i.e., the canonical representation of the channel with respect to the updated intruder knowledge K' , with an overline symbol indicating, as in spi calculus, that it is an output. The data ρ sent by the process is not specified here, but its effect on the environment knowledge is included in the environment action label $\delta_K(\rho)$, the exact format of which will be detailed below.

A transition taking the form (73) is referred to as an *input transition* and represents an input on channel σ . It implies a data transfer from the environment to the process. Therefore, no modification in environment knowledge takes place. The process action label is analogous to the previous transition, whereas the complementary action label $\rho[K]$ is the canonical representation of the data term generated by the intruder, i.e., the description of how ρ can be built using the current intruder knowledge.

To formally define our environment-sensitive LTS, we use a set of derivation rules based upon the derivation system defined in [Abadi and Gordon 1999] for the reaction relation, and recalled in Section 2.2: in practice, the reaction relation definition axioms and rules are assumed implicitly.

The transition relation $\xrightarrow[\phi]{\mu}$ for the concrete ES-LTS is the least relation that satisfies the following derivation rules:

$$\frac{P \equiv P' \quad K_1 \triangleright P' \xrightarrow[\phi]{\mu} K_2 \triangleright Q' \quad Q' \equiv Q}{K_1 \triangleright P \xrightarrow[\phi]{\mu} K_2 \triangleright Q} \quad (74) \qquad \frac{P \longrightarrow P'}{K \triangleright P \xrightarrow[-]{\tau} K \triangleright P'} \quad (75)$$

$$\frac{K \triangleright P \xrightarrow[\phi]{\mu} K' \triangleright P' \quad \mu \neq \tau}{K \triangleright (P|Q) \xrightarrow[\phi]{\mu} K' \triangleright (P'|Q)} \quad (76) \qquad \frac{K \triangleright P \xrightarrow[\phi]{\mu} K' \triangleright P'}{K \triangleright (\nu b)P \xrightarrow[\phi]{\mu} K' \triangleright (\nu b)P'} \quad (77)$$

$$\frac{K \vdash \sigma \quad K' = f(\rho, K)}{K \triangleright \overline{\sigma}(\rho).P \xrightarrow[\langle \delta_K^-(\rho), \delta_K^-(\rho), \rho \rangle_{[K']}]{\overline{\sigma}[K']}} K' \triangleright P} \quad (78) \qquad \frac{K \vdash \sigma \quad K \vdash \rho}{K \triangleright \sigma(x).P \xrightarrow[\rho[K]]{\overline{\sigma}[K]}} K \triangleright P[\rho/x]} \quad (79)$$

Rules (74) and (75) represent the link point between our derivation rules and the original spi calculus semantics. Rule (74) specifies that structural equivalent expressions have the same behavior, whereas rule (75) specifies that $\xrightarrow[-]{\tau}$ actually means reaction.

Rule (76) specifies how the parallel operator is handled. Note that the case of reaction transitions ($\mu = \tau$) is not specifically addressed here, because it falls within the application domain of rule (75).

Rule (77) specifies that restrictions do not affect our transition relation directly. This is because, as will be made clear in the following, the information conveyed by restriction operators is already included in the intruder knowledge representation, which directly affects the possibility of transition occurrence.

Rules (78) and (79) control the occurrence of output and input transitions, respectively. They share the precondition $K \vdash \sigma$, i.e., they require that the channel identifier σ can be produced by the intruder. The complementary action label of output transitions, as indicated by the symbol $\delta_K(\rho)$ in (72), is the canonical repre-

sensation of a triple $\langle \delta_K^-(\rho), \delta_K^-(\rho), \rho \rangle$, which describes in an abstract and synthetic way how the intruder knowledge is affected by the reception of the data term ρ . The first component of the triple is defined as:

$$\delta_K^-(\rho) = \{ \langle \theta, \theta[K] \rangle \mid \theta \in \delta_{\bar{\Sigma}}^-(\rho) \} \text{ with } \bar{\Sigma} = \text{dom}(K) \quad (80)$$

It describes the terms that are eliminated from K in the transformation from K to K' when ρ is received. Each element of $\delta_K^-(\rho)$ is a pair $\langle \theta, \theta[K] \rangle$, where $\theta \in \delta_{\bar{\Sigma}}^-(\rho)$ is one of the terms removed from $\bar{\Sigma}$ in the reduction of $\bar{\Sigma} \cup \{\rho\}$, and $\theta[K]$ is the index assigned to it in K . The second component of the triple is:

$$\begin{aligned} \delta_K^-(\rho) = & \{ \langle \{[\sigma_1]\}_{\sigma_2^-}, \{[\sigma_1]\}_{\sigma_2^+} \rangle \mid \{[\sigma_1]\}_{\sigma_2^+} \in \delta_{\bar{\Sigma}}^-(\rho) \} \\ & \cup \{ \langle \{[\sigma_1]\}_{\sigma_2^+}, \{[\sigma_1]\}_{\sigma_2^-} \rangle \mid \{[\sigma_1]\}_{\sigma_2^-} \in \delta_{\bar{\Sigma}}^-(\rho) \} \end{aligned} \quad (81)$$

This is a set of pairs describing the terms $\theta \in \delta_{\bar{\Sigma}}^-(\rho)$, i.e., the terms that become decipherable after ρ has been received without being eliminated from the intruder knowledge domain. Each pair includes θ itself (the second component), and the term that is obtained from θ replacing the encryption key with the corresponding decryption key, which is a term that can be built from the elements of $\bar{\Sigma} \cup \{\rho\}$. Therefore, the pair $\langle \{[\sigma_1]\}_{\sigma_2^-}, \{[\sigma_1]\}_{\sigma_2^+} \rangle \in \delta_K^-(\rho)$ means that the intruder becomes able to extract σ_1 from $\{[\sigma_1]\}_{\sigma_2^+}$ using σ_2^- , but is not able to re-build $\{[\sigma_1]\}_{\sigma_2^+}$. The third component of the triple is the received data ρ .

Note that, because the triple $\langle \delta_K^-(\rho), \delta_K^-(\rho), \rho \rangle$ is finally interpreted with respect to K' (i.e., $\delta_K(\rho) = \langle \delta_K^-(\rho), \delta_K^-(\rho), \rho \rangle [K'] = \langle \delta_K^-(\rho)[K'], \delta_K^-(\rho)[K'], \rho[K'] \rangle$), each element of $\delta_K^-(\rho)[K']$ takes the form $(\theta[K'], \theta[K][K']) = (\theta[K'], \theta[K])$, i.e., it is composed of the interpretation of a term $\theta \in \delta_{\bar{\Sigma}}^-(\rho)$ with respect to the new and old intruder knowledge functions, respectively. Note also that the environment label of output transitions does not include an explicit canonical representation of the elements of $\delta_{\bar{\Sigma}}^-(\rho)$, because they are represented by indexes that already occur inside the canonical representations of the elements of $\delta_{\bar{\Sigma}}^-(\rho)$, $\delta_{\bar{\Sigma}}^-(\rho)$, and ρ .

As an example, consider the state $K \triangleright P$ with $P \triangleq (\nu k_3)(\bar{c}(k_3^-).c(x).Q)$, and K defined as in (69) (with $\text{dom}(K) = \bar{\Sigma}$ defined as in Figure 7). In this case we have $K \vdash c$, therefore enabling the output transition (rule (78)). It also follows that $K' = f(k_3^-, K)$ is that defined in (70), and $\delta_{\bar{\Sigma}}^-(k_3^-)$ and $\delta_{\bar{\Sigma}}^-(k_3^-)$ are as in Figure 7. Therefore, the following output transition can occur:

$$K \triangleright (\nu k_3)(\bar{c}(k_3^-).c(x).Q) \xrightarrow[\langle \delta_K^-(k_3^-)[K'], \delta_K^-(k_3^-)[K'], k_3^-[K'] \rangle]{\bar{l}_0} K' \triangleright (\nu k_3)(c(x).Q)$$

where $\delta_K^-(k_3^-)[K'] = \{ \langle \{[l_7]\}_{l_6^+}, l_2 \rangle, \langle l_6^+, l_3 \rangle \}$, $\delta_K^-(k_3^-)[K'] = \{ \langle \{[l_6]\}_{l_4}\}_{l_5}, l_1 \rangle \}$, and $k_3^-[K'] = l_5$.

Because $K' \vdash c$, with $c[K'] = l_0$, an input transition can occur after the output transition, according to rule (79). If we assume that the message ρ built by the intruder is $\{(m, k_3^-)\}_{k_2}$, which satisfies $K' \vdash \rho$ with $\{(m, k_3^-)\}_{k_2}[K'] = \{(l_7, l_5)\}_{l_4}$, then the input transition is:

$$K' \triangleright (\nu k_3)(c(x).Q) \xrightarrow[\{(l_7, l_5)\}_{l_4}]{l_0} K' \triangleright (\nu k_3)Q[\{(m, k_3^-)\}_{k_2}/x]$$

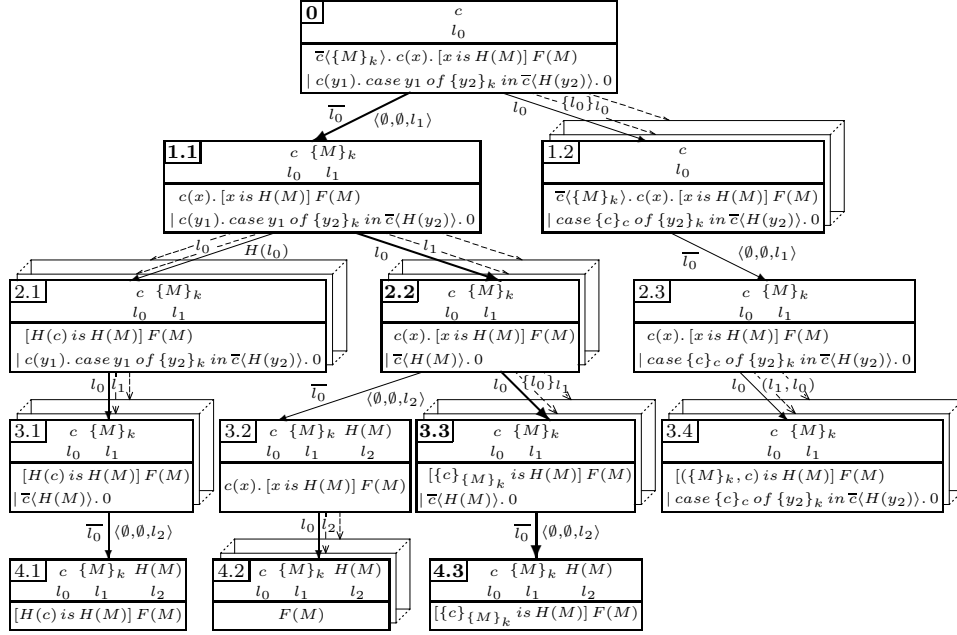


Fig. 8. A simple protocol: concrete ES-LTS with states and transitions

As a further example, consider again the specification of Figure 2: Figure 8 presents part of the corresponding concrete ES-LTS where reaction transitions are not represented, and, for each infinite set of input transitions, only one is fully represented with its subsequent behavior. The presence of other non-represented input transitions is sketched by means of dashed arrows leading to semi-hidden states. Each state box reports the intruder knowledge (upper half) and the process expression (lower half), and each arrow between two states is labeled with the process action label on the left and with the environment complementary action label on the right hand side. The initial intruder knowledge is assumed to be $K = \{\langle c, l_0 \rangle\}$. Analyze, for example, the run highlighted with bold state numbers and thick arrows. Initially (in state 0), the intruder knows only channel c , which is labeled l_0 , and the process is ready for two events: an input and an output on channel c . The output of $\{M\}_k$ leads to state 1.1, where the knowledge function is properly updated. The transition is labeled with the index of the output channel ($\overline{l_0}$) and an output environment label having $\delta_K^-(\rho)[K'] = \delta_K^-(\rho)[K'] = \emptyset$ and $\rho[K'] = l_1$ (because $\rho = \{M\}_k$). Subsequently, state 2.2 is reached by means of the input of term $\{M\}_k$ on channel c . An infinite set of other possible input transitions on c is possible here. The process action label of the transition analyzed is the index of the input channel (l_0), and the environment label is the canonical representation of $\{M\}_k$ with respect to the current knowledge function (l_1). Note that, because x is replaced by $\{M\}_k$, the internal operation $case\ x\ of\ \{y\}_k\ in \dots$ is successful. Consequently, both a new infinite set of input transitions and an output transition are enabled. The next step on the highlighted path is the input transition that

corresponds to the input value $\{c\}_{\{M\}_k} (\{l_0\}_{l_1})$, which leads to state 3.3. Because the behavior $[\{c\}_{\{M\}_k} \text{ is } H(M)]F(M)$ is stuck, only the output of $H(M)$ is possible, which leads to a state where the whole process is stuck.

3.4 Symbolic data representations

The symbolic ES-LTS is characterized by symbolic states and transitions, each of which actually represents a potentially infinite set of corresponding concrete states and transitions. Such symbolic representations are all based on *generic terms*, which are the symbolic data representations. When a spi calculus process is ready to perform an input operation, the intruder can interact with it by sending any term that can be built from its current knowledge. As already noted, the set of such terms is infinite, and the concrete ES-LTS model contains an infinite number of transitions, one for each different term that the intruder can send. In the symbolic ES-LTS model, such a set of transitions is represented as a single transition, where the data exchanged is a *generic term*, symbolically representing all the terms that could be passed to the spi process by the attacker.

To introduce generic terms in spi calculus, we extend the language with an additional infinite and countable set of names Γ , such that $\Gamma \cap \mathcal{A} = \emptyset$ and $\Gamma \cap I = \emptyset$. In the remainder of this paper, γ ranges over Γ . Each name $\gamma \in \Gamma$ uniquely identifies an *unspecialized generic term*, i.e., a term generated by the intruder and not constrained to assume any specific syntactical form. The concrete terms represented by γ are all those that the intruder was able to generate when γ was issued. It is worth noting that generic terms are not intended to be used in specifications, but only as internal abstract representations, useful in the verification algorithm.

By combining unspecialized generic terms using term operators, it is possible to build structured generic terms called *specialized generic terms*. They are generic but constrained to take a given form. For example, $\{m\}_\gamma$ is a generic term that is constrained to represent only cyphertexts obtained by encrypting m with the keys represented by γ . In general, a generic term is a spi calculus term $\eta \in \mathcal{M}(\mathcal{A} \cup I \cup \Gamma)$ that has at least a subterm $\gamma \in \Gamma$. Similarly, by using generic terms inside processes or knowledge functions, we obtain symbolic processes and symbolic knowledge functions. For example, $P = \bar{c}.\langle \{\gamma_1\}_{\gamma_2} \rangle.0$ is a symbolic process representing a set of concrete processes, one for each different pair of concrete terms represented by γ_1 and γ_2 . Note that while a symbolic process is meaningful with any generic term interpretation, a symbolic knowledge function has a meaning only if all the concrete values represented by its generic terms yield valid concrete knowledge functions. For example, the symbolic knowledge function $K = \{(m_1, l_0), (\{\gamma\}_{m_2}, l_1), (\{m_1\}_{m_2}, l_2)\}$ has a meaning only if γ cannot represent the concrete term m_1 , otherwise we would have a concrete $K = \{(m_1, l_0), (\{m_1\}_{m_2}, l_1), (\{m_1\}_{m_2}, l_2)\}$, which is not a function because it maps the same term $\{m_1\}_{m_2}$ onto two different indexes. Below, in this section, the means to ensure that all the symbolic functions occurring in a symbolic ES-LTS are meaningful is explained.

Each symbolic ES-LTS state is characterized by a (finite) current set of unspecialized generic terms $G \subset \Gamma$. They are the only unspecialized generic terms that can occur in the current symbolic state. Consequently, only generic terms belonging to $\mathcal{M}(\mathcal{A} \cup I \cup G)$ can occur in the current symbolic state. A function $\Upsilon : G \rightarrow 2^{\mathcal{M}(\mathcal{A} \cup I \cup G)}$, which is part of the symbolic ES-LTS state, gives the current

interpretation of the unspecialized generic terms. Each $\gamma \in G$ is mapped onto a corresponding knowledge function domain $\Upsilon(\gamma)$, which represents the minimal closure seed of the set of terms that was available to the intruder when γ was generated. The notation η_Υ indicates a generic term η that must be interpreted according to Υ . The subscript is used only when the specification of Υ is not obvious.

When the term represented by γ is generated by the intruder it can take the whole set of values $\widehat{\Upsilon(\gamma)}$, and a different subsequent behavior is possible for each of these. However, for the purpose of testing equivalence, such behaviors are indistinguishable from one another, and can be represented as a single symbolic behavior until the occurrence of some operation is conditioned by the value that was initially exchanged. Whenever this happens, only the behaviors corresponding to values that satisfy the condition are allowed to proceed and this is symbolically described by narrowing the set of terms represented by each unspecialized generic term γ down to the largest subset of $\widehat{\Upsilon(\gamma)}$ compatible with the operations performed. In most cases, a narrowing of this kind is equivalent to the substitution of one or more unspecialized generic terms with corresponding specialized generic terms or concrete terms. For example, a pair-splitting operation on a generic term γ narrows the set of concrete terms represented by γ so as to include only pairs of elements of $\widehat{\Upsilon(\gamma)}$. This is equivalent to applying the substitution $\langle(\gamma', \gamma'')/\gamma\rangle$, where γ' and γ'' are two new unspecialized generic terms representing the two components of the pairs. Naturally, new generic terms γ' and γ'' must be assigned the same knowledge function of γ . Similarly, if γ is a generic term such that $\Upsilon(\gamma) \vdash m$, the match operation $[m \text{ is } \gamma]$ narrows $\widehat{\Upsilon(\gamma)}$ down to $\{m\}$, which is represented by the substitution $\langle m/\gamma\rangle$. Substitutions such as those above are called *specializations*, because they substitute unspecialized generic terms with corresponding specialized or concrete terms. Specializations are formally defined by substitution lists taking the form $\langle\eta_1/\gamma_1, \dots, \eta_n/\gamma_n\rangle$, where, for each $i, j \in \{1, \dots, n\}$,

$$\gamma_i \in \text{dom}(\Upsilon) \quad , \quad \Upsilon(\gamma_i) \vdash \eta_i \quad , \quad \gamma_i \not\preceq \eta_j \quad (82)$$

The requirement $\gamma_i \not\preceq \eta_j$ means that specializations completely eliminate generic terms $\gamma_1, \dots, \gamma_n$. In the remainder of this paper, ξ ranges over specializations, and Δ over sets of specializations. The (possibly empty) set of new unspecialized generic terms introduced by a specialization is defined as $\text{new}_\Upsilon(\langle\eta_1/\gamma_1 \dots \eta_n/\gamma_n\rangle) = \{\gamma' \in \Gamma \setminus \text{dom}(\Upsilon) \mid \gamma' \preceq \eta_i \text{ for some } i \in \{1, \dots, n\}\}$, whereas the set of terms replaced by a specialization is defined as $\text{old}_\Upsilon(\langle\eta_1/\gamma_1 \dots \eta_n/\gamma_n\rangle) = \{\gamma_1, \dots, \gamma_n\}$. The null substitution \top is also a specialization, with $\text{new}_\Upsilon(\top) = \text{old}_\Upsilon(\top) = \emptyset$. When a specialization ξ is applied to a symbolic state, function Υ is updated accordingly, because some old unspecialized generic terms are eliminated from the state while one or more new unspecialized generic terms may be introduced. The new generic term interpretation function is denoted by $\Upsilon\{\xi\}$. Its new domain is $\text{dom}(\Upsilon\{\xi\}) = \text{dom}(\Upsilon) \cup \text{new}_\Upsilon(\xi) \setminus \text{old}_\Upsilon(\xi)$. For $\gamma \in \text{dom}(\Upsilon) \setminus \text{old}_\Upsilon(\xi)$, we have $\Upsilon\{\xi\}(\gamma) = \Upsilon(\gamma)[\xi]$, whereas for $\gamma \in \text{new}_\Upsilon(\xi)$, $\Upsilon\{\xi\}(\gamma)$ is the largest minimal closure seed compatible with (82). For example, if $\xi = \langle(\gamma', \gamma'')/\gamma\rangle$ is the specialization deriving from a pair splitting on γ , we have: $\text{new}_\Upsilon(\xi) = \{\gamma', \gamma''\}$, $\text{old}_\Upsilon(\xi) = \{\gamma\}$, and $\Upsilon\{\xi\}(\gamma') = \Upsilon\{\xi\}(\gamma'') = \Upsilon(\gamma)$. If instead $\xi = \langle\gamma_1/\gamma_2\rangle$ is the specialization derived from a match $[\gamma_1 \text{ is } \gamma_2]$, we have $\text{new}_\Upsilon(\xi) = \emptyset$, and $\text{old}_\Upsilon(\xi) = \{\gamma_2\}$. It can be realized that

the knowledge function associated with each new generic term is always one of the knowledge functions associated with the substituted generic terms.

The set of all possible specializations can change from state to state, and, as it depends only on Υ , it is denoted S_Υ . Two elements of S_Υ that are equal up to a renaming of new unspecialized generic terms must be considered indistinguishable. Therefore, in the remainder of the paper, equality between specializations is intended in this broader sense. Moreover, as the set of terms that can occur in a symbolic state is $\mathcal{M}(\mathcal{A} \cup I \cup \text{dom}(\Upsilon))$, each $\xi \in S_\Upsilon$ can be regarded as a mapping from $\mathcal{M}(\mathcal{A} \cup I \cup \text{dom}(\Upsilon))$ to the new set of terms $\mathcal{M}(\mathcal{A} \cup I \cup \text{dom}(\Upsilon\{\xi\}))$.

It is possible to introduce a partial order \subset on S_Υ , such that $\xi_1 \subset \xi_2$ if ξ_1 is more restrictive than ξ_2 . Formally, $\xi_1 \subset \xi_2$ if and only if a non-null generic term specialization $\xi \in S_{\Upsilon\{\xi_2\}}$ exists such that $\xi_1 = \xi_2\xi$. In other words, $\xi_1 \subset \xi_2$ means that ξ_1 can be obtained as the composition of ξ_2 with a further non-null specialization ξ . We also say that ξ_1 can be *produced* by ξ_2 . If ξ_1 and ξ_2 are represented by finite substitution lists, it is possible to algorithmically decide if $\xi_1 \subset \xi_2$. For example, if $\gamma', \gamma'' \notin \text{dom}(\Upsilon)$ are new generic terms, $\langle\{\gamma', \gamma''\}_m/\gamma_1\rangle \subset \langle\{\gamma'_1\}_m/\gamma_1\rangle$ because a further specialization of γ'_1 in the right hand side can make the second specialization equal to the first, i.e., $\langle\{\gamma'_1\}_m/\gamma_1\rangle\langle(\gamma'_1, \gamma'_2)/\gamma'_1\rangle = \langle\{\gamma'_1, \gamma'_2\}_m/\gamma_1\rangle$, and, because equality of specializations is defined up to a renaming of new unspecialized generic terms, $\langle\{\gamma'_1, \gamma'_2\}_m/\gamma_1\rangle = \langle\{\gamma', \gamma''\}_m/\gamma_1\rangle$. Another example is $\langle\{\gamma'_1\}_m/\gamma_1, m/\gamma_2\rangle \subset \langle m/\gamma_2\rangle$, because $\langle m/\gamma_2\rangle\langle\{\gamma'_1\}_m/\gamma_1\rangle = \langle\{\gamma'_1\}_m/\gamma_1, m/\gamma_2\rangle$. The set S_Υ with the partial order \subset is a join-semilattice. This can be easily shown, because at least one upper bound exists for any two elements of S_Υ (\top is an upper bound for any pair of elements).

If $\Delta \subseteq S_\Upsilon$ is a set of specializations and $\xi \in S_\Upsilon$, we say that ξ can be *produced* by Δ , denoted $\xi \subset \Delta$ iff ξ can be produced by one of the elements of Δ , i.e., $\xi \subset \xi_i$ for some $\xi_i \in \Delta$.

The set of all concrete terms represented by a generic term η_Υ is the set of all concrete terms that can be obtained by applying any possible specialization to η , i.e., $\{\eta[\xi] \in \mathcal{M}(\mathcal{A} \cup I) \mid \xi \in S_\Upsilon\}$. For example, if $\eta = \{\gamma\}_m$, then the specializations $\xi \in S_\Upsilon$ such that $\eta[\xi] \in \mathcal{M}(\mathcal{A} \cup I)$ are all those that substitute γ with a concrete term that can be produced by $\Upsilon(\gamma)$.

When a specialization ξ is applied, the set of concrete terms represented by any generic term η_Υ is narrowed. Moreover, if $\xi_1 \subseteq \xi_2$, the application of ξ_1 yields a set of concrete terms that is a subset of the set of terms originated by ξ_2 . Because a specialization represents a particular narrowing of the concrete values represented by generic terms, a set of specializations $\Delta \subseteq S_\Upsilon$ represents a family of such narrowings. A set $\Delta \subseteq S_\Upsilon$ is said to be *irredundant* if it contains only maximal elements, i.e., it does not contain any two elements ξ_1, ξ_2 such that $\xi_1 \subset \xi_2$. For example, if we have $\text{dom}(\Upsilon) = \{\gamma_1, \gamma_2\}$ and $\Upsilon(\gamma_1) = \Upsilon(\gamma_2) = \{m\}$, then the set $\Delta = \{\langle H(\gamma'_1)/\gamma_1, m/\gamma_2\rangle, \langle m/\gamma_2\rangle, \langle H(m)/\gamma_1, m/\gamma_2\rangle\}$ is redundant because $\langle H(m)/\gamma_1, m/\gamma_2\rangle \subset \langle H(\gamma'_1)/\gamma_1, m/\gamma_2\rangle \subset \langle m/\gamma_2\rangle$. Any redundant subset of S_Υ can be transformed into a corresponding irredundant subset by eliminating redundant (i.e., non-maximal) elements. For example, the irredundant set corresponding to the set that has just been introduced is $\Delta' = \{\langle m/\gamma_2\rangle\}$.

Naturally, the elimination of redundant elements from a set Δ does not change

the set of all the concrete terms that can be obtained by specializing η with all the specializations that can be produced by Δ . In the remainder of this paper only irredundant sets of specializations are considered.

Specializations are adequate to describe symbolic representation narrowings related to operations performed by the process, but they are not precise enough to describe narrowing deriving from operations on symbolic intruder knowledge. For example, consider the intruder knowledge

$$K = \{(c, l_0), (d, l_1), (\{(c, d)\}_{k_1}, l_2), (\{(\gamma_0, \gamma_1)\}_{k_1}\}_{k_2}, l_3)\} \quad (83)$$

with $\text{dom}(\Upsilon) = \{\gamma_0, \gamma_1\}$ and $\Upsilon(\gamma_0) = \Upsilon(\gamma_1) = \{c, d\}$, and process $P = (\nu k_2)\bar{c}\langle k_2 \rangle.P'$. It is clear that the output of k_2 allows the intruder to decode $\{(\gamma_0, \gamma_1)\}_{k_1}\}_{k_2}$, therefore obtaining $\{(\gamma_0, \gamma_1)\}_{k_1}$. Then, $\{(\gamma_0, \gamma_1)\}_{k_1}\}_{k_2}$ is removed from the knowledge, and two different final knowledge representations can be reached, depending on the concrete values considered for γ_0 and γ_1 . If we consider the behaviors where γ_0 and γ_1 are equal to c and d , respectively, only k_2 is added to $\text{dom}(K)$, because the message extracted from $\{(\gamma_0, \gamma_1)\}_{k_1}\}_{k_2}$ is already known to the intruder. If instead we consider the behaviors where γ_0 and γ_1 take other values, $\{(\gamma_0, \gamma_1)\}_{k_1}$ is a new data item for the intruder, and it is added to the intruder knowledge along with k_2 . In the symbolic ES-LTS, this situation is represented as having two symbolic transitions, leading respectively to the two knowledge representations

$$K'_1 = \{(c, l_0), (d, l_1), (\{(c, d)\}_{k_1}, l_2), (k_2, l_4)\} \quad (84)$$

$$K'_2 = \{(c, l_0), (d, l_1), (\{(c, d)\}_{k_1}, l_2), (k_2, l_4), (\{(\gamma_0, \gamma_1)\}_{k_1}, l_5)\} \quad (85)$$

If the first transition is taken, the sets of terms represented by γ_0 and γ_1 are narrowed by specialization $\langle c/\gamma_0, d/\gamma_1 \rangle$. If instead the second transition is taken, it is necessary to narrow the set of terms represented by γ_0 and γ_1 so as to exclude the fact that $(\gamma_0, \gamma_1) = (c, d)$. This ensures that the resulting knowledge K'_2 is a meaningful symbolic knowledge function. This second form of narrowing cannot be expressed by a specialization, but it can be described by specifying that some specializations will never be applied. Formally, we define an *extended narrowing specification* as a pair $\langle \xi, \delta_\Lambda \rangle$, where ξ is a specialization that must be applied, and $\delta_\Lambda = \{\xi_1, \dots, \xi_k\}$ is an irredundant set of forbidden further specializations. In the previous example, the second narrowing can be expressed by the extended narrowing specification $\langle \top, \{\langle c/\gamma_0, d/\gamma_1 \rangle\} \rangle$. Of course, $\langle \xi, \emptyset \rangle = \xi$.

By analogy with what has been done for simple specializations, it is also possible to introduce a partial order on the set of extended narrowings, therefore obtaining a join-semilattice, which extends S_Υ . As long as the computation proceeds, forbidden specializations are added, and we denote as Λ the irredundant set of specializations obtained as the union of all forbidden specializations accumulated up to the current state.

A specialization ξ is *compatible* with the prohibitions imposed by Λ if $\xi \not\in \Lambda$, i.e., at least one of the generic term assignments it yields cannot be produced by the elements of Λ . In other words, $\xi \not\in \Lambda$ means that there is at least one specialization that can be applied after ξ without violating the prohibitions imposed by Λ . For example, $\langle c/\gamma_0 \rangle$ is compatible with $\Lambda = \{\langle c/\gamma_0, d/\gamma_1 \rangle\}$, i.e., $\langle c/\gamma_0 \rangle \not\in \Lambda$, because

any further specialization of γ_1 different from $\langle d/\gamma_1 \rangle$, such as, for example, $H(d)/\gamma_1$, can be applied after $\langle c/\gamma_0 \rangle$ without violating the prohibitions imposed by Λ .

The set of specializations compatible with Λ is denoted $S_{\Upsilon, \Lambda} = \{\xi \in S_{\Upsilon} \mid \xi \not\subset \Lambda\}$. A generic term η to be interpreted according to Υ , but with the limitations imposed by Λ , is denoted $\eta_{\Upsilon, \Lambda}$.

Given two generic terms σ and ρ , we define their *unification*, denoted $\sigma \bullet \rho$, as the irredundant set of specializations compatible with Λ that make σ and ρ equal, i.e., $\sigma \bullet \rho = \{\xi \in S_{\Upsilon, \Lambda} \mid \sigma[\xi] = \rho[\xi]\}$. If σ and ρ are unconditionally equal (i.e., $\sigma = \rho$), $\sigma \bullet \rho = \{\top\}$, whereas if they cannot be equal, $\sigma \bullet \rho = \emptyset$. If σ , ρ , and Λ are finite, and $im(\Upsilon)$ includes only finite sets, then the computation of $\sigma \bullet \rho$ can be performed in a finite number of steps. As an example, compute $\gamma_0 \bullet \{\gamma_1\}_{\gamma_2}$, which is useful to specify which behaviors can successfully execute the match operation $[\gamma_0 \text{ is } \{\gamma_1\}_{\gamma_2}]$. Let $\Upsilon(\gamma_0) = \{\{m\}_d, \{m\}_e\}$, and $\Upsilon(\gamma_1) = \Upsilon(\gamma_2) = \{m, d, e\}$. It is clear that γ_0 needs to be specialized into a shared-key encrypted term, and the computation of $\gamma_0 \bullet \{\gamma_1\}_{\gamma_2}$ gives three possible specializations: $\xi_1 = \langle \{m\}_d/\gamma_0, m/\gamma_1, d/\gamma_2 \rangle$, $\xi_2 = \langle \{m\}_e/\gamma_0, m/\gamma_1, e/\gamma_2 \rangle$, and $\xi_3 = \langle \{\gamma'_0\}_{\gamma''_0}/\gamma_0, \gamma'_0/\gamma_1, \gamma''_0/\gamma_2 \rangle$, with $\Upsilon(\gamma'_0) = \Upsilon(\gamma''_0) = \Upsilon(\gamma_0)$. Note that the sets of concrete terms obtained from specializations ξ_1 , ξ_2 , and ξ_3 are all disjoint, because $m \notin \widehat{\Upsilon(\gamma'_0)}$ and $\{d, e\} \not\subset \widehat{\Upsilon(\gamma''_0)}$.

If a specialization $\xi \in S_{\Upsilon, \Lambda}$ is applied to a symbolic state, Λ is updated accordingly, and the resulting set is

$$\Lambda\{\xi\} = \{\xi' \in S_{\Upsilon\{\xi\}} \mid \xi\xi' \subset \Lambda\} \quad (86)$$

i.e., the new forbidden specializations ξ' are all those that, if applied after ξ , yield an overall specialization $\xi\xi'$ that is not compatible with Λ . If ξ and the forbidden specializations Λ are represented as finite substitution lists, $\Lambda\{\xi\}$ can be computed algorithmically. For example, the application of $\langle c/\gamma_0 \rangle$ in the example above (with $\Lambda = \{\langle c/\gamma_0, d/\gamma_1 \rangle\}$) leads to the new set of forbidden specializations $\Lambda' = \Lambda\{\langle c/\gamma_0 \rangle\} = \{\langle d/\gamma_1 \rangle\}$, and the further application of $\langle H(d)/\gamma_1 \rangle$ leads to $\Lambda'' = \Lambda'\{\langle H(d)/\gamma_1 \rangle\} = \emptyset$.

The notation introduced for generic terms can also be extended to symbolic knowledge functions and process descriptions. Specifically, $K_{\Upsilon, \Lambda}$, $P_{\Upsilon, \Lambda}$, and $(K \triangleright P)_{\Upsilon, \Lambda}$ are symbolic elements to be interpreted according to the pair (Υ, Λ) . If $\Upsilon = \emptyset$ and $\Lambda = \emptyset$, the current state is concrete, i.e., it does not contain generic terms.

We say that K is *consistent* with (Υ, Λ) if it includes only terms in $\mathcal{M}(\mathcal{A} \cup I \cup dom(\Upsilon))$ and, for each $\xi \in S_{\Upsilon, \Lambda}$, $K[\xi]$ is a knowledge function, i.e., it satisfies the properties of functions, it is bijective, and its domain is a minimal closure seed. Of course, $K_{\Upsilon, \Lambda}$ is a meaningful symbolic representation only if K is consistent with (Υ, Λ) , and such consistency is guaranteed if each transition that modifies the intruder knowledge updates Λ so as to include in it any specialization that invalidates the resulting symbolic intruder knowledge function. A process P is consistent with Υ, Λ if the terms it contains are in the set $\mathcal{M}(\mathcal{A} \cup I \cup dom(\Upsilon))$. In the remainder of the paper, the sets of all the symbolic knowledge functions and processes consistent with the interpretation given by (Υ, Λ) are denoted $\mathcal{K}_{\Upsilon, \Lambda}$ and $\mathcal{P}_{\Upsilon, \Lambda}$, respectively.

In conclusion, each symbolic ES-LTS state, denoted $(K \triangleright P)_{\Upsilon, \Lambda}$, is made up of

a current generic term interpretation Υ, Λ , a current symbolic knowledge function K consistent with it, and a current symbolic process behavior expression P that is also consistent with it.

3.5 The symbolic ES-LTS derivation system

Each symbolic transition represents an infinite set of corresponding concrete transitions, all of the same type. Therefore, symbolic transitions can be categorized into reaction, input, and output transitions, in the same way as concrete transitions. The labels of symbolic transitions are similar to the corresponding concrete labels, but they may contain generic terms. Moreover, complementary action labels may include an additional field that specifies a narrowing on the possible behaviors and can take either the form of a pure specialization ξ or the form of an extended narrowing specification $\langle \xi, \delta_\Lambda \rangle$.

Specifically, the general forms taken by symbolic transitions are:

$$(K \triangleright P)_{\Upsilon, \Lambda} \xrightarrow[\xi[K']]{\tau} (K' \triangleright P')_{\Upsilon', \Lambda'} \quad (87)$$

$$(K \triangleright P)_{\Upsilon, \Lambda} \xrightarrow[\langle \xi, \delta_\Lambda \rangle[K'], \delta_K(\rho)]{\sigma[\xi][K']} (K' \triangleright P')_{\Upsilon', \Lambda'} \quad (88)$$

$$(K \triangleright P)_{\Upsilon, \Lambda} \xrightarrow[\gamma]{\sigma[K]} (K \triangleright P')_{\Upsilon', \Lambda} \quad (89)$$

Reaction transitions (87) may contain narrowing specifications, which always take the form of pure specializations. A reaction transition that contains a non-null specialization is referred to as a *specialization transition*, and is associated with internal operations, such as pair splittings, comparisons, or decryptions, which allow only a restricted set of concrete behaviors to proceed. The execution of a specialization transition labeled by $\xi[K']$ can change any state component, the new resulting state being:

$$(K \triangleright P)_{\Upsilon, \Lambda} \{\xi\} = (K[\xi] \triangleright P[\xi])_{\Upsilon\{\xi\}, \Lambda\{\xi\}} = (K' \triangleright P')_{\Upsilon', \Lambda'} \quad (90)$$

The same state transformation also occurs in output transitions (88), where, however, the state is also subject to the knowledge update described by $\delta_K(\rho)$, and to the unallowed specializations update implied by δ_Λ .

It should be noted that the complementary action label of input transitions (89) is no longer a concrete term, but a new unspecialized generic term γ that represents any data term that the intruder can generate. Input transitions are never characterized by narrowing specifications.

The derivation system for the symbolic ES-LTS, by analogy with that for the concrete ES-LTS, includes all the original spi reaction relation definition axioms and rules. Although generic terms are new, with respect to the original spi calculus, we assume that the derivation rules as defined in [Abadi and Gordon 1999] apply to the extended set of names $\mathcal{A} \cup \Gamma$.

Following the analogy with the concrete ES-LTS, the symbolic derivation system includes the equivalent of rules (74)-(77), the only difference being the presence of the new state subscript Υ, Λ .

Rules (78) and (79) are substituted by their symbolic counterparts:

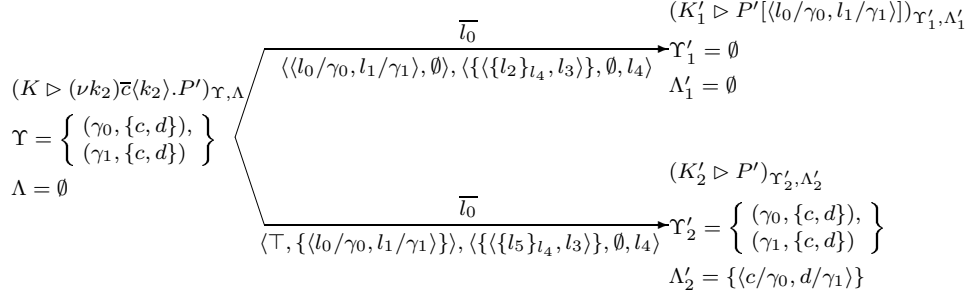


Fig. 9. An example with two symbolic transitions

$$\frac{K \vdash \sigma \quad \langle \xi, \delta_\Lambda \rangle \in \Theta(\rho, K_{\Upsilon, \Lambda}) \quad K'_{\Upsilon', \Lambda'} = f_{\langle \xi, \delta_\Lambda \rangle}(\rho, K_{\Upsilon, \Lambda})}{(K \triangleright \overline{\sigma}(\rho).P)_{\Upsilon, \Lambda} \xrightarrow[\langle \xi, \delta_\Lambda \rangle[K'], \langle \delta_K^-(\rho), \delta_K^-(\rho), \rho \rangle[K']]{\overline{\sigma[\xi][K']}} (K' \triangleright P[\xi])_{\Upsilon', \Lambda'}} \quad (91)$$

$$\frac{K \vdash \sigma \quad \gamma \notin \text{dom}(\Upsilon)}{(K \triangleright \sigma(x).P)_{\Upsilon, \Lambda} \xrightarrow[\gamma]{\overline{\sigma[K]}} (K \triangleright P[\gamma/x])_{\Upsilon \cup \{(\gamma, \text{dom}(K))\}, \Lambda}} \quad (92)$$

where

$$f_{\langle \xi, \delta_\Lambda \rangle}(\rho, K_{\Upsilon, \Lambda}) = f(\rho[\xi], K[\xi])_{\Upsilon\{\xi\}, (\Lambda \cup \delta_\Lambda)\{\xi\}} \quad (93)$$

$$\Theta(\rho, K_{\Upsilon, \Lambda}) = \{ \langle \xi, \delta_\Lambda \rangle \mid \xi \in S_{\Upsilon, \Lambda}, \delta_\Lambda \in S_\Upsilon, f_{\langle \xi, \delta_\Lambda \rangle}(\rho, K_{\Upsilon, \Lambda}) \in \mathcal{K}_{\Upsilon, \Lambda} \} \quad (94)$$

Function $f_{\langle \xi, \delta_\Lambda \rangle}$ is the symbolic version of function f after the application of the narrowing specified by $\langle \xi, \delta_\Lambda \rangle$. As seen in (93), this results from the composition of the knowledge transformation implied by the application of ξ (and described by (90)), the addition of δ_Λ to Λ , and the knowledge transformation described by function $f()$. $\Theta(\rho, K_{\Upsilon, \Lambda})$ is the irredundant set of narrowings $\langle \xi, \delta_\Lambda \rangle$ that make $f_{\langle \xi, \delta_\Lambda \rangle}(\rho, K_{\Upsilon, \Lambda})$ a valid knowledge function. Note that it is possible that the new knowledge function $K'_{\Upsilon', \Lambda'}$ reached after the output of ρ depends on how the generic terms are specialized. Therefore, there may be more than one possible $K'_{\Upsilon', \Lambda'}$, each corresponding to a different transition and to a different element of $\Theta(\rho, K_{\Upsilon, \Lambda})$. For example, if the intruder knowledge is that reported in (83) and $P = (\nu k_2)\overline{C}(k_2).P'$, we have already noted that there are two possible resulting symbolic knowledge functions ((84) and (85)). Therefore, in this example, $\Theta(\rho, K_{\Upsilon, \Lambda})$ includes two different extended narrowings. The first is $\langle \xi_1, \delta_{\Lambda_1} \rangle = (\langle c/\gamma_0, d/\gamma_1 \rangle, \emptyset)$, which leads to knowledge function (84), with $\Upsilon'_1 = \emptyset$ and $\Lambda'_1 = \emptyset$. In this case, the corresponding process action label is $\overline{\sigma[\xi_1][K'_1]} = \overline{l_0}$, and the two items of the complementary action label are $\langle \xi_1, \delta_{\Lambda_1} \rangle[K'_1] = (\langle l_0/\gamma_0, l_1/\gamma_1 \rangle, \emptyset)$ and $\langle \delta_K^-(\rho), \delta_K^-(\rho), \rho \rangle[K'_1] = \langle \{ \{l_2\}_{l_4}, l_3 \} \rangle, \emptyset, l_4$. This means that, after the arrival of k_2 , the piece of knowledge previously labeled l_3 can be seen as $\{l_2\}_{l_4}$. The second element of $\Theta(\rho, K_{\Upsilon, \Lambda})$ is $\langle \xi_2, \delta_{\Lambda_2} \rangle = (\emptyset, \{ \langle c/\gamma_0, d/\gamma_1 \rangle \})$, which leads to knowledge function (85), with $\Upsilon'_2 = \Upsilon$ and $\Lambda'_2 = \{ \langle c/\gamma_0, d/\gamma_1 \rangle \}$. In this case, the corresponding process action is again $\overline{\sigma[\xi_2][K'_2]} = \overline{l_0}$, whereas the two items of the complementary action label are $\langle \xi_2, \delta_{\Lambda_2} \rangle[K'_2] = (\emptyset, \{ \langle l_0/\gamma_0, l_1/\gamma_1 \rangle \})$ and $\langle \delta_K^-(\rho), \delta_K^-(\rho), \rho \rangle[K'_2] = \langle \{ \{l_5\}_{l_4}, l_3 \} \rangle, \emptyset, l_4$. This means that, after the arrival of k_2 , the piece of knowl-

edge named l_3 can be seen as $\{l_5\}_{l_4}$, where l_5 is a new piece of knowledge. The two complete resulting transitions and states are shown in Figure 9.

The semantics of specialization transitions are defined by the rules:

$$\frac{(K \triangleright P)_{\Upsilon, \Lambda} \xrightarrow[\xi[K[\xi]]]{\tau} (K \triangleright P')_{\Upsilon, \Lambda} \{\xi\}}{(K \triangleright (P|Q))_{\Upsilon, \Lambda} \xrightarrow[\xi[K[\xi]]]{\tau} (K \triangleright (P'|Q))_{\Upsilon, \Lambda} \{\xi\}} \quad (95)$$

$$\frac{\xi \in \sigma \bullet \rho \quad \xi \neq \top}{(K \triangleright (\bar{\sigma}(\theta).P \mid \rho(x).Q))_{\Upsilon, \Lambda} \xrightarrow[\xi[K[\xi]]]{\tau} (K \triangleright (\bar{\sigma}(\theta).P \mid \rho(x).Q))_{\Upsilon, \Lambda} \{\xi\}} \quad (96)$$

$$\frac{\xi \in \sigma \bullet \rho \quad \xi \neq \top}{(K \triangleright ([\sigma \text{ is } \rho]P))_{\Upsilon, \Lambda} \xrightarrow[\xi[K[\xi]]]{\tau} (K \triangleright [\sigma \text{ is } \rho]P)_{\Upsilon, \Lambda} \{\xi\}} \quad (97)$$

$$\frac{\gamma', \gamma'' \notin \text{dom}(\Upsilon)}{(K \triangleright (\text{let } (x, y) = \gamma \text{ in } P))_{\Upsilon, \Lambda} \xrightarrow[\gamma', \gamma''/\gamma]{\tau} (K \triangleright (\text{let } (x, y) = \gamma \text{ in } P))_{\Upsilon, \Lambda} \{(\gamma', \gamma'')/\gamma\}} \quad (98)$$

$$\frac{-}{(K \triangleright (\text{case } \gamma \text{ of } 0:P \text{ suc}(x):Q))_{\Upsilon, \Lambda} \xrightarrow[0/\gamma]{\tau} (K \triangleright (\text{case } \gamma \text{ of } 0:P \text{ suc}(x):Q))_{\Upsilon, \Lambda} \{0/\gamma\}} \quad (99)$$

$$\frac{\gamma' \notin \text{dom}(\Upsilon)}{(K \triangleright (\text{case } \gamma \text{ of } 0:P \text{ suc}(x):Q))_{\Upsilon, \Lambda} \xrightarrow[\text{suc}(\gamma')/\gamma]{\tau} (K \triangleright (\text{case } \gamma \text{ of } 0:P \text{ suc}(x):Q))_{\Upsilon, \Lambda} \{\text{suc}(\gamma')/\gamma\}} \quad (100)$$

$$\frac{\xi \in \eta \circ \rho \quad \xi \neq \top}{(K \triangleright (\text{case } \eta \text{ of } \{x\}_{\rho} \text{ in } P))_{\Upsilon, \Lambda} \xrightarrow[\xi[K[\xi]]]{\tau} (K \triangleright (\text{case } \eta \text{ of } \{x\}_{\rho} \text{ in } P))_{\Upsilon, \Lambda} \{\xi\}} \quad (101)$$

$$\frac{\xi \in \eta \oplus \rho \quad \xi \neq \top}{(K \triangleright (\text{case } \eta \text{ of } \{[x]\}_{\rho} \text{ in } P))_{\Upsilon, \Lambda} \xrightarrow[\xi[K[\xi]]]{\tau} (K \triangleright (\text{case } \eta \text{ of } \{[x]\}_{\rho} \text{ in } P))_{\Upsilon, \Lambda} \{\xi\}} \quad (102)$$

$$\frac{\xi \in \eta \ominus \rho \quad \xi \neq \top}{(K \triangleright (\text{case } \eta \text{ of } \{[x]\}_{\rho} \text{ in } P))_{\Upsilon, \Lambda} \xrightarrow[\xi[K[\xi]]]{\tau} (K \triangleright (\text{case } \eta \text{ of } \{[x]\}_{\rho} \text{ in } P))_{\Upsilon, \Lambda} \{\xi\}} \quad (103)$$

Rule (95) specifies how the parallel operator is treated, whereas the other rules specify all the situations in which specialization transitions can occur. Note that, according to (90), the new knowledge after specialization ξ is $K' = K[\xi]$, which means that $\xi[K'] = \xi[K[\xi]]$. Moreover, because $\xi \in S_{\Upsilon, \Lambda}$, this means that $\xi[K'] \in S_{\Upsilon[K'], \Lambda[K']}$, i.e., the canonical representation of a specialization ξ with respect to K' is also a specialization, according to the interpretation given by $\Upsilon[K']$ and $\Lambda[K']$. Note also that specialization transitions do not execute a spi calculus process action but they only apply a specialization, which enables further evolutions according to the other rules. For example, rule (96) applies a specialization $\xi \in \sigma \bullet \rho$, which makes σ and ρ equal, therefore enabling an internal synchronization between $\bar{\sigma}(\theta).P$ and $\rho(x).Q$.

The operators \circ , \oplus , and \ominus denote unifications similar to \bullet . $\eta \circ \rho$ is the set of specializations that must be applied to η and ρ to make η a term encrypted under key ρ , whereas \oplus and \ominus are the public-key and private-key variants of \circ . The formal definition of \circ is: $\eta \circ \rho = \{\xi \in S_{\Upsilon, \Lambda} \mid \exists \sigma \mid \eta[\xi] = \{\sigma\}_{\rho[\xi]}\}$.

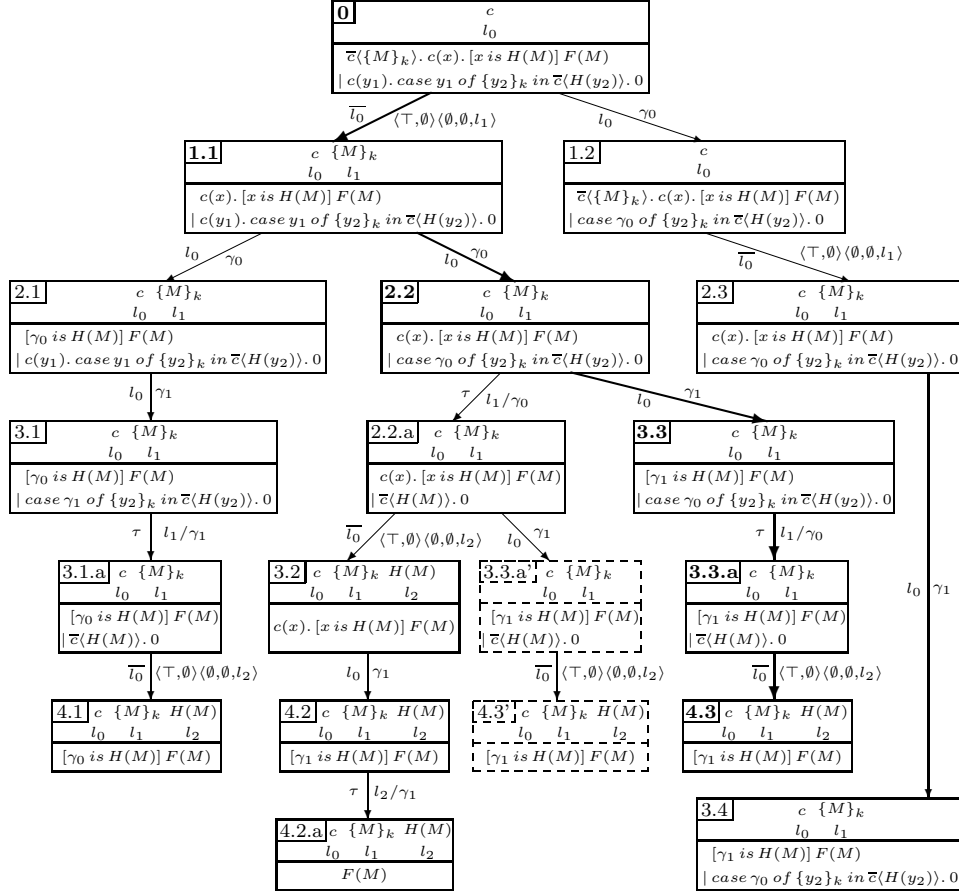


Fig. 10. A simple protocol: symbolic ES-LTS with states and transitions

Figure 10 shows the symbolic ES-LTS for the example of Figure 2, where, for simplicity, reaction transitions and their subsequent behaviors are omitted. The state numbering follows that of Figure 8, in the sense that the number associated with each symbolic state in Figure 10 is the same that in Figure 8 was assigned to the corresponding concrete state. However, because the symbolic ES-LTS also includes specialization transitions, some concrete states of Figure 8 are represented by more than one symbolic state in Figure 10. Different symbolic states representing the same concrete state of Figure 8 are distinguished by a letter, which is added after the state number. The states represented by dashed boxes come from the interleaving introduced by specialization transitions. The path starting from state 2.2 and ending with state 4.3' differs from that starting from 2.2 and ending with 4.3 only in the order of the first two transitions $((l_0, \gamma_1), (\tau, l_1/\gamma_0)$ vs. $(\tau, l_1/\gamma_0), (l_0, \gamma_1)$.

The set of forbidden specializations Λ is not shown because it is always empty in this example, whereas Υ is not explicitly represented because it can be easily

deduced from the state where each generic term is generated.

Analyze the run highlighted with bold state numbers and thick arrows. The initial state (state 0) is the same as that in the concrete ES-LTS. The first transition is the output of $\{M\}_k$, which leads to state 1.1. This is a concrete transition, because, up to this state, no generic term has been introduced. The process action label is \overline{l}_0 , as in the concrete ES-LTS, whereas the environment action label is $\langle \xi, \delta_\Lambda \rangle \langle \delta_K^-(\rho), \delta_K^-(\rho), \rho \rangle [K'] = \langle \top, \emptyset \rangle \langle \emptyset, \emptyset, l_1 \rangle$, which corresponds to the concrete ES-LTS label $\langle \emptyset, \emptyset, l_1 \rangle$. Starting from this state, state 2.2 can be reached by means of an input transition. This is the first really symbolic transition, because it introduces generic term γ_0 , which symbolically represents any data that can be sent to the process. Naturally, $\Upsilon(\gamma_0) = \{c, \{M\}_k\}$ is the domain of the intruder knowledge associated with state 1.1. Starting from state 2.2, two transitions are possible, enabled by rules (92) and (101). The input transition leads to state 3.3. Here, the operation $[\gamma_1 \text{ is } H(M)]$ cannot be successfully executed, because $\gamma_1 \bullet H(M) = \emptyset$, because $\Upsilon(\gamma_1) = \{c, \{M\}_k\}$. On the other hand, $\{M\}_k$ belongs to $\Upsilon(\gamma_0)$, which allows the operation *case* γ_0 of $\{y_2\}_k$ in \dots to be executed successfully. Then, a specialization transition with labels τ and $\langle l_1/\gamma_0 \rangle$ is enabled, which leads to state 3.3.a. Finally, only an output transition is possible, leading to state 4.3. The transition label of such a transition has the same structure as the previously described output transition, where $\rho = H(M)$ (labeled with l_2) substitutes $\rho = \{M\}_k$ (labeled with l_1). At this point the behavior cannot evolve any more.

The behavior starting from state 2.2 and ending in state 4.2.a, where the process becomes ready to behave as $F(M)$, has more states because the output of $H(M)$ (l_2) precedes the input event, therefore making the match $[\gamma_1 \text{ is } H(M)]$ successful after specialization $\langle H(M)/\gamma_1 \rangle$.

3.6 Traces

A trace is a string of symbols representing a sequence of *observable* computation steps in an ES-LTS. It is possible to define concrete traces and symbolic traces, according to which ES-LTS is considered. In any case, trace symbols take the form (μ, ϕ) , where μ and ϕ are the process action and the complementary action labels of the executed transitions. In the concrete ES-LTS, all computation steps except internal steps (i.e., input and output transitions) are observable. In the symbolic ES-LTS, input and output transitions are always observable, and pure reactions are unobservable, whereas specialization transitions are considered observable only if followed by input or output transitions. If a specialization is not followed by an input or output operation, the intruder has no way of realizing that it has taken place, and this is the reason it is considered unobservable. For example, the ES-LTS of process $c(x).[x \text{ is } m].0$ with an initial intruder knowledge domain including c and m , gives an input transition followed by an unobservable specialization transition. This is consistent with the intruder being unable to distinguish the above process from process $c(x).0$.

In the following, if not otherwise specified, t and t' denote traces, ϵ denotes the empty trace, a and a' denote trace symbols, $t.t'$ denotes trace t concatenated with trace t' , and $a.t$ and $t.a$ denote the traces obtained prepending and appending symbol a to trace t , respectively. Finally, π denotes the process action label of an input or output transition.

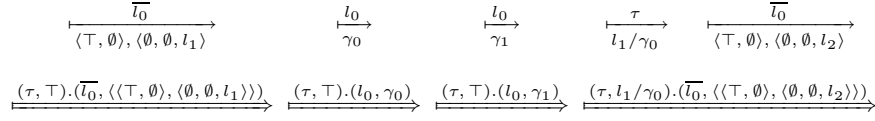


Fig. 11. An example of a symbolic trace

To formally define the traces of an ES-LTS, we introduce the binary relation \xRightarrow{t} on the set of states of an ES-LTS such that $S_1 \xRightarrow{t} S_2$ iff t is a trace starting from state S_1 and ending with state S_2 .

First consider the concrete ES-LTS. After defining $\xRightarrow{\epsilon}$ as the reflexive and transitive closure of $\xrightarrow{\tau}$, we inductively define $\xRightarrow{t, (\pi, \phi)}$ as $\xRightarrow{t} \xRightarrow{\epsilon} \xrightarrow{\pi/\phi} \xRightarrow{\epsilon}$.

For the symbolic ES-LTS, we have a similar inductive definition, where $\xRightarrow{\epsilon}$ is defined as for the concrete ES-LTS, and $(K \triangleright P)_{\Upsilon, \Lambda} \xRightarrow{t, (\tau, \xi), (\pi, \phi)} (K' \triangleright P')_{\Upsilon', \Lambda'}$ iff either $\xi = \top$ and $(K \triangleright P)_{\Upsilon, \Lambda} \xRightarrow{t} \xRightarrow{\epsilon} \xrightarrow{\pi/\phi} \xRightarrow{\epsilon} (K' \triangleright P')_{\Upsilon', \Lambda'}$, or there exist ξ_1, \dots, ξ_n such that $\xi = \xi_1 \dots \xi_n$, and $(K \triangleright P)_{\Upsilon, \Lambda} \xRightarrow{t} \xrightarrow{\tau/\xi_1} \dots \xrightarrow{\tau/\xi_n} \xrightarrow{\pi/\phi} \xRightarrow{\epsilon} (K' \triangleright P')_{\Upsilon', \Lambda'}$. This definition takes into account the fact that if two different sequences of specializations yield the same overall specialization then they must be considered indistinguishable. Moreover, each input/output symbol (π, ϕ) is preceded by a specialization symbol (τ, ξ) , describing the net effect of the (possibly empty) sequence of specialization transitions that are executed before the input/output transition. Specializations not followed by input/output operations are not considered because they are unobservable. Concrete traces can be regarded as symbolic traces where all the specializations and the extended narrowing specifications are null.

The concrete traces of a spi process P with an initial intruder knowledge K are defined as:

$$ctr(P, K) = \{t \mid \exists P', K' \mid K \triangleright P \xRightarrow{t} K' \triangleright P'\} \quad (104)$$

The most conservative assumption about the initial knowledge of the intruder is that the intruder initially knows any free name of P . In a similar manner, we define the symbolic traces of P :

$$str(P, K) = \{t \mid \exists P', K', \Upsilon', \Lambda' \mid (K \triangleright P)_{\emptyset, \emptyset} \xRightarrow{t} (K' \triangleright P')_{\Upsilon', \Lambda'}\} \quad (105)$$

Note that the initial state of a symbolic ES-LTS is concrete, and this is the reason initially $\Upsilon = \emptyset$ and $\Lambda = \emptyset$.

As an example, the symbolic trace corresponding to the thick-lined run of Figure 10 is shown in Figure 11, below the corresponding sequence of transitions.

Note that the null specialization symbol (τ, \top) has been added before those of the first three transitions, because they are not preceded by specialization transitions. For simplicity, in the remainder of the paper we omit null specializations and null narrowing specifications in symbolic traces.

The information on a symbolic trace t makes it possible to compute some elements of the ES-LTS state that is reached after the execution of t . Specifically, if $(K' \triangleright$

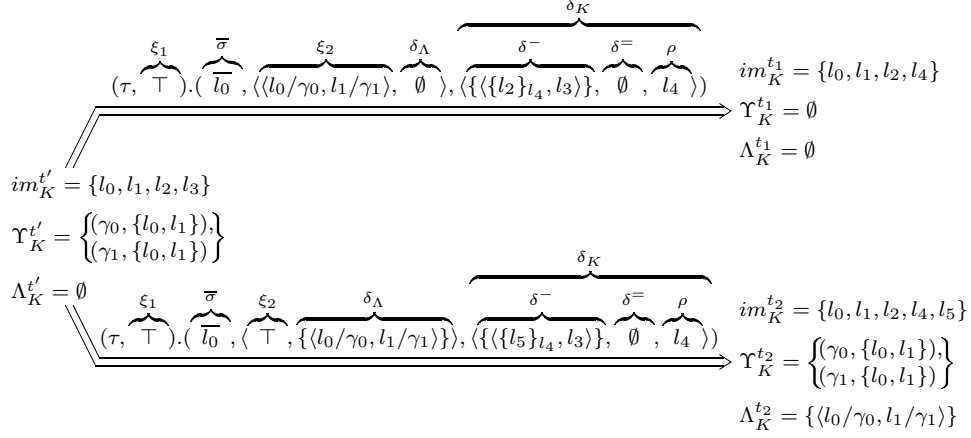


Fig. 12. An example with two symbolic traces

$P')_{\Upsilon', \Lambda'}$ is such a state and if the image of the initial intruder knowledge is known, it is possible to compute $im(K')$, $\Upsilon'[K']$, and $\Lambda'[K']$. Such elements are respectively denoted as im_K^t , Υ_K^t , and Λ_K^t , where K is the initial intruder knowledge. On the basis of the symbolic ES-LTS derivation rules, they can be computed as follows:

$$im_K^t = \begin{cases} im(K) & \text{if } t = \epsilon \\ im_K^{t'} & \text{if } t = t'.(\tau, \xi_1).(\sigma, \gamma) \\ im_K^{t'} \cup n(\delta_K) \setminus old(\delta_K) & \text{if } t = t'.(\tau, \xi_1).(\overline{\sigma}, \langle \xi_2, \delta_\Lambda \rangle, \delta_K) \end{cases} \quad (106)$$

$$\Upsilon_K^t = \begin{cases} \emptyset & \text{if } t = \epsilon \\ \Upsilon_K^{t'} \{ \xi_1 \} \cup \langle \gamma, im_K^{t'} \rangle & \text{if } t = t'.(\tau, \xi_1).(\sigma, \gamma) \\ \Upsilon_K^{t'} \{ \xi_1 \xi_2 \} [\lambda_{\delta_K}] & \text{if } t = t'.(\tau, \xi_1).(\overline{\sigma}, \langle \xi_2, \delta_\Lambda \rangle, \delta_K) \end{cases} \quad (107)$$

$$\Lambda_K^t = \begin{cases} \emptyset & \text{if } t = \epsilon \\ \Lambda_K^{t'} \{ \xi_1 \} & \text{if } t = t'.(\tau, \xi_1).(\sigma, \gamma) \\ \Lambda_K^{t'} \{ \xi_1 \xi_2 \} [\lambda_{\delta_K}] \cup \{ \delta_\Lambda \} & \text{if } t = t'.(\tau, \xi_1).(\overline{\sigma}, \langle \xi_2, \delta_\Lambda \rangle, \delta_K) \end{cases} \quad (108)$$

where $n(\delta_K)^2$ is the set of names occurring in δ_K , and, if $\delta_K = \langle \delta^- \delta^= \rho \rangle$, $old(\delta_K) = \{ \theta \mid (\theta', \theta) \in \delta^- \text{ for some } \theta' \}$ is the set of indexes removed from the intruder knowledge image in an output transition having environment label δ_K . Moreover, $\lambda_{\delta_K} = \{ \theta/\theta' \mid \langle \theta, \theta' \rangle \in \delta^- \}$ is the substitution that converts any canonical representation $\theta[K]$ into $\theta[K']$, where K and K' are the knowledge functions before and after an output transition with environment label δ_K , respectively. For example consider the traces $t_1 = t'.(\tau, \overline{\top}).(\overline{l_0}, \langle \langle l_0/\gamma_0, l_1/\gamma_1 \rangle, \emptyset \rangle, \langle \{ \{ l_2 \}_{l_4}, l_3 \} \rangle, \emptyset, l_4)$ and $t_2 = t'.(\tau, \overline{\top}).(\overline{l_0}, \langle \overline{\top}, \{ \langle l_0/\gamma_0, l_1/\gamma_1 \rangle \} \rangle, \langle \{ \{ l_5 \}_{l_4}, l_3 \} \rangle, \emptyset, l_4)$, derived from the transitions shown in Figure 9, where t' is the common prefix leading to state $(K \triangleright (\nu k_2) \overline{c} \langle k_2 \rangle . P')_{\Upsilon, \Lambda}$. Starting from $im_K^{t'}$, $\Upsilon_K^{t'}$, and $\Lambda_K^{t'}$, and applying (106), (107), and (108), $im_K^{t_1}$, $\Upsilon_K^{t_1}$, $\Lambda_K^{t_1}$ and $im_K^{t_2}$, $\Upsilon_K^{t_2}$, $\Lambda_K^{t_2}$ can be computed as shown in

²In the remainder of the paper, for simplicity, $\delta_K(\rho)$ is written δ_K .

Figure 12. To improve readability, each label of the two traces has been *tokenized* and related to each meaningful object of formulas (106), (107), and (108).

For the first trace, $n(\delta_K) = \{l_2, l_3, l_4\}$, $old(\delta_K) = \{l_3\}$ and $\lambda_{\delta_K} = \langle \{l_2\}_{l_4}/l_3 \rangle$. Consequently, $im_K^{t_1} = im_K^{t'_1} \cup \{l_2, l_3, l_4\} \setminus \{l_3\} = \{l_0, l_1, l_2, l_4\}$, $\Upsilon_K^{t_1} = \Upsilon_K^{t'_1} \{ \langle l_0/\gamma_0, l_1/\gamma_1 \rangle \} [\{l_2\}_{l_4}/l_3] = \emptyset$, and, because $\Lambda_K^{t'_1} = \emptyset$ and $\delta_\Lambda = \emptyset$, then $\Lambda_K^{t_1} = \emptyset$.

For the second trace, $n(\delta_K) = \{l_3, l_4, l_5\}$, $old(\delta_K) = \{l_3\}$ and $\lambda_{\delta_K} = \langle \{l_5\}_{l_4}/l_3 \rangle$. Then, $im_K^{t_2} = \{l_0, l_1, l_2, l_4, l_5\}$. In this case, $\xi_2 = \top$ and $\Upsilon_K^{t_2} = \Upsilon_K^{t'_2} \{ \top \} [\{l_5\}_{l_4}/l_3] = \Upsilon_K^{t'_2}$, because $\Upsilon_K^{t'_2}$ does not contain l_3 and remains unaffected by the substitution. Because $\Lambda_K^{t'_2} = \emptyset$, the only contribution to $\Lambda_K^{t_2}$ is given by $\delta_\Lambda = \{ \langle l_0/\gamma_0, l_1/\gamma_1 \rangle \}$, i.e., $\Lambda_K^{t_2} = \{ \langle l_0/\gamma_0, l_1/\gamma_1 \rangle \}$.

Each symbolic trace t with an initial intruder knowledge K represents a set of corresponding concrete traces, denoted $concr_K(t)$. To formally define this set, first define the subset of $S_{\Upsilon, \Lambda}$ that includes all the specializations that convert symbolic states into concrete states. Such a subset is formally defined as $S_{\Upsilon, \Lambda}^c = \{ \xi \in S_{\Upsilon, \Lambda} \mid \Upsilon\{\xi\} = \emptyset \}$, where it is worth noting that $\Upsilon\{\xi\} = \emptyset$ implies $\Lambda\{\xi\} = \emptyset$.

If $\xi \in S_{\Upsilon_K^{t'}, \Lambda_K^{t'}}^c$, then denote $act(t, \xi)$ as the concrete trace represented by t that is obtained by applying ξ to t . Formally, the inductive definition of $act(t, \xi)$ is:

$$act(t, \xi) = \begin{cases} \epsilon & \text{if } t = \epsilon \\ act(t', b_t(\xi)).a\{\xi\} & \text{if } t = t'.(\tau, \xi_1).a \end{cases} \quad (109)$$

where $a\{\xi\}$ is trace symbol a updated according to specialization ξ , i.e., $(\sigma, \gamma)\{\xi\} = (\sigma, \gamma)[\xi]$ and $(\bar{\sigma}, \langle \xi_2, \delta_\Lambda \rangle, \delta_K)\{\xi\} = (\bar{\sigma}[\xi], \langle \xi_2[\xi], \delta_\Lambda\{\xi\} \rangle, \delta_K[\xi])$, while, if $t = t'.(\tau, \xi_1)$, $a, b_t(\xi)$ is the specialization that must be applied to t' (i.e., to the prefix of t) in order to obtain the concrete trace. Naturally, $b_t(\xi)$ must incorporate ξ_1 , any specialization occurring in a , and ξ . Moreover, such specializations must be properly converted into valid elements of $S_{\Upsilon_K^{t'}, \Lambda_K^{t'}}^c$, as explained below. The formal definition of $b_t(\xi)$ is then:

$$b_t(\xi) = \begin{cases} (\xi_1\xi) \setminus dom(\Upsilon_K^{t'}) & \text{if } t = t'.(\tau, \xi_1).(\sigma, \gamma) \\ ((\xi_2\xi)[\lambda_{\delta_K}^{-1}[\xi]]) \setminus dom(\Upsilon_K^{t'}) & \text{if } t = t'.(\tau, \xi_1).(\bar{\sigma}, \langle \xi_2, \delta_\Lambda \rangle, \delta_K) \end{cases} \quad (110)$$

where, $\xi \setminus dom(\Upsilon_K^{t'})$ is ξ restricted to the set of generic terms $dom(\Upsilon_K^{t'})$, i.e., the specialization that substitutes each $\gamma \in dom(\Upsilon_K^{t'})$ with $\gamma[\xi]$. If $t = t'.(\tau, \xi_1).(\sigma, \gamma)$ and ξ is applied to t , t' is subject to ξ_1 followed by ξ , where such specializations are restricted to the domain of $\Upsilon_K^{t'}$. If instead $t = t'.(\tau, \xi_1).(\bar{\sigma}, \langle \xi_2, \delta_\Lambda \rangle, \delta_K)$, t' is subject to specializations ξ_1 , ξ_2 , and ξ . The additional transformation $[\lambda_{\delta_K}^{-1}[\xi]]$ applied to $\xi_2\xi$ is needed because ξ_2 and ξ are expressed as canonical representations with respect to the final intruder knowledge, whereas the specializations that are applied to t' must be expressed with respect to the knowledge the intruder had before the output statement. If we denote as K the knowledge before the output statement and as K' the final knowledge, the transformation that, for any θ , converts $\theta[K']$ into $\theta[K]$, is $\lambda_{\delta_K}^{-1}[\xi]$. Therefore, $(\xi_2\xi)[\lambda_{\delta_K}^{-1}[\xi]]$ is the same as $\xi_2\xi$, but expressed with respect to the knowledge that the intruder had before the output statement.

As an example, consider the symbolic trace t of Figure 11 which corresponds to the thick run of Figure 10. The left table of Figure 13 shows how im_K^t , Υ_K^t , and Λ_K^t are computed, whereas the right hand table shows how $act(t, \xi)$ is computed for $\xi = \langle \eta/\gamma_1' \rangle$, where η is any concrete term that can be built using l_0 and l_1 . It is

$$\underbrace{(\tau, \top) \cdot (\overline{l_0}, \langle \top, \emptyset \rangle, \langle \emptyset, \emptyset, l_1 \rangle)}_{t'''} \cdot (\tau, \top) \cdot (l_0, \gamma_0) \cdot (\tau, \top) \cdot (l_0, \gamma_1) \cdot (\tau, \langle l_1/\gamma_0 \rangle) \cdot (\overline{l_0}, \langle \top, \emptyset \rangle, \langle \emptyset, \emptyset, l_2 \rangle)$$

$$\underbrace{\hspace{10em}}_{t''}$$

$$\underbrace{\hspace{15em}}_{t'}$$

$$\underbrace{\hspace{20em}}_t$$

	im_K^t	Υ_K^t	Λ_K^t
ϵ	$\{l_0\}$	\emptyset	\emptyset
t'''	$\{l_0, l_1\}$	\emptyset	\emptyset
t''	$\{l_0, l_1\}$	$\{(\gamma_0, \{l_0, l_1\})\}$	\emptyset
t'	$\{l_0, l_1\}$	$\{(\gamma_0, \{l_0, l_1\}), (\gamma_1, \{l_0, l_1\})\}$	\emptyset
t	$\{l_0, l_1, l_2\}$	$\{(\gamma_1, \{l_0, l_1\})\}$	\emptyset

	ξ	$b_t(\xi)$	$act(t, \xi)$
ϵ	\top		ϵ
t'''	\top	\top	$\epsilon \cdot (\overline{l_0}, \langle \emptyset, \emptyset, l_1 \rangle)$
t''	$\langle l_1/\gamma_0 \rangle$	\top	$act(t''', \top) \cdot (l_0, l_1)$
t'	$\langle \eta/\gamma_1, l_1/\gamma_0 \rangle$	$\langle l_1/\gamma_0 \rangle$	$act(t'', \langle l_1/\gamma_0 \rangle) \cdot (l_0, \eta)$
t	$\langle \eta/\gamma_1 \rangle$	$\langle \eta/\gamma_1, l_1/\gamma_0 \rangle$	$act(t', \langle \eta/\gamma_1, l_1/\gamma_0 \rangle) \cdot (\overline{l_0}, \langle \emptyset, \emptyset, l_2 \rangle)$

$act(t, \eta/\gamma_1) = (\overline{l_0}, \langle \emptyset, \emptyset, l_1 \rangle) \cdot (l_0, l_1) \cdot (l_0, \eta) \cdot (\overline{l_0}, \langle \emptyset, \emptyset, l_2 \rangle)$

Fig. 13. An example of the computation of $act(t, \xi)$

worth noting that the computation of im_K^t , Υ_K^t , and Λ_K^t proceeds top-down in the table, whereas the computation of $act(t, \xi)$ proceeds backwards. Each allowable value of η gives a concrete trace of $concr_K(t)$: for example, $\eta = \{l_0\}_{l_1}$ gives the concrete trace corresponding to the run from state 0 to state 4.3 in Figure 8.

Having defined function $act()$, the concrete traces represented by a symbolic trace t that starts from an initial intruder knowledge K can be defined as:

$$concr_K(t) = \{act(t, \xi) \mid \xi \in S_{\Upsilon_K^t, \Lambda_K^t}^c\} \quad (111)$$

Function $concr_K()$ can be extended to sets of traces in the obvious manner: therefore, $concr_K(str(P, K))$ is the whole set of concrete traces represented by the symbolic traces of P .

Following the analogy with the other symbolic elements, it is possible to define a partial order on the set of the symbolic traces interpreted according to a given initial intruder knowledge K . Such a partial order is denoted \subseteq_K , and is defined in such a way that $concr_K(t_1) \subseteq concr_K(t_2)$ iff $t_1 \subseteq_K t_2$.

More precisely, the partial order \subseteq_K can be considered as a special case of a more general partial order that is denoted in the same way but is defined on the set of pairs $\langle t, \xi \rangle$, where t is a trace and $\xi \in S_{\Upsilon_K^t, \Lambda_K^t}$ is the environment label of a specialization that can be applied to t . The relationship $\langle t_1, \xi_1 \rangle \subseteq_K \langle t_2, \xi_2 \rangle$ means that all the concrete traces that are obtained from t_1 via specializations compatible with ξ_1 can also be obtained from t_2 via specializations compatible with ξ_2 . Then, $t_1 \subseteq_K t_2$ is defined as $\langle t_1, \top \rangle \subseteq_K \langle t_2, \top \rangle$.

The formal definition of $\langle t_1, \xi_1 \rangle \subseteq_K \langle t_2, \xi_2 \rangle$ is inductive. The base, which applies to traces of length 0, is the axiom:

$$\langle \epsilon, \top \rangle \subseteq_K \langle \epsilon, \top \rangle \quad (112)$$

while the induction step states that, for any $t_1 = t'_1.(\tau, \xi_1^1).a_1$ and $t_2 = t'_2.(\tau, \xi_2^2).a_2$,

$$\langle t_1, \xi_1 \rangle \subseteq_K \langle t_2, \xi_2 \rangle \iff \exists \xi \in S_{\Upsilon_K^{t_2}\{\xi_2\}, \Lambda_K^{t_2}\{\xi_2\}} \mid a_1\{\xi_1\} = a_2\{\xi_2\xi\} \wedge \langle t'_1, b_{t_1}(\xi_1) \rangle \subseteq_K \langle t'_2, b_{t_2}(\xi_2\xi) \rangle \quad (113)$$

As an example, let t_1 be the symbolic trace corresponding to the run from state 0 to state 4.3' in Figure 10, and let t_2 be the symbolic trace of Figure 11, which corresponds to the thick run of Figure 10. From the above definitions it follows that $\langle t_1, \top \rangle \subseteq_K \langle t_2, \top \rangle$, i.e., $t_1 \subseteq_K t_2$. This can be verified because, if t'_1 and t'_2 are the prefixes of t_1 and t_2 , respectively, then $\langle t_1, \top \rangle \subseteq_K \langle t_2, \top \rangle$ can be re-written as

$$\underbrace{\langle t'_1.(\tau, \underbrace{\xi_1^1}_{\sigma^1}).(\overbrace{(\overline{l_0}, \underbrace{\langle \top, \emptyset \rangle}_{\delta_\Lambda^1}, \underbrace{\langle \emptyset, \emptyset, l_2 \rangle}_{\delta_K^1})}_{a_1}), \underbrace{\top}_{\xi_1} \rangle}_{t_1} \subseteq_K \underbrace{\langle t'_2.(\tau, \underbrace{\langle l_1/\gamma_0 \rangle}_{\xi_1^2}).(\overbrace{(\overline{l_0}, \underbrace{\langle \top, \emptyset \rangle}_{\delta_\Lambda^2}, \underbrace{\langle \emptyset, \emptyset, l_2 \rangle}_{\delta_K^2})}_{a_2}), \underbrace{\top}_{\xi_2} \rangle}_{t_2}$$

Because $a_1 = a_2$ and $\xi_1 = \xi_2 = \top$, it follows that $\xi = \top$ trivially satisfies $a_1\{\xi_1\} = a_2\{\xi_2\xi\}$. Then, it remains to verify that $\langle t'_1, b_{t_1}(\xi_1) \rangle \subseteq_K \langle t'_2, b_{t_2}(\xi_2\xi) \rangle$. From (110), it transpires that $b_{t_1}(\xi_1) = (\xi_1^1((\xi_1^2\xi_1)[\lambda_{\delta_K^1}^{-1}[\xi_1]])) \setminus \text{dom}(\Upsilon_K^{t'_1}) = \top$, because $\delta_K^- = \emptyset$. Consequently, $[\lambda_{\delta_K^1}^{-1}] = \top$ and $\xi_1^1 = \xi_2^1 = \xi_1 = \top$. Similarly, we have $b_{t_2}(\xi_2\xi) = (\xi_1^2((\xi_2^2\xi_2\xi)[\lambda_{\delta_K^2}^{-1}[\xi_2\xi]])) \setminus \text{dom}(\Upsilon_K^{t'_2}) = \langle l_1/\gamma_0 \rangle$, because $\xi_1^2 = \langle l_1/\gamma_0 \rangle$. Then, the previous formula reduces to $\langle t'_1, \top \rangle \subseteq_K \langle t'_2, \langle l_1/\gamma_0 \rangle \rangle$, i.e., denoting the prefixes of t'_1 and t'_2 as t''_1 and t''_2 ,

$$\underbrace{\langle t''_1.(\tau, \underbrace{\langle l_1/\gamma_0 \rangle}_{\xi_1^1}).(\overbrace{(l_0, \gamma_1)}^{\sigma^1}), \underbrace{\top}_{\xi_1} \rangle}_{t'_1} \subseteq_K \underbrace{\langle t''_2.(\tau, \underbrace{\top}_{\xi_1^2}).(\overbrace{(l_0, \gamma_1)}^{\sigma^2}), \underbrace{\langle l_1/\gamma_0 \rangle}_{\xi_2} \rangle}_{t'_2}$$

The equation $a_1\{\xi_1\} = a_2\{\xi_2\xi\}$ can be written as $(l_0, \gamma_1)\{\top\} = (l_0, \gamma_1)\{\langle l_1/\gamma_0 \rangle\xi\}$, which, again, holds for $\xi = \top$. Moreover, $b_{t'_1}(\xi_1) = (\xi_1^1\xi_1) \setminus \text{dom}(\Upsilon_K^{t''_1}) = \langle l_1/\gamma_0 \rangle$, and $b_{t'_2}(\xi_2\xi) = (\xi_1^2\xi_2\xi) \setminus \text{dom}(\Upsilon_K^{t''_2}) = \langle l_1/\gamma_0 \rangle$. Then we recursively have to verify that $\langle t''_1, \langle l_1/\gamma_0 \rangle \rangle \subseteq_K \langle t''_2, \langle l_1/\gamma_0 \rangle \rangle$, but t''_1 and t''_2 are equal (as can be seen in Figure 10), and thus their preorder relationship is trivially verified. In a similar way it can be verified that $t_2 \subseteq_K t_1$ also holds.

The preorder \subseteq_K can be extended to sets of symbolic traces: if T_1 and T_2 are two sets of symbolic traces starting from the same initial knowledge K , then $T_1 \subseteq_K T_2$ iff $\forall t_1 \in T_1 \exists t_2 \in T_2 \mid t_1 \subseteq_K t_2$.

Trace equivalence on sets of symbolic traces is defined as the kernel of the preorder \subseteq_K , i.e., $\subseteq_K \cap \subseteq_K^{-1}$.

The last part of this section is devoted to proving the coincidence between trace equivalence defined over sets of concrete traces (which is simply defined as set equality) and trace equivalence defined over sets of symbolic traces. The proof requires some preliminary results.

The first preliminary result is that trace inclusion is equivalent to the inclusion of the corresponding sets of represented concrete traces, which is formalized in the following theorem.

THEOREM 3.8. *If t_1 and t_2 are symbolic traces, then $t_1 \subseteq_K t_2 \iff \text{concr}_K(t_1) \subseteq \text{concr}_K(t_2)$.*

PROOF. Theorem 3.8 is a special case of the following proposition, which can be proved by induction:

$$\begin{aligned} \langle t_1, \xi_1 \rangle \subseteq_K \langle t_2, \xi_2 \rangle &\iff \\ (\forall \xi_1^c \in S_{\Upsilon_K^{t_1}, \Lambda_K^{t_1}}^c \mid \xi_1^c \subseteq \xi_1) \exists \xi_2^c \in S_{\Upsilon_K^{t_2}, \Lambda_K^{t_2}}^c \mid \xi_2^c \subseteq \xi_2 \wedge \text{act}(t_1, \xi_1^c) &= \text{act}(t_2, \xi_2^c) \end{aligned} \quad (114)$$

The base (for t_1 and t_2 of length 0, i.e., $t_1 = t_2 = \epsilon$) is trivially true.

To prove the induction step, assume that (114) holds for traces of length n . Let t_1 and t_2 be any traces of length n , and let $t'_1 = t_1 \cdot (\tau, \xi_1^1).a_1$ and $t'_2 = t_2 \cdot (\tau, \xi_2^2).a_2$ be any traces of length $n+1$. We have to prove that (114) holds for t'_1 and t'_2 also. The two implications in (114) are proved separately.

First, assume that $\langle t'_1, \xi'_1 \rangle \subseteq_K \langle t'_2, \xi'_2 \rangle$, with $\xi'_1 \in S_{\Upsilon_K^{t'_1}, \Lambda_K^{t'_1}}$, and $\xi'_2 \in S_{\Upsilon_K^{t'_2}, \Lambda_K^{t'_2}}$, and let $\xi_1^{c'} \in S_{\Upsilon_K^{t'_1}, \Lambda_K^{t'_1}}^c$, with $\xi_1^{c'} \subseteq \xi'_1$.

By the initial assumption and by the definition of \subseteq_K , it follows that

$$\exists \xi \in S_{\Upsilon_K^{t'_2}, \Lambda_K^{t'_2}} \mid a_1\{\xi'_1\} = a_2\{\xi'_2\} \wedge \langle t_1, b_{t'_1}(\xi'_1) \rangle \subseteq_K \langle t_2, b_{t'_2}(\xi'_2) \rangle \quad (115)$$

Because t_1 and t_2 are of length n , equation (114) holds. In particular, considering that $\xi_1^{c'} \subseteq \xi'_1$ implies $b_{t'_1}(\xi_1^{c'}) \subseteq b_{t'_1}(\xi'_1)$, from (114) with $\xi_1^c = b_{t'_1}(\xi_1^{c'})$ it follows that

$$\exists \xi_2^c \in S_{\Upsilon_K^{t_2}, \Lambda_K^{t_2}}^c \mid \xi_2^c \subseteq b_{t'_2}(\xi'_2) \wedge \text{act}(t_1, \xi_1^c) = \text{act}(t_2, \xi_2^c) \quad (116)$$

Because $\xi_2^c \subseteq b_{t'_2}(\xi'_2)$, then a specialization ξ_2^* must exist so that $\xi_2^c = b_{t'_2}(\xi'_2 \xi_2^*)$. Let ξ_1^* be the specialization that substitutes only the new generic terms introduced in $(\tau, \xi_1^1).a_1$, and replaces them in the same way as $\xi_1^{c'}$. Then, taking $\xi_2^{c'} = \xi'_2 \xi_2^* \xi_1^*$, it follows that $\xi_2^{c'} \in S_{\Upsilon_K^{t'_2}, \Lambda_K^{t'_2}}^c$ and $b_{t'_2}(\xi_2^{c'}) = b_{t'_2}(\xi'_2 \xi_2^*)$ (because the names replaced by ξ_1^* are not included in $\text{dom}(\Upsilon_K^{t'_2})$). Using the previous results, we have that

$$\text{act}(t'_1, \xi_1^{c'}) = \text{act}(t_1, b_{t'_1}(\xi_1^{c'})).a_1\{\xi_1^{c'}\} = \text{act}(t_1, \xi_1^c).a_1\{\xi_1^{c'}\} \quad (117)$$

$$\text{act}(t'_2, \xi_2^{c'}) = \text{act}(t_2, b_{t'_2}(\xi_2^{c'})).a_2\{\xi_2^{c'}\} = \text{act}(t_2, \xi_2^c).a_1\{\xi'_1 \xi_2^* \xi_1^*\} \quad (118)$$

Because by (116) $\text{act}(t_1, \xi_1^c) = \text{act}(t_2, \xi_2^c)$, t_1 and t_2 must have the same structure and must contain the same generic terms. Moreover, all such generic terms are substituted in the same way by ξ_1^c and ξ_2^c . Consequently, they are also substituted in the same way by $\xi_1^{c'}$ and $\xi_2^{c'}$. Moreover, because ξ_1^* substitutes the new generic terms in the same way as by $\xi_1^{c'}$, it follows that $a_1\{\xi_1^{c'}\} = a_1\{\xi'_1 \xi_2^* \xi_1^*\}$, which means that the direct implication in (114) holds.

To prove the reverse implication of formula (114), the initial assumption is:

$$(\forall \xi_1^{c'} \in S_{\Upsilon_K^{t'_1}, \Lambda_K^{t'_1}}^c \mid \xi_1^{c'} \subseteq \xi_1' \mid \exists \xi_2^{c'} \in S_{\Upsilon_K^{t'_2}, \Lambda_K^{t'_2}}^c \mid \xi_2^{c'} \subseteq \xi_2' \wedge \text{act}(t'_1, \xi_1^{c'}) = \text{act}(t'_2, \xi_2^{c'})) \quad (119)$$

From the above assumption it follows that for any $\xi_1^{*'} \in S_{\Upsilon_K^{t'_1}, \Lambda_K^{t'_1}}^c$ such that $\xi_1' \xi_1^{*'} \in S_{\Upsilon_K^{t'_1}, \Lambda_K^{t'_1}}^c$, there exists a $\xi_2^{*'} \in S_{\Upsilon_K^{t'_2}, \Lambda_K^{t'_2}}^c$ such that $\xi_2' \xi_2^{*'} \in S_{\Upsilon_K^{t'_2}, \Lambda_K^{t'_2}}^c$ and $\text{act}(t'_1, \xi_1' \xi_1^{*'}) = \text{act}(t'_2, \xi_2' \xi_2^{*'})$. From the definition of $\text{act}()$, this implies that $a_1\{\xi_1' \xi_1^{*'}\} = a_2\{\xi_2' \xi_2^{*'}\}$. Because this holds for any $\xi_1^{*'}$, there must exist a specialization ξ^* such that $a_2\{\xi_2' \xi^*\} = a_1\{\xi_1'\}$.

To prove that $\langle t_1, \xi_1 \rangle \subseteq_K \langle t_2, \xi_2 \rangle$, we have to prove that $\langle t_1, b_{t'_1}(\xi_1') \rangle \subseteq_K \langle t_2, b_{t'_2}(\xi_2') \rangle$. Let $\xi_1^c \in S_{\Upsilon_K^{t_1}, \Lambda_K^{t_1}}^c$ with $\xi_1^c \subseteq b_{t'_1}(\xi_1')$, and let $\xi_1^{c'} \subseteq \xi_1^c$ be the specialization such that $b_{t'_1}(\xi_1^{c'}) = \xi_1^c$. Then our initial assumption (119) means that $\exists \xi_2^{c'} \in S_{\Upsilon_K^{t'_2}, \Lambda_K^{t'_2}}^c \mid \xi_2^{c'} \subseteq \xi_2' \wedge \text{act}(t'_1, \xi_1^{c'}) = \text{act}(t'_2, \xi_2^{c'})$. This implies that, taking $\xi_2^c = b_{t'_2}(\xi_2^{c'})$, $\text{act}(t_1, \xi_1^c) = \text{act}(t_2, \xi_2^c)$. Then, because (114) is true for traces of length n , we also have that $\langle t_1, b_{t'_1}(\xi_1') \rangle \subseteq_K \langle t_2, b_{t'_2}(\xi_2') \rangle$. \square

Before proving the final theorem, we need to formally prove that $\text{concr}_K(\text{str}(P, K))$ really computes the concrete traces of P , as previously claimed. This is expressed by the following lemma.

LEMMA 3.9. *For any knowledge function K and process P such that $\text{fn}(P) \subseteq \text{dom}(K)$, $\text{concr}_K(\text{str}(P, K)) = \text{ctr}(P, K)$.*

PROOF. We have to prove that, for each concrete trace t_c , $t_c \in \text{ctr}(P, K) \iff t_c \in \text{concr}_K(\text{str}(P, K))$. Because $t_c \in \text{ctr}(P, K)$ is equivalent to $\exists K_c, P_c \mid (K \triangleright P) \xrightarrow{t_c} (K_c \triangleright P_c)$, and $t_c \in \text{concr}_K(\text{str}(P, K))$ is equivalent to $\exists t \in \text{str}(P, K), \xi \mid t_c = \text{act}(t, \xi)$, which in turn is equivalent to $\exists K_s, P_s, \Upsilon, \Lambda, t, \xi \mid (K \triangleright P)_{\emptyset, \emptyset} \xrightarrow{t} (K_s \triangleright P_s)_{\Upsilon, \Lambda} \wedge t_c = \text{act}(t, \xi)$, then we finally have to prove that, for each concrete trace t_c ,

$$(\exists K_c, P_c \mid (K \triangleright P) \xrightarrow{t_c} (K_c \triangleright P_c)) \iff (\exists K_s, P_s, \Upsilon, \Lambda, t, \xi \mid (K \triangleright P)_{\emptyset, \emptyset} \xrightarrow{t} (K_s \triangleright P_s)_{\Upsilon, \Lambda} \wedge t_c = \text{act}(t, \xi)) \quad (120)$$

Because it is useful for the proof, we also claim that if the right and left hand sides of (120) are true, then

$$(K_s \triangleright P_s)_{\Upsilon, \Lambda} \{\xi[K_c^{-1}]\} = (K_c \triangleright P_c) \quad (121)$$

Now we prove by induction on concrete traces t_c that (120) and (121) hold. For $t_c = \epsilon$, both sides of (120) are trivially true with $\xi = \top$, and (121) is also trivially true. Then, we have to prove that (120) and (121) hold for any $t'_c.(\pi_c, \phi_c)$, assuming that they hold for t'_c , i.e., that:

$$(\exists K'_c, P'_c \mid (K \triangleright P) \xrightarrow{t'_c} (K'_c \triangleright P'_c)) \iff (\exists K'_s, P'_s, \Upsilon', \Lambda', t', \xi' \mid (K \triangleright P)_{\emptyset, \emptyset} \xrightarrow{t'} (K'_s \triangleright P'_s)_{\Upsilon', \Lambda'} \wedge t'_c = \text{act}(t', \xi')) \quad (122)$$

and that, if both sides of (122) are true, then

$$(K'_s \triangleright P'_s)_{\Upsilon', \Lambda'} \{\xi' [K_c'^{-1}]\} = (K'_c \triangleright P'_c) \quad (123)$$

When both sides of (122) are false, it also results in both sides of (120) being false, and then (120) holds. Consider now the case in which both sides of (122) are true. Because by (123) $(K'_c \triangleright P'_c)$ is a specialization of symbolic state $(K'_s \triangleright P'_s)_{\Upsilon', \Lambda'}$, it can be verified by inspection that any axiom or rule of the concrete semantics can be applied to state $(K'_c \triangleright P'_c)$ iff a corresponding axiom or rule of the symbolic semantics can be applied either to state $(K'_s \triangleright P'_s)_{\Upsilon', \Lambda'}$ or to a state that is reached from $(K'_s \triangleright P'_s)_{\Upsilon', \Lambda'}$ after one or more applications of rules (95)-(103). Therefore, it follows that for any concrete trace symbol (π_c, ϕ_c) ,

$$\begin{aligned} \exists K_c, P_c \mid (K'_c \triangleright P'_c) &\xrightarrow{(\pi_c, \phi_c)} (K_c \triangleright P_c) \iff \\ \exists K_s, P_s, \Upsilon, \Lambda, \xi_1, \pi_s, \phi_s \mid (K'_s \triangleright P'_s)_{\Upsilon', \Lambda'} &\xrightarrow{(\tau, \xi_1) \cdot (\pi_s, \phi_s)} (K_s \triangleright P_s)_{\Upsilon, \Lambda} \end{aligned} \quad (124)$$

where (π_c, ϕ_c) and (π_s, ϕ_s) derive from the application of corresponding input or output rules and ξ_1 must be compatible with ξ' (i.e., $\xi_1 \subseteq \xi'$). This means that an intermediate symbolic state $(K''_s \triangleright P''_s)_{\Upsilon'', \Lambda''}$ exists that is reached from $(K'_s \triangleright P'_s)_{\Upsilon', \Lambda'}$ after the application of specialization ξ_1 . If ξ_0 is the specialization such that $\xi_1 \xi_0 = \xi'$, the state reached from $(K''_s \triangleright P''_s)_{\Upsilon'', \Lambda''}$ applying ξ_0 is the same as that reached from $(K'_s \triangleright P'_s)_{\Upsilon', \Lambda'}$ applying $\xi_1 \xi_0$, i.e., $(K''_s \triangleright P''_s)_{\Upsilon'', \Lambda''} \{\xi_0 [K_c'^{-1}]\} = (K'_s \triangleright P'_s)_{\Upsilon', \Lambda'} \{\xi_1 \xi_0 [K_c'^{-1}]\} = (K'_c \triangleright P'_c)$. As a consequence, if $\pi_s = \sigma_s[K_s'']$ and $\pi_c = \sigma_c[K_c']$ (or $\pi_s = \overline{\sigma}_s[K_s'']$ and $\pi_c = \overline{\sigma}_c[K_c']$), then $\sigma_c = \sigma_s[\xi_0 [K_c'^{-1}]]$.

We must now distinguish the input and output cases. If (π_c, ϕ_c) is an input, then by the concrete semantics we have that $K'_c = K_c$, $K''_s = K_s$, and $(\pi_c, \phi_c) = (\sigma_c[K_c'], \rho_c[K_c'])$, and it also follows that $(\pi_s, \phi_s) = (\sigma_s[K_s''], \gamma)$. Now, we claim that $t'_c \cdot (\pi_c, \phi_c) = \text{act}(t' \cdot (\tau, \xi_1) \cdot (\pi_s, \phi_s), \xi_0 \langle \rho / \gamma \rangle)$. Putting together (122), (124), and this claim, we can conclude that (120) holds for $t_c = t'_c \cdot (\pi_c, \phi_c)$. To prove that the claim is true, by the definition of $\text{act}()$ we have to prove the equality $\sigma_s[K_s''][\xi_0] = \sigma_c[K_c']$, which, after the substitution $\sigma_c = \sigma_s[\xi_0 [K_c'^{-1}]]$, can be rewritten as

$$\sigma_s[K_s''][\xi_0] = \sigma_s[\xi_0 [K_c'^{-1}]] [K_c'] \quad (125)$$

To prove this, first consider the cases $\sigma_s \in \text{dom}(K_s'')$ and $\sigma_s \in \Gamma$. If $\sigma_s \in \text{dom}(K_s'')$, and $\sigma_s[K_s'']$ is an index, then the left hand side of (125) is $\sigma_s[K_s''][\xi_0] = \sigma_s[K_s'']$. Moreover, it follows that $\sigma_s[\xi_0 [K_c'^{-1}]] \in \text{dom}(K_s''[\xi_0 [K_c'^{-1}]])$, and $\sigma_s[\xi_0 [K_c'^{-1}]]$ is mapped by $K_s''[\xi_0 [K_c'^{-1}]] = K'_c$ in exactly the same way as σ_s is mapped by K_s'' , i.e., $\sigma_s[\xi_0 [K_c'^{-1}]] [K_c'] = \sigma_s[K_s'']$. Then, (125) holds. If instead $\sigma_s \in \Gamma$, it means that the left hand side of (125) is $\sigma_s[K_s''][\xi_0] = \sigma_s[\xi_0]$, because $\sigma_s \notin \text{dom}(K_s'')$, while the right hand side is $\sigma_s[\xi_0 [K_c'^{-1}]] [K_c'] = \sigma_s[\xi_0 [K_c'^{-1}]] [K_c'] = \sigma_s[\xi_0]$, because $\sigma_s \notin \text{dom}(K_c')$. Then, (125) holds. Finally, if $\sigma_s \notin \text{dom}(K_s'') \cup \Gamma$, it means that, because $K_s'' \vdash \sigma_s$, σ_s can be built using only elements of $\text{dom}(K_s'') \cup \Gamma$, and, because such elements are all substituted in the same way by substitutions $[K_s''][\xi_0]$ and $[\xi_0 [K_c'^{-1}]] [K_c']$, we have that σ_s is also substituted in the same way.

Now that we have proved (120) for $t_c = t'_c \cdot (\sigma_c[K_c'], \rho_c)$, it is simple to prove that (121) also holds because from (79) and (92), it follows that $(K_c \triangleright P_c) = (K'_c \triangleright P'_c[\rho_c/x])$ and $(K_s \triangleright P_s)_{\Upsilon, \Lambda} = (K''_s \triangleright P''_s[\gamma/x])_{\Upsilon, \Lambda}$, then, $(K_s \triangleright P_s)_{\Upsilon, \Lambda} \{(\xi_0 \langle \rho_c / \gamma \rangle)\}$

$$[K_c^{-1}] = (K_s'' \triangleright P_s''[\gamma/x])_{\Upsilon, \Lambda} \{ \xi_0[K_c'^{-1}] \langle \rho_c / \gamma \rangle \} = (K_s''[\xi_0[K_c'^{-1}]] \triangleright P_s''[\rho_c/x])_{\Upsilon, \Lambda} = (K_c \triangleright P_c).$$

The proof for the output case is similar, and is left to the reader. \square

The coincidence between equality of sets of concrete traces and trace equivalence defined over sets of symbolic traces is derived from the equivalence of the corresponding preorders. This is expressed by the following, final, theorem.

THEOREM 3.10. *Let P and Q be two spi processes, and K be a knowledge function with $\text{dom}(K) = \text{fn}(P) \cup \text{fn}(Q)$. Then, $\text{ctr}(P, K) \subseteq \text{ctr}(Q, K) \iff \text{str}(P, K) \subseteq_K \text{str}(Q, K)$.*

PROOF. Because by lemma 3.9 it follows that $\text{ctr}(P, K) = \text{concr}_K(\text{str}(P, K))$ and $\text{ctr}(Q, K) = \text{concr}_K(\text{str}(Q, K))$, the proof of theorem 3.10 reduces to proving that $\text{concr}_K(\text{str}(P, K)) \subseteq \text{concr}_K(\text{str}(Q, K)) \iff \text{str}(P, K) \subseteq_K \text{str}(Q, K)$. The two implications are proved separately:

1) Proof of $\text{str}(P, K) \subseteq_K \text{str}(Q, K) \Rightarrow \text{concr}_K(\text{str}(P, K)) \subseteq \text{concr}_K(\text{str}(Q, K))$

Let $t_c \in \text{concr}_K(\text{str}(P, K))$. Then, by the definition of concr_K , $t_c \in \text{concr}_K(t)$ for some $t \in \text{str}(P, K)$. Moreover, from $\text{str}(P, K) \subseteq_K \text{str}(Q, K)$ it follows that $\exists t' \in \text{str}(Q, K) \mid t \subseteq_K t'$. Finally, by theorem 3.8, $\text{concr}_K(t) \subseteq \text{concr}_K(t') \subseteq \text{concr}_K(\text{str}(Q, K))$, which implies that $t_c \in \text{concr}_K(\text{str}(Q, K))$.

2) Proof of $\text{concr}_K(\text{str}(P, K)) \subseteq \text{concr}_K(\text{str}(Q, K)) \Rightarrow \text{str}(P, K) \subseteq_K \text{str}(Q, K)$

Let $t \in \text{str}(P, K)$. Then, by the definition of concr_K , it follows that $\text{concr}_K(t) \subseteq \text{concr}_K(\text{str}(P, K)) \subseteq \text{concr}_K(\text{str}(Q, K))$. This implies that there must exist a symbolic trace $t' \in \text{str}(Q, K)$ such that $\text{concr}_K(t) \subseteq \text{concr}_K(t')$. Were this not true, we would have a set of traces $\{t_1, \dots, t_n\} \subset \text{str}(P, K)$ such that $t_i \not\subseteq_K t_j \forall i, j \in [1, \dots, n]$, $\text{concr}_K(t) \not\subseteq \text{concr}_K(t_i) \forall i \in [1, \dots, n]$, and $\text{concr}_K(t) \subseteq \text{concr}_K(\{t_1, \dots, t_n\})$, which is impossible. By theorem 3.8 it follows that $t \subseteq_K t'$. Because this reasoning holds for any $t \in \text{str}(P, K)$, it means that $t \in \text{str}(Q, K)$. \square

Note that the set $\text{str}(P, K)$ can be redundant, because it is possible that one of the symbolic traces of P represents concrete traces that are already represented by another symbolic trace. More precisely, a trace $t \in \text{str}(P, K)$ is redundant if $\exists t' \in \text{str}(P, K) \mid t \subseteq_K t'$. Redundant traces can be found by the trace inclusion definition and can be safely eliminated from the set of symbolic traces, therefore making the symbolic trace preorder check simpler.

For example, the trace corresponding to the path in the ES-LTS of Figure 10 that includes the dashed boxes (the path that leads to state 4.3') can be safely eliminated from the set of symbolic traces, because, as was shown above, it is included in the trace that corresponds to the path ending in state 4.3.

4. SOUNDNESS AND COMPLETENESS

In this section we prove that our concrete trace semantics is adequate to check testing equivalence, i.e., that two spi processes are testing equivalent iff they have the same set of concrete traces. This result is proved by showing the coincidence of the testing equivalence preorder $P \sqsubseteq Q$ and the corresponding trace preorder $\text{ctr}(P, K) \subseteq \text{ctr}(Q, K)$, where K is an initial knowledge function with $\text{dom}(K) = \text{fn}(P) \cup \text{fn}(Q)$. Soundness (i.e., $\text{ctr}(P, K) \subseteq \text{ctr}(Q, K) \Rightarrow P \sqsubseteq Q$) and completeness (i.e., $P \sqsubseteq Q \Rightarrow \text{ctr}(P, K) \subseteq \text{ctr}(Q, K)$) are proved separately.

4.1 Soundness

THEOREM 4.1. (Soundness) *Let P and Q be any two spi processes, and K an initial knowledge function with $\text{dom}(K) = \text{fn}(P) \cup \text{fn}(Q)$. Then $\text{ctr}(P, K) \subseteq \text{ctr}(Q, K) \Rightarrow P \sqsubseteq Q$.*

From the definition of $P \sqsubseteq Q$, given in (28), we must prove that the two conditions:

$$\text{ctr}(P, K) \subseteq \text{ctr}(Q, K) \quad (126)$$

$$\text{and } R \text{ is any test process such that } (P|R) \Downarrow \omega \quad (127)$$

also imply

$$(Q|R) \Downarrow \omega \quad (128)$$

From (127) and from the definition and properties of $\Downarrow \omega$, given in (27) and (24)-(26), it follows that there exist $R_0 \dots R_n$ and $P_0 \dots P_n$ such that, $R_0 = R$, $P_0 = P$, and

$$(P_i|R_i) \rightarrow (P_{i+1}|R_{i+1}) \quad \forall 0 \leq i < n \quad (129)$$

$$\text{and } R_n \Downarrow \omega \quad (130)$$

We claim that there must also exist $R'_0 \dots R'_n$ and $Q_0 \dots Q_n$ such that, $R'_0 = R_0$, $Q_0 = Q$, and

$$(Q_i|R'_i)(\rightarrow) * (Q_{i+1}|R'_{i+1}) \quad \forall 0 \leq i < n \quad (131)$$

$$\text{and } R'_n \Downarrow \omega \quad (132)$$

If this claim is true, (128) is also true by definition.

To prove this claim, we first introduce the relation \approx , which binds pairs (K, R) , where the first component K is an intruder knowledge function and the second component R is a test process. We set $(K_1, R_1) \approx (K_2, R_2)$ iff

$$\text{im}(K_1) = \text{im}(K_2) \quad (133)$$

$$\exists R^* \mid R_1 = R^*[K_1^{-1}] \wedge R_2 = R^*[K_2^{-1}] \quad (134)$$

Intuitively, $(K_1, R_1) \approx (K_2, R_2)$ means that R_1 and R_2 can be obtained by applying the substitutions $[K_1^{-1}]$ and $[K_2^{-1}]$, respectively, to a common expression R^* (containing free names that are identifiers of the set $\text{im}(K_1)$).

We can now express the following lemma:

LEMMA 4.2. *If (126) holds and there exist $R_0 \dots R_n$ and $P_0 \dots P_n$ such that $R_0 = R$, $P_0 = P$, and (129) holds, then $R'_0 \dots R'_n$, $Q_0 \dots Q_n$, $K_0 \dots K_n$ and $K'_0 \dots K'_n$ also exist such that $R'_0 = R_0$, $Q_0 = Q$, (131) holds, $K_0 = K'_0 = K$, and:*

$$\forall i \in [0, n] \quad (K_i, R_i) \approx (K'_i, R'_i) \quad (135)$$

$$\forall i \in [0, n] \quad K_i \triangleright P_i \xrightarrow{(\mu_i, \phi_i)} K_{i+1} \triangleright P_{i+1} \Rightarrow K'_i \triangleright Q_i \xrightarrow{(\mu_i, \phi_i)} K'_{i+1} \triangleright Q_{i+1} \quad (136)$$

The proof of our initial claim descends directly from this lemma, because the relationship $(K_n, R_n) \approx (K'_n, R'_n)$ implies that $\exists R^*$ such that $R_n = R^*[K_n^{-1}]$ and $R'_n = R^*[K'_n^{-1}]$, and, because substitutions $[K_n^{-1}]$ and $[K'_n^{-1}]$ cannot affect predicate $R_n \Downarrow \omega$, it follows that $R_n \Downarrow \omega$ iff $R'_n \Downarrow \omega$.

PROOF OF LEMMA 4.2. By induction on n .

Base ($n=0$). To prove the base it is sufficient to show that equation (135) holds for $n = 0$, i.e., $(K_0, R_0) \approx (K'_0, R'_0)$. Because $K'_0 = K_0$ and $R'_0 = R_0 = R$, we have that trivially $im(K_0) = im(K'_0)$ and we can write $R'_0 = R_0 = R^*[K_0^{-1}]$, where $R^* = R_0[K_0]$ (i.e., R^* is R_0 with each occurrence of terms in the domain of K_0 replaced by its unique identifier).

Induction. To prove the induction step, we assume that the lemma holds for a given $n = k$ and prove that it holds for $n = k + 1$ as well. Assume that the precondition of (136) with $n = k+1$ for $i = k$ holds, i.e., $K_k \triangleright P_k \xRightarrow{(\mu_k, \phi_k)} K_{k+1} \triangleright P_{k+1}$. Because all the traces of P are also traces of Q by (126), and because (136) holds for $n = k$, it follows that there must exist Q_{k+1} and K'_{k+1} such that $K'_k \triangleright Q_k \xRightarrow{(\mu_k, \phi_k)} K'_{k+1} \triangleright Q_{k+1}$, which implies that equation (136) also holds for $n = k + 1$. Moreover, by the definition of $\xRightarrow{(\mu_k, \phi_k)}$, we have that there must exist a Q'_k and a Q'_{k+1} such that

$$K'_k \triangleright Q_k \xRightarrow{\epsilon} K'_k \triangleright Q'_k \xrightarrow[\phi_k]{\mu_k} K'_{k+1} \triangleright Q'_{k+1} \xRightarrow{\epsilon} K'_{k+1} \triangleright Q_{k+1}. \quad (137)$$

To prove that equations (131) and (135) also hold for $n = k + 1$ we must distinguish two cases, according to the type of transition represented by label (μ_k, ϕ_k) .

The first case corresponds to an input transition. The derivation rule for input transitions (79) implies that $K_{k+1} = K_k$ and $K'_{k+1} = K'_k$. From (129) for $n = k + 1$ we know that $(P_k | R_k) \rightarrow (P_{k+1} | R_{k+1})$. From the derivation rule for input transitions, it follows that when P_k evolves into P_{k+1} , the input statement $\mu_k(x)[K_k^{-1}]$ with $P_{k+1} = P_k[\phi_k[K_k^{-1}]/x]$ is executed. Consequently, when R_k evolves to R_{k+1} , the corresponding output statement $\overline{\mu_k}(\phi_k)[K_k^{-1}]$ is executed. However, because (135) holds for $n = k$, it follows that $R_k = R_k^*[K_k^{-1}]$ and $R'_k = R_k^*[K_k^{-1}]$. Therefore, R_k^* can execute the output statement $\overline{\mu_k}(\phi_k)$, and evolve into a new process R_{k+1}^* such that $R_{k+1} = R_{k+1}^*[K_k^{-1}]$, and, consequently, R'_k can execute the output statement $\overline{\mu_k}(\phi_k)[K_k'^{-1}]$ and evolve into $R'_{k+1} = R_{k+1}^*[K_k'^{-1}]$. Therefore, it follows that equation (135) holds for $n = k + 1$. On the other hand, from (137) and from the derivation rule for input transitions we have that Q'_k can execute the same output statement $\overline{\mu_k}(\phi_k)[K_k'^{-1}]$, evolving into Q'_{k+1} . Therefore, R'_k and Q'_k can synchronize, i.e., $(Q'_k | R'_k) \rightarrow (Q'_{k+1} | R'_{k+1})$. This result, together with the fact that $\xrightarrow{\tau}$ means reaction, implies that equation (131) holds for $n = k + 1$.

The second case, corresponding to an output transition, is similar to the above and is left to the reader. \square

4.2 Completeness

THEOREM 4.3. (Completeness) *Let P and Q be any two spi processes, and K an initial knowledge function with $dom(K) = fn(P) \cup fn(Q)$. Then $P \sqsubseteq Q \Rightarrow ctr(P, K) \subseteq ctr(Q, K)$.*

We prove the equivalent implication $ctr(P, K) \not\subseteq ctr(Q, K) \Rightarrow P \not\sqsubseteq Q$.

If $ctr(P, K) \not\subseteq ctr(Q, K)$ then at least one trace t exists such that $t \in ctr(P, K)$ and $t \notin ctr(Q, K)$.

We show that, using t and the initial knowledge K , it is possible to build a spi test $R_{t,K}$ such that

$$(P|R_{t,K}) \Downarrow \omega \quad \wedge \quad (Q|R_{t,K}) \not\Downarrow \omega \quad (138)$$

which implies that $P \not\sqsubseteq Q$ by definition.

4.2.1 Construction of $R_{t,K}$. Process $R_{t,K}$ is the spi description of an intruder that can distinguish between P and Q , exploiting the fact that P can execute trace t while Q cannot. For example, if the purpose of the verification is a strong secrecy check, P and Q are typically two instances of the same protocol that carry two different secret data items. In this case, $R_{t,K}$ is an intruder that can determine which of the two data items has been transferred. Therefore, the algorithm to build $R_{t,K}$ is not only a technical means to prove the theorem, but it is also useful to build possible protocol attackers written in spi, starting from trace differences.

In the spi description of the intruder process $R_{t,K}$, the current intruder knowledge K_R is represented by means of a set of *knowledge variables*, l_i , whose names are the indexes that belong to $im(K_R)$ and whose values are the corresponding spi terms $K_R^{-1}(l_i)$. Because the initial knowledge of the intruder that we wish to represent is K , it means that initially the intruder process has a knowledge variable for each name in the set $im(K)$, bound to the corresponding value. Therefore, the general form taken by $R_{t,K}$ is

$$R_{t,K} = R_t^*(im(K))[K^{-1}] \quad (139)$$

where $R_t^*(im(K))$ is a spi process parameterized by free names that are elements of $im(K)$. The actual intruder process is simply obtained by substituting each index $l \in im(K)$ which occurs in $R_t^*(im(K))$ with the corresponding spi term $K^{-1}(l)$.

$R_t^*(im(K))$ is a linear sequence of spi constructions describing the action(s) taken by the intruder at each interaction, and it terminates with the successful test termination Ω process. If F is a set of free names, $R_t^*(F)$ can be defined inductively as:

$$R_t^*(F) = \begin{cases} \Omega & \text{if } t = \epsilon \\ A(a, F)R_{t'}^*(F'(a, F)) & \text{if } t = a.t' \end{cases} \quad (140)$$

where $A(a, F)$ is the spi description of the intruder actions that correspond to label a , and F represents the image of the current intruder knowledge function (and therefore, the set of variables where the current knowledge elements are stored), whereas $F'(a, F)$ is the new intruder knowledge function image after the occurrence of a .

If $a = (\mu, \phi)$ is an input transition label, then the corresponding intruder action is simply an output described by $A((\mu, \phi), F) = \overline{\mu}(\phi)$, and, because the intruder knowledge does not change, it follows that $F'(a, F) = F$.

If, instead, $a = (\overline{\mu}, \phi)$ is an output transition label, with $\phi = \langle \{\theta_1, l_1\}, \dots, \{\theta_n, l_n\} \rangle$, $\delta = \rho$, then $F'(a, F)$ can be computed by adding all the indexes occurring in expressions $\theta_1, \dots, \theta_n, \delta = \rho$, to F and removing the indexes l_1, \dots, l_n . In this case, the intruder actions $A(a, F)$ take the form $\mu(x).D(a, F)$, where x stands for the new variable where the input data is stored, and $D(a, F)$ is a list of additional internal spi actions, such as comparisons, decryptions, and pair splittings, taken by the intruder after receiving the input data. Such additional actions have

two purposes: on one hand they assign the correct values to the new knowledge variables, and on the other hand they check the consistency of the input data and of the new knowledge with the information carried by label a .

The actual name taken by the input variable x may vary according to how the input data are expected to be related to the intruder knowledge, which is expressed by ρ . If $\rho \in F'(a, F) \setminus F(a, F)$, i.e., the data term input by the intruder is labeled by a new index ρ in the new knowledge, then such a term is stored in the knowledge variable $x = \rho$, because it will be part of the new knowledge. In all other cases, the input term is something that can be built from other knowledge variables, so it must be stored in a temporary knowledge variable that will not be part of the final new knowledge. The name of such a temporary variable can be any new name.

Now we define the additional spi actions $D(a, F)$. If K is the intruder knowledge before the interaction represented by a occurs, and ρ is the input data, then $D(a, F)$ are the spi operations required to compute the spi representation of the new knowledge $f(\rho, K)$ and to perform all the possible consistency checks on it. Because we know from Proposition 3.4 that there is a finite reduction of $\text{dom}(K) \cup \{\rho\}$ that leads to $\text{dom}(f(\rho, K))$, it is possible to compute the values of the new knowledge variables by a finite sequence of computation steps, each of which corresponds to the implementation of a reduction rule. More precisely, at each step it is guaranteed that at least one of the equations (55)-(65) has a true pre-condition, so that the corresponding reduction rule can be applied. Applying reduction rule $U = \langle \Sigma_I, C, \Sigma_O \rangle$ means eliminating the set of terms Σ_I from the current knowledge domain and adding the set of terms Σ_O to it. Because we know in advance which elements should be eliminated from $\text{dom}(K)$ during the reduction and how such elements can be expressed in terms of the new knowledge variables, we can pre-determine the sequence of reduction rules to be applied and decide to which variables the new added terms must be assigned or with which old variables they must be compared. For each reduction rule of such a sequence, $D(a, F)$ contains a corresponding spi calculus operation that checks that the rule pre-condition holds, computes the new terms, and assigns them to the right new variables or compares them with the corresponding knowledge variables. Removal of old variables from the knowledge is implemented implicitly by avoiding further references to such variables.

To formally express the algorithm that pre-determines the sequence of reduction rules to be applied and generates the spi calculus sequence $D(a, F)$, it is necessary to introduce a knowledge representation function that represents, at each step, the set of all knowledge variables available to the spi test process, including the temporary variables, with their current values expressed in terms of the new knowledge variables.

Denote such a function Ξ . Initially, $\text{im}(\Xi) = F$, and Ξ^{-1} maps l_k onto θ_k for $1 \leq k \leq n$ and the other indexes of F onto themselves. Then, ρ is introduced in the new knowledge and the knowledge is reduced, computing, at each reduction step, the spi operation(s) that the intruder must execute. This is described in the algorithm in Figure 14. Initially, ρ is added to $\text{dom}(\Xi)$, but this occurs only if it cannot be built from the current knowledge, otherwise the intruder is simply instructed to test if ρ is really as expected. Then, the algorithm repeatedly looks for a reduction rule that can be applied, applies it as described above, and generates the corresponding


```

 $D(a, F)$  is initially empty;
let  $\rho$  be the output term label included in  $a$ ;
if  $r(\rho, \text{dom}(\Xi))$  then add " $[x \text{ is } \rho]$ " to  $D(a, F)$  else  $\Xi = \Xi \cup \{\langle \rho, x \rangle\}$  endif;
repeat
  if  $\exists \langle \text{suc}(\sigma), l_i \rangle \in \Xi$  then
     $y = \text{getID}(\sigma, \Xi)$ ;
    add "case  $l_i$  of 0:0  $\text{suc}(y)$ :" to  $D(a, F)$ ;
    if  $r(\sigma, \text{dom}(\Xi))$  then add " $[y \text{ is } \sigma[\Xi]]$ " to  $D(a, F)$  else  $\Xi = \Xi \cup \{\langle \sigma, y \rangle\}$  endif;
     $\Xi = \Xi \setminus \{\langle \text{suc}(\sigma), l_i \rangle\}$ ;
  else if  $\exists \langle (\sigma_1, \sigma_2), l_i \rangle \in \Xi$  then
     $y_1 = \text{getID}(\sigma_1, \Xi)$ ;  $y_2 = \text{getID}(\sigma_2, \Xi)$ ;
    add "let  $(y_1, y_2) = l_i$  in" to  $D(a, F)$ ;
    if  $r(\sigma_1, \text{dom}(\Xi))$  then add " $[y_1 \text{ is } \sigma_1[\Xi]]$ " to  $D(a, F)$  else  $\Xi = \Xi \cup \{\langle \sigma_1, y_1 \rangle\}$  endif;
    if  $r(\sigma_2, \text{dom}(\Xi))$  then add " $[y_2 \text{ is } \sigma_2[\Xi]]$ " to  $D(a, F)$  else  $\Xi = \Xi \cup \{\langle \sigma_2, y_2 \rangle\}$  endif;
     $\Xi = \Xi \setminus \{\langle (\sigma_1, \sigma_2), l_i \rangle\}$ ;
  else if  $\exists \langle \{\sigma_1\}_{\sigma_2}, l_i \rangle \in \Xi$  and  $r(\sigma_2, \text{dom}(\Xi))$  then
     $y_1 = \text{getID}(\sigma_1, \Xi)$ ;
    add "case  $l_i$  of  $\{\{y_1\}_{\sigma_2}[\Xi]$  in" to  $D(a, F)$ ;
    if  $r(\sigma_1, \text{dom}(\Xi))$  then add " $[y_1 \text{ is } \sigma_1[\Xi]]$ " to  $D(a, F)$  else  $\Xi = \Xi \cup \{\langle \sigma_1, y_1 \rangle\}$  endif;
     $\Xi = \Xi \setminus \{\langle \{\sigma_1\}_{\sigma_2}, l_i \rangle\}$ ;
  else if  $\exists \langle H(\sigma), l_i \rangle \in \Xi$  and  $r(\sigma, \text{dom}(\Xi))$  then
    add " $[l_i \text{ is } H(\sigma[\Xi])]$ " to  $D(a, F)$ ;
     $\Xi = \Xi \setminus \{\langle H(\sigma), l_i \rangle\}$ ;
  else if  $\exists \langle \{\{\sigma_1\}\}_{\sigma_2^+}, l_i \rangle \in \Xi$  and  $r(\sigma_2^-, \text{dom}(\Xi))$  then
     $y_1 = \text{getID}(\sigma_1, \Xi)$ ;
    if  $D(a, F)$  does not contain a decoding statement for  $l_i$  then
      add "case  $l_i$  of  $\{\{y_1\}\}_{\sigma_2^+}[\Xi]$  in" to  $D(a, F)$ ;
      if  $r(\sigma_1, \text{dom}(\Xi))$  then add " $[y_1 \text{ is } \sigma_1[\Xi]]$ " to  $D(a, F)$ 
        else  $\Xi = \Xi \cup \{\langle \sigma_1, y_1 \rangle\}$  endif;
    endif
    if  $r(\sigma_2^+, \text{dom}(\Xi))$  and  $r(\sigma_1, \text{dom}(\Xi))$  then
      add " $[l_i \text{ is } \{\{\sigma_1\}\}_{\sigma_2^+}[\Xi]]$ " to  $D(a, F)$ 
       $\Xi = \Xi \setminus \{\langle \{\{\sigma_1\}\}_{\sigma_2^+}, l_i \rangle\}$ ;
    endif
  else if  $\exists \langle \{\{\sigma_1\}\}_{\sigma_2^-}, l_i \rangle \in \Xi$  then
    symmetric of previous case
  else if  $\exists \langle \sigma^+, l_i \rangle \in \Xi$  and  $r(\sigma, \text{dom}(\Xi))$  then
    add " $[l_i \text{ is } \sigma^+[\Xi]]$ " to  $D(a, F)$ ;
     $\Xi = \Xi \setminus \{\langle \sigma^+, l_i \rangle\}$ ;
  else if  $\exists \langle \sigma^-, l_i \rangle \in \Xi$  and  $r(\sigma, \text{dom}(\Xi))$  then
    symmetric of previous case
  endif
until  $\nexists \langle \sigma, i \rangle \in \Xi \mid \sigma \neq i$ .

```

where $\text{getID}(\sigma, \Xi) = \begin{cases} \sigma & \text{if } \sigma \text{ is atomic and } \sigma \notin \text{dom}(\Xi) \\ \text{a new variable identifier} & \text{otherwise} \end{cases}$

Fig. 14. The algorithm to compute $D(a, F)$

intruder operations. The loop is repeated until all the new knowledge variables have been assigned, i.e., until $\Xi(l_i) = l_i$ for all $l_i \in \text{im}(f(\rho, K))$, which means that the spi process has a set of bound variables representing the new knowledge $f(\rho, K)$.

For example, if we assume that the label is $(\overline{\mu}, \phi)$ with $\mu = l_4$ and $\phi = \langle \{\{l_1\}_{l_2},$

$l_3\}\rangle, \emptyset, (l_2, \{l_4\}_{l_1})$, and that $F = \{l_4, l_3\}$, then it follows that initially $l_4 = \Xi(l_4)$ and $l_3 = \Xi(\{l_1\}_{l_2})$. After the input, the mapping $x = \Xi((l_2, \{l_4\}_{l_1}))$ is added to Ξ . It also follows that $F' = \{l_4, l_2, l_1\}$, i.e., the new knowledge variables that must be assigned are l_1 and l_2 . The list of spi operations to be executed after the input, determined by the algorithm in Figure 14, is:

$$\begin{aligned} D(a, F) ::= & \\ & \text{let } (l_2, x_1) = x \text{ in} \\ & \text{case } l_3 \text{ of } \{l_1\}_{l_2} \text{ in} \\ & \text{case } x_1 \text{ of } \{x_2\}_{l_1} \text{ in} \\ & [x_2 \text{ is } l_4] \end{aligned}$$

where x_1 and x_2 are two new temporary variables.

4.2.2 Properties of $R_{t,K}$. In this section we prove that process $R_{t,K}$ is such that equation (138) holds. To achieve this, it is necessary to prove the following preliminary theorems.

THEOREM 4.4. $(P|(A(a, im(K))R')[K^{-1}])(\rightarrow) * (P'|R'[K'^{-1}]) \iff K \triangleright P \xRightarrow{a} K' \triangleright P'$ where R' is any spi process.

PROOF. Consider the input and output cases separately.

If a is an input, it takes the form $(\sigma[K], \rho[K])$, and $A(a, im(K)) = \overline{\sigma[K]} \langle \rho[K] \rangle$. Therefore, we have $A(a, im(K))[K^{-1}] = \overline{\sigma} \langle \rho \rangle$. On the basis of the rules of the spi calculus reaction semantics, it is possible to affirm that $(P|\overline{\sigma} \langle \rho \rangle).(R'[K^{-1}]) (\rightarrow) * (P'|R'[K'^{-1}])$ iff $P(\rightarrow) * P_1$, where P_1 is a process that is ready to execute an input statement of the form $\sigma(x)$, and, when such a statement is executed and the value ρ is received, it reduces to $P'_1[\rho/x] = P'$. If we consider now the ES-LTS derivation system, and in particular rule (79), we find that the above condition can occur iff $K \triangleright P \xRightarrow{(\sigma[K], \rho[K])} K' \triangleright P'$, which is the right hand side of the theorem statement.

The case when a is an output is dealt with in a similar manner. If a is an output, it takes the form $(\overline{\sigma[K]}, \delta_K)$, where $\phi = \langle \{ \langle \theta[K'], \theta[K] \rangle \mid \theta \in \delta_{\Sigma}^-(\rho) \}, \delta^+ = [K'], \rho[K'] \rangle$, and $A(a, im(K))[K^{-1}] = \sigma(x).(D(a, im(K))[K^{-1}])$. On the basis of the rules of the spi calculus reaction semantics, it is possible to affirm that the reactions $(P|\sigma(x).D(a, im(K))A'[K^{-1}])(\rightarrow) * (P'|A'[K'^{-1}])$ are possible iff $P(\rightarrow) * P_1$, where P_1 is a process that is ready to execute an output statement of the form $\overline{\sigma} \langle \rho' \rangle$ with ρ' such that $(D(a, im(K))A'[K^{-1}])[\rho'/x] \equiv A'[K'^{-1}]$. The latter structural equivalence means that all the spi constructs included in $D(a, im(K))$ can be eliminated by applying rules (1)-(7), with K'^{-1} representing the combination of the substitutions K^{-1} , $\langle \rho'/x \rangle$, and any further substitution implied by the application of rules (1)-(7). Because $D(a, im(K))$ has been built so as to compute $K' = f(\rho', K)$, and because $D(a, im(K))$ also contains a check that ascertains whether the received data is related to K' as expressed by $\rho[K']$, if the above structural equivalence holds we can conclude that $\rho' = \rho$ and that the subsequent application of the substitutions K^{-1} , ρ'/x and the substitutions implied by the application of rules (1)-(7) yield the same effect as substitution K' . Moreover, because $D(a, im(K))$ also checks whether $\theta[K]$ can be obtained after the computation of K' as $\theta[K']$, it follows that

rule (78) holds, as does the right hand side of the theorem statement. \square

THEOREM 4.5. $(P|R_{t,K})(\rightarrow) * (P'|\Omega) \iff \exists K' \mid K \triangleright P \xRightarrow{t} K' \triangleright P'$

PROOF. By induction on the length of t .

Base ($t = \epsilon$). If $t = \epsilon$, $R_{t,K} = \Omega$, therefore $(P|R_{t,K})(\rightarrow) * (P'|\Omega)$ holds iff $P(\rightarrow) * P'$, which means $K \triangleright P \xRightarrow{\epsilon} K \triangleright P'$.

Induction. We wish to show that if theorem 4.5 holds for a trace t , then it also holds for a trace $a.t$. If we set

$$X_1 = (P|R_{a.t,K})(\rightarrow) * (P_1|R_{t,K_1}) \quad (141)$$

$$Y_1 = K \triangleright P \xRightarrow{t} K_1 \triangleright P_1 \quad (142)$$

$$X = (P_1|R_{t,K_1})(\rightarrow) * (P'|\Omega) \quad (143)$$

$$Y = K_1 \triangleright P_1 \xRightarrow{t} K' \triangleright P' \quad (144)$$

it follows from theorem 4.4 that $X_1 \iff Y_1$. Moreover, because theorem 4.5 holds for trace t , it also follows that $X \iff Y$. However, $X_1 \iff Y_1$ and $X \iff Y$ imply $(X_1 \wedge X) \iff (Y_1 \wedge Y)$, which is theorem 4.5 for trace $a.t$. \square

COROLLARY 4.6. $(P|R_{t,K}) \Downarrow \omega \iff \exists P', K' \mid K \triangleright P \xRightarrow{t} K' \triangleright P'$

PROOF. Because P is not a test process, $P \not\Downarrow \omega$, and P cannot evolve into a process that exhibits ω . Then, $(P|R_{t,K}) \Downarrow \omega$ holds if and only if $(P|R_{t,K})(\rightarrow) * (P'|\Omega)$ and $P' \Downarrow \omega$. By the linear structure of $R_{t,K}$, it is clear that the above condition is true if and only if $(P|R_{t,K})(\rightarrow) * (P'|\Omega)$. Therefore, Corollary 4.6 descends directly from Theorem 4.5. \square

Corollary 4.6 states that $(P|R_{t,K}) \Downarrow \omega$ if and only if t is a trace of P . Therefore it also implies that equation (138) holds.

4.2.3 An example of attack detection and intruder construction. Figure 15 illustrates how our theory can be used to carry out the verification of the authenticity property for the wide-mouthed frog protocol. To find the protocol bug, the model must admit at least two protocol sessions, because the attacker needs to capture messages exchanged during a session to cheat the responder of the other session. This means that at least two instances of each process must be present. The upper frame of Figure 15 shows the protocol and reference specifications with two parallel sessions. Note that the specifications of both agents are the same as in Figures 3 and 5, but they have been made parametric with respect to channels, to let each session take place on a separate set of channels. Moreover, the unspecified behavior $F()$ has been made explicit as an output event on channel c_F of the cleartext of the message received by agent B from (hopefully) agent A .

The two sessions are represented by processes A_1, S_1, B_1 and A_2, S_2, B_2 , respectively, and their interactions are carried out on channels $c_{AS_1}, c_{SB_1}, c_{AB_1}$ and $c_{AS_2}, c_{SB_2}, c_{AB_2}$, respectively. The transferred messages are M_1 and M_2 , respectively, and, to distinguish the two fresh keys created by A in the two sessions, they are called k_{AB_1} and k_{AB_2} , respectively.

The details of the attack are shown in the central frame of the figure: the first session takes place in the usual way, but the attacker intercepts messages $\{k_{AB_1}\}_{k_{SB}}$

$P_A(M, c_{AS}, c_{AB})$	$\triangleq (\nu k_{AB}) (\overline{c_{AS}} \langle \{k_{AB}\}_{k_{AS}} \rangle . \overline{c_{AB}} \langle \{M\}_{k_{AB}} \rangle)$
$P_S(c_{AS}, c_{SB})$	$\triangleq c_{AS}(x_1) . \text{case } x_1 \text{ of } \{x_2\}_{k_{AS}} \text{ in } \overline{c_{SB}} \langle \{x_2\}_{k_{SB}} \rangle$
$P_B(c_{SB}, c_{AB}, c_F)$	$\triangleq c_{SB}(y_1) . \text{case } y_1 \text{ of } \{y_2\}_{k_{SB}} \text{ in } c_{AB}(y_3) .$ $\text{case } y_3 \text{ of } \{y_4\}_{y_2} \text{ in } \overline{c_F} \langle y_4 \rangle$
$P_{B_{spec}}(M, c_{SB}, c_{AB}, c_F)$	$\triangleq c_{SB}(y_1) . \text{case } y_1 \text{ of } \{y_2\}_{k_{SB}} \text{ in } c_{AB}(y_3) .$ $\text{case } y_3 \text{ of } \{y_4\}_{y_2} \text{ in } \overline{c_F} \langle M \rangle$
$P_{wmf}(M_1, M_2)$	$\triangleq (\nu k_{AS})(\nu k_{SB})($ $P_A(M_1, c_{AS_1}, c_{AB_1}) \mid P_S(c_{AS_1}, c_{SB_1}) \mid P_B(c_{SB_1}, c_{AB_1}, c_{F_1})$ $\mid P_A(M_2, c_{AS_2}, c_{AB_2}) \mid P_S(c_{AS_2}, c_{SB_2}) \mid P_B(c_{SB_2}, c_{AB_2}, c_{F_2}))$
$P_{wmf_{spec}}(M_1, M_2)$	$\triangleq (\nu k_{AS})(\nu k_{SB})($ $P_A(M_1, c_{AS_1}, c_{AB_1}) \mid P_S(c_{AS_1}, c_{SB_1}) \mid P_{B_{spec}}(M_1, c_{SB_1}, c_{AB_1}, c_{F_1})$ $\mid P_A(M_2, c_{AS_2}, c_{AB_2}) \mid P_S(c_{AS_2}, c_{SB_2}) \mid P_{B_{spec}}(M_2, c_{SB_2}, c_{AB_2}, c_{F_2}))$
$2'_1) \ S_1 \rightarrow I(B_1) : \{k_{AB_1}\}_{k_{SB}} \text{ on } c_{SB_1}$	$2'_1) \ I \triangleq c_{SB_1}(l_{10}).$
$2''_1) \ I(S_1) \rightarrow B_1 : \{k_{AB_1}\}_{k_{SB}} \text{ on } c_{SB_1}$	$2''_1) \ \overline{c_{SB_1}} \langle l_{10} \rangle.$
$2''_2) \ I(S_2) \rightarrow B_2 : \{k_{AB_1}\}_{k_{SB}} \text{ on } c_{SB_2}$	$2''_2) \ \overline{c_{SB_2}} \langle l_{10} \rangle.$
$3'_1) \ A_1 \rightarrow I(B_1) : \{M_1\}_{k_{AB_1}} \text{ on } c_{AB_1}$	$3'_1) \ c_{AB_1}(l_{11}).$
$3''_1) \ I(A_1) \rightarrow B_1 : \{M_1\}_{k_{AB_1}} \text{ on } c_{AB_1}$	$3''_1) \ \overline{c_{AB_1}} \langle l_{11} \rangle.$
$3''_2) \ I(A_2) \rightarrow B_2 : \{M_1\}_{k_{AB_1}} \text{ on } c_{AB_2}$	$3''_2) \ \overline{c_{AB_2}} \langle l_{11} \rangle.$
	$c_{F_1}(x_1).[x_1 \text{ is } M_1]$
	$c_{F_2}(x_2).[x_2 \text{ is } M_1].\Omega$
$K = \left\{ (c_{AS_1}, l_0), (c_{AB_1}, l_1), (M_1, l_2), (c_{SB_1}, l_3), (c_{F_1}, l_4), \right.$ $\left. (c_{AS_2}, l_5), (c_{AB_2}, l_6), (M_2, l_7), (c_{SB_2}, l_8), (c_{F_2}, l_9) \right\}$	
$2'_1) \Downarrow (\tau, \top).(\overline{l_3}, \langle \top, \emptyset \rangle, \langle \emptyset, \emptyset, l_{10} \rangle)$	$2'_1) \Downarrow (\overline{l_3}, \langle \emptyset, \emptyset, l_{10} \rangle)$
$2''_1) \Downarrow (\tau, \top).(l_3, \gamma_0)$	$2''_1) \Downarrow (l_3, l_{10})$
$2''_2) \Downarrow (\tau, \langle l_{10}/\gamma_0 \rangle).(l_8, \gamma_1)$	$2''_2) \Downarrow (l_8, l_{10})$
$3'_1) \Downarrow (\tau, \langle l_{10}/\gamma_1 \rangle).(\overline{l_1}, \langle \top, \emptyset \rangle, \langle \emptyset, \emptyset, l_{11} \rangle)$	$3'_1) \Downarrow (\overline{l_1}, \langle \emptyset, \emptyset, l_{11} \rangle)$
$3''_1) \Downarrow (\tau, \top).(l_1, \gamma_2)$	$3''_1) \Downarrow (l_1, l_{11})$
$3''_2) \Downarrow (\tau, \langle l_{11}/\gamma_2 \rangle).(l_6, \gamma_3)$	$3''_2) \Downarrow (l_6, l_{11})$
$\Downarrow (\tau, \langle l_{11}/\gamma_3 \rangle).(\overline{l_4}, \langle \top, \emptyset \rangle, \langle \emptyset, \emptyset, l_2 \rangle)$	$\Downarrow (\overline{l_4}, \langle \emptyset, \emptyset, l_2 \rangle)$
$\Downarrow (\tau, \top).(\overline{l_9}, \langle \top, \emptyset \rangle, \langle \emptyset, \emptyset, l_2 \rangle)$	$\Downarrow (\overline{l_9}, \langle \emptyset, \emptyset, l_2 \rangle)$
symbolic trace	concrete trace

Fig. 15. An attack on the protocol and the related intruder

and $\{M_1\}_{k_{AB_1}}$, and forwards them to both B_1 and B_2 , therefore leading B to believe that two sessions with A have been carried out. For this reason, each one of the steps labeled 2) and 3) in Figure 3 is split into three subsequent steps here: in the first step the intruder intercepts the message, and in the second and third steps it forwards the message to the two sessions. For example, in $2'_1)$, the message that S_1 sends on channel c_{SB_1} is intercepted by the attacker, which plays the role of B_1

($I(B_1)$). Then the attacker forwards the message to B_1 in step $2''_1$) acting as S_1 ($I(S_1)$), and to B_2 , acting as S_2 ($I(S_2)$ in step $2''_2$). This last message is the first step of the attack, because B becomes convinced that k_{AB_1} is a fresh key created by A and coming from S_2 . Using the same technique in the next steps, the attacker sends B_2 the message $\{M_1\}_{k_{AB_1}}$, leading B to believe that it comes from A . At this point we have two sessions ending on B , while only one of them was actually started by A .

In P_{wmf} the two instances of P_B output (on channels c_{F_1} and c_{F_2} , respectively) the cleartext of the second message they have received, while in $P_{wmf_{spec}}$ they *magically* output the messages that the two instances of P_A plan to send them, i.e., M_1 and M_2 , respectively. Thus, a difference between the expected value output by $P_{B_{spec}}$ in $P_{wmf_{spec}}$ and the value output by P_B in P_{wmf} means that an authenticity flaw has been found.

The initial attacker's knowledge K is shown at the top of the lower frame of Figure 15. By using a first preliminary prototype tool that implements our theory, we found that P_{wmf} and $P_{wmf_{spec}}$ are not testing equivalent. In particular, the symbolic trace reported in Figure 15 comes from P_{wmf} and is not included in any corresponding trace of $P_{wmf_{spec}}$. Note that the symbolic trace follows each step of the attack: first the output on channel c_{SB_1} ($\overline{l_3}$) of $\{k_{AB_1}\}_{k_{SB}}$ (stored as l_{10} in the attacker's knowledge), next the input of the generic term γ_0 on channel c_{SB_1} , then the subsequent specialization $\langle l_{10}/\gamma_0 \rangle$ due to the decryption operation *case y_1 of $\{y_2\}_{k_{SB}}$ in \dots* , etc. This symbolic trace represents just one corresponding concrete trace, which can be computed by applying (111). From the concrete trace, the spi specification of process I reported in the central frame of Figure 15 can be built by using (139).

5. RELATED WORK

The problem of spi calculus testing equivalence verification has already been addressed in [Abadi and Gordon 1998] and in [Boreale et al. 2002]. However, both these papers provide proof systems rather than finite models to be checked.

The method proposed in this paper to verify testing equivalence, instead, is based on state exploration and is completely automatic. Despite this basic difference between our method and the previous methods, some of the techniques used in our paper have been inspired by [Boreale et al. 2002]. In particular, the concept of an environment-sensitive labeled transition system has been borrowed from [Boreale et al. 2002]. However, because finiteness of models is not needed for theorem proving, the authors of [Boreale et al. 2002] do not deal with symbolic techniques, and therefore their ES-LTS can be related only to our concrete ES-LTS.

As in [Boreale et al. 2002], our concrete ES-LTS states comprise an environment knowledge representation and a spi calculus process expression, and our concrete ES-LTS transitions are labeled by a process action label and an environment action label. However, both environment knowledge representations and transition labels differ substantially from those introduced in [Boreale et al. 2002]. Such differences are mainly related to the different verification techniques used. Because [Boreale et al. 2002] uses proof methods, it does not pay particular attention to computational aspects, such as efficient storage and management of the environ-

ment knowledge, finite size of the ES-LTS itself, efficient trace comparison methods, etc. Instead, the above aspects needed particular attention in the development of our technique.

Specifically, concerning environment knowledge, the difference is that we keep it in a minimized canonical form, whereas in [Boreale et al. 2002] it simply collects terms exactly as they are received from the process. Concerning transition labels, the difference is in regard to input and output transition labels. Specifically, in [Boreale et al. 2002], output transitions are labeled by process action and environment action labels taking the forms $(\nu \tilde{b})\overline{a}(M)$ and $\eta(x)$, respectively, where \tilde{b} is a set of fresh names possibly included as sub-terms of the output data M , η is an expression specifying how the channel name a can be built using the current knowledge variables, and x is the name of the new knowledge variable where the environment stores M (in this case, the pair (M, x) is added to the knowledge). Input action labels take the form aM and $(\nu \tilde{b})\overline{\eta}(\zeta)$, where the meaning of η , a , and M is the same as in the previous case, and ζ describes how M can be built from the current knowledge variables and \tilde{b} . A first difference that can be pointed out is that in [Boreale et al. 2002] only the environment action label is used in trace comparisons, while the process action label is redundant. A second difference is that in [Boreale et al. 2002] transition labels do not incorporate all the information needed to verify testing equivalence, because they take into account only how a message or a channel is built from the intruder knowledge variables but they do not take into account how a new piece of knowledge added to the environment knowledge is related to the messages already present in it. As a consequence, in [Boreale et al. 2002], trace equivalence cannot be defined simply as trace identity, as we do, but its definition is more complex and involves a step-by-step knowledge equivalence concept. More precisely, trace equivalence in [Boreale et al. 2002] requires the identity of environment action labels of transitions and the *equivalence* between the knowledge representations after each step of the trace. It is worth noting also that, because in [Boreale et al. 2002] the environment knowledge is not kept in a minimized form, η and ζ must include both encoding and decoding operators, whereas with our minimized environment knowledge, messages produced by the intruder are built using only constructive operators, and are much simpler.

A further point that differentiates our model from that presented in [Boreale et al. 2002] is that we deal with the whole spi calculus language as defined in [Abadi and Gordon 1999], with the only exclusion being replication, whereas [Boreale et al. 2002] deals with a spi calculus dialect where public-key encryption, hashing, integers, and non-atomic keys are not considered.

Symbolic techniques, as a basis for providing process algebras with alternative semantics, have been introduced in [Hennessy and Lin 1995] for a CCS-like process algebra, and in [Boreale and De Nicola 1996] for π -calculus, and have been used in both cases for bisimulation equivalence checking. Because both CCS and π -calculus do not deal with cryptographic primitives, both the process and its environment always perceive the exchanged messages in the same way: this makes an explicit model of knowledge useless and leads to a much simpler management of transitions with respect to algebras equipped with cryptographic primitives.

The use of symbolic techniques to make cryptographic protocol models finite,

therefore enabling the use of state exploration methods, has also been addressed by other researchers [Amadio and Lugiez 2000; Boreale 2001; Boreale and Buscemi 2002; Fiore and Abadi 2001; Huima 1999]. However, they do not deal with testing equivalence verification, but only with the verification of simple security properties based on the set of data that the intruder can produce. Moreover, they use simplified models characterized by public channels only, i.e., no private channel can be modeled. Among these, only [Boreale 2001; Boreale and Buscemi 2002; Fiore and Abadi 2001] deal with spi calculus specifications, although none includes channels and integers. [Boreale 2001] deals only with shared atomic keys, whereas [Boreale and Buscemi 2002] enhances [Boreale 2001], allowing further cryptographic primitives, such as hashing and public/private-key cryptography. It allows messages to be arbitrarily nested, but keys still remain atomic. On the other hand, [Fiore and Abadi 2001] deals with non-atomic keys, although no completeness results are given. [Boreale 2001; Boreale and Buscemi 2002; Fiore and Abadi 2001] use a derivation system to generate symbolic protocol traces. Such traces, however, are redundant because the derivation systems used to generate them do not exclude the values that cannot be produced by the intruder. Therefore traces must be analyzed, possibly on the fly as in [Boreale 2001; Boreale and Buscemi 2002], to determine which of them are meaningful, i.e., actually correspond to concrete traces. Conversely, with our approach, the intruder generation capabilities are incorporated in the derivation system, which directly generates only meaningful traces.

Symbolic techniques have also been used in Athena [Song 1999], a model checker based on strand space models [Thayer et al. 1998]. The most interesting point in favor of Athena is its ability to deal symbolically with potentially infinite numbers of sessions, although termination of the verification algorithm may not always be guaranteed. Note also that Athena cannot deal with testing equivalence verifications, and is based on protocol models simpler than those expressible in spi calculus (it cannot describe non-atomic keys and private channels, and admits tests only while receiving data).

A minimized representation of the intruder knowledge that has some similarities with ours has been introduced in [Clarke et al. 1998] and incorporated in the Brutus model checker [Clarke et al. 2000]. Such a representation is composed of the set of messages learned by the intruder, closed under elimination rules (pair splitting and decryption), so as to include only data items that cannot be further decomposed. The technique presented in [Clarke et al. 1998] is heavily based on the theory presented in [Prawitz 1965], and is not compatible with non-atomic keys. This limitation is recognized by the authors, who restrict their keys to be names. Our technique starts from the same idea of representing only data items that cannot be decomposed, but, instead of simply exploiting the theory presented in [Prawitz 1965], we develop a new specific theory that also works with non-atomic keys. Moreover, our knowledge representations associate data with indexes, so as to make testing equivalence verification possible.

Similarly to Athena, Brutus [Clarke et al. 2000] is based on a protocol description language that is less expressive than spi calculus, and cannot verify testing equivalence. Moreover, it does not use symbolic representations, but, to have finite models, it limits the length of the data that the intruder can build, thereby

restricting the set of attacks it can find.

Note that the trace semantics presented in this paper can be scaled down to deal more efficiently with security properties that are simpler than testing equivalence. For example, to deal with security properties that depend only on the set of data that the intruder can produce, the indexing on the intruder knowledge and the labels on traces can be safely removed. Using this reduced model, properties can be verified by the classical reachability analysis or model checking approaches. For example, to verify that the intruder never discovers a given data term σ , it is sufficient to search the state space for states characterized by an intruder knowledge that can produce σ . Based on this consideration, the trace semantics introduced in this paper can be regarded as an extension of all the previously proposed similar semantics [Boreale 2001; Boreale and Buscemi 2002; Boreale et al. 2002; Fiore and Abadi 2001].

6. CONCLUSIONS

A trace semantics for spi calculus that can be used to automatically check testing equivalence by exhaustive generation and simple comparison of traces has been presented. Finiteness of the set of traces, and therefore decidability, has been ensured by using symbolic techniques and by excluding specifications characterized by infinite numbers of parallel processes. Apart from the latter limitation, the whole spi calculus language has been addressed and the soundness and completeness of the proposed trace equivalence with respect to testing equivalence have been formally proved.

The theory presented in this paper can be used as a correct basis to implement a fully automatic testing equivalence checker for spi calculus. Whenever two descriptions are not recognized as equivalent, the spi calculus specification of an intruder that exploits the difference can be automatically built by the tool. In this way, the verification process can be made quite intuitive and simple. The problems related to the efficient implementation of such a tool, including the adoption of complexity reduction mechanisms such as symmetry-based reductions or partial-order reductions, are outside the scope of this paper and are left for further study.

The results presented in this paper extend what has already been achieved in the field of automatic verification of security protocols, because they give a viable alternative to the use of theorem proving for the verification of complex security properties based on testing equivalence.

APPENDIX

symbol	meaning	page
P, Q, R	range over spi calculus processes	5
\mathcal{A}	is the set of spi calculus names	8
m, x, y	range over names: x and y preferably used for variables	5
$\mathcal{M}(\mathcal{A})$	is the set of spi calculus terms built with names \mathcal{A}	8
σ, ρ, θ	range over terms	4
$\sigma \preceq \rho$	σ is a subterm of ρ	8

symbol	meaning	page
$fn(P)$	is the set of free names of P	8
$bn(P)$	is the set of bound names of P	8
$n(P)$	is the whole set of names of P , i.e., $n(P) = fn(P) \cup bn(P)$	8
$\langle \sigma_1/\rho_1, \dots, \sigma_n/\rho_n \rangle$	is a term substitution list where $\sigma_i, \rho_i \in \mathcal{M}(\mathcal{A})$ and ρ_i are pairwise different terms. It also denotes the corresponding term substitution function (called substitution for short)	8,20
λ	ranges over substitutions	8
\top	is the null substitution	8
$\lambda_1 \lambda_2$	is the composition of substitutions λ_1 and λ_2	8
$[\sigma_1/\rho_1, \dots, \sigma_n/\rho_n]$	stands for $[\langle \sigma_1/\rho_1, \dots, \sigma_n/\rho_n \rangle]$	8
$[\langle \sigma_1/\rho_1, \dots, \sigma_n/\rho_n \rangle]$	is the postfix operator that replaces each occurrence of ρ_i with σ_i with the rule that if $\rho_i \preceq \rho_j$ and $i \neq j$, any occurrence of ρ_i inside an occurrence of ρ_j is not substituted	8,20
Σ	ranges over sets of terms. It usually represents the set of terms the intruder has learned	9
$\widehat{\Sigma}$	is the closure of the set of terms Σ under rules (32)-(45)	13
$\overline{\Sigma}$	is the <i>minimal closure seed</i> of Σ	14
$\delta_{\overline{\Sigma}}^-(\rho)$	is the set of terms eliminated from $\overline{\Sigma}$ in the reduction of $\overline{\Sigma} \cup \{\rho\}$	18
$\delta_{\overline{\Sigma}}^+(\rho)$	is the set of terms added to $\overline{\Sigma}$ in the reduction of $\overline{\Sigma} \cup \{\rho\}$	18
$\delta_{\overline{\Sigma}}^{\equiv}(\rho)$	is the set of those terms that become decipherable after the reduction of $\overline{\Sigma} \cup \{\rho\}$ without being eliminated from it	18
$r(\sigma, \Sigma)$	is the predicate that is true if σ can be built from the elements of Σ using only constructive closure rules	15
$\langle \Sigma_I, C, \Sigma_O \rangle$	is the reduction rule with premises Σ_I and conclusions Σ_O	16
U	ranges over reduction rules	16
$\Sigma \xrightarrow{U} \Sigma'$	specifies that Σ is transformed into Σ' by reduction rule U	16
I	is the set of indexes: an infinite, countable, totally ordered set of names that extends the set of spi calculus names and is such that $\mathcal{A} \cap I = \emptyset$	19
l	ranges over indexes	19
$next(l)$	the successor of l in I	19

symbol	meaning	page
K	is the intruder knowledge representation. It is defined as a bijective function $K : \overline{\Sigma} \rightarrow L$, with $L \subset I$	19
$f(\rho, K)$	is the new intruder knowledge after term ρ becomes known	20
$[K]$	is the postfix operator that replaces each occurrence of $\rho \preceq \theta \mid \rho \in \text{Dom}(K)$ with $K(\rho)$	21
$\overline{\Sigma} \vdash \sigma$	$\overline{\Sigma}$ can produce term σ , i.e., $\sigma \in \widehat{\overline{\Sigma}}$	19
$K \vdash \sigma$	K can produce term σ , i.e., $\sigma \in \text{dom}(K)$	19
$\delta_K^-(\rho)$	is the set of pairs describing the terms that are eliminated from K in the transformation from K to $f(\rho, K)$ when ρ is received	23
$\delta_K^{\overline{-}}(\rho)$	is the set of pairs describing the terms that become decipherable after ρ has been received without being eliminated from the intruder knowledge domain	23
$\delta_K(\rho)$	stands for the triple $\langle \delta_K^-(\rho), \delta_K^{\overline{-}}(\rho), \rho \rangle [K']$, where $K' = f(\rho, K)$	22
$K \triangleright P$	is the state of the concrete ES-LTS	12
μ	is the label describing the action performed by process P	12
ϕ	is the label describing the complementary action performed by the environment	12
Γ	is the set of <i>unspecialized generic terms</i> . It extends the set of spi calculus names and is such that $\Gamma \cap (\mathcal{A} \cup I) = \emptyset$	25
γ	ranges over unspecialized generic terms	25
η	ranges over generic terms	25
$\Upsilon(\gamma)$	is the minimal closure seed of the terms that were available to the intruder when γ was generated	25
η_{Υ}	is a generic term η that must be interpreted according to Υ	26
ξ	ranges over specializations	26
Δ	ranges over sets of specializations	26
$\text{new}_{\Upsilon}(\xi)$	is the set of new unspecialized generic terms introduced in Υ when specialization ξ is applied	26
$\text{old}_{\Upsilon}(\xi)$	is the set of unspecialized generic terms replaced in Υ when specialization ξ is applied	26
$\Upsilon\{\xi\}$	is the generic term interpretation function updated after the application of specialization ξ	26
S_{Υ}	is the set of all possible specializations (in the current state)	27

symbol	meaning	page
$\xi_1 \subset \xi_2$	ξ_1 can be <i>produced</i> by ξ_2 , i.e., a non-null generic term specialization $\xi \in S_{\Upsilon\{\xi_2\}}$ exists such that $\xi_1 = \xi_2\xi$	27
$\xi \subset \Delta$	ξ can be produced by $\Delta \subseteq S_{\Upsilon}$, i.e., $\xi \subset \xi_i$ for some $\xi_i \in \Delta$	27
$\langle \xi, \delta_{\Lambda} \rangle$	is an <i>extended narrowing specification</i> , where ξ is a specialization that must be applied and $\delta_{\Lambda} = \{\xi_1, \dots, \xi_k\}$ is an irredundant set of forbidden further specializations	28
Λ	is the irredundant set of all forbidden specializations accumulated up to the current state	28
$S_{\Upsilon, \Lambda}$	is the set of specializations compatible with Λ , i.e., $\{\xi \in S_{\Upsilon} \mid \xi \not\subset \Lambda\}$	29
$\eta_{\Upsilon, \Lambda}$	is a generic term η that must be interpreted according to Υ , but with the limitations imposed by Λ	29
$\sigma \bullet \rho$	is the <i>unification</i> of σ and ρ . It is defined as the irredundant set of specializations compatible with Λ that make σ and ρ equal, i.e., $\{\xi \in S_{\Upsilon, \Lambda} \mid \sigma[\xi] = \rho[\xi]\}$	29
$\sigma \circ \rho$	is the set of specializations compatible with Λ that must be applied to σ and ρ to make σ a term encrypted under key ρ , i.e., $\sigma \circ \rho = \{\xi \in S_{\Upsilon, \Lambda} \cup \{\top\} \mid \exists \eta \mid \sigma[\xi] = \{\eta\}_{\rho[\xi]}\}$	32
$\sigma \oplus \rho$	is the public-key variant of $\sigma \circ \rho$	32
$\sigma \ominus \rho$	is the private-key variant of $\sigma \circ \rho$	32
$\Lambda\{\xi\}$	is the set of forbidden specializations updated after the application of ξ	29
$(K \triangleright P)_{\Upsilon, \Lambda}$	is the state of the symbolic ES-LTS. Both K and P can contain generic terms interpreted according to Υ and Λ	12
$\mathcal{K}_{\Upsilon, \Lambda}$	is the set of all the symbolic knowledge functions consistent with the interpretation given by Υ and Λ	29
$\mathcal{P}_{\Upsilon, \Lambda}$	is the set of all the symbolic processes consistent with the interpretation given by Υ and Λ	29
$(K \triangleright P)_{\Upsilon, \Lambda}\{\xi\}$	is the symbolic ES-LTS state, updated after the application of specialization ξ , i.e., $(K[\xi] \triangleright P[\xi])_{\Upsilon\{\xi\}, \Lambda\{\xi\}}$	30
$f_{\langle \xi, \delta_{\Lambda} \rangle}(\rho, K_{\Upsilon, \Lambda})$	is the symbolic version of function f after the application of the narrowing specified by $\langle \xi, \delta_{\Lambda} \rangle$, i.e., $f(\rho[\xi], K[\xi])_{\Upsilon\{\xi\}, (\Lambda \cup \delta_{\Lambda})\{\xi\}}$	31

symbol	meaning	page
$\Theta(\rho, K_{\Upsilon, \Lambda})$	is the irredundant set of narrowings $\langle \xi, \delta_\Lambda \rangle$ that make $f_{\langle \xi, \delta_\Lambda \rangle}(\rho, K_{\Upsilon, \Lambda})$ a valid knowledge function, i.e., $\{\langle \xi, \delta_\Lambda \rangle \mid \xi \in S_{\Upsilon, \Lambda}, \delta_\Lambda \in S_{\Upsilon}, f_{\langle \xi, \delta_\Lambda \rangle}(\rho, K_{\Upsilon, \Lambda}) \in \mathcal{K}_{\Upsilon, \Lambda}\}$	31
$P \rightarrow P'$	is the reaction relation, i.e., a binary relation on processes such that $P \rightarrow P'$ means that process P can evolve into P' by performing an internal synchronization	10
$P(\rightarrow) * P'$	is the reflexive and transitive closure of \rightarrow	11
ω	is the distinguished success action that signals that a process has passed a test	11
Ω	is a distinguished process that can perform only ω	11
$P \downarrow \omega$	means that P exhibits ω	11
$P \Downarrow \omega$	is the convergence predicate, defined as $\exists Q \mid (P(\rightarrow) * Q) \wedge (Q \downarrow \omega)$	11
$P \sqsubseteq Q$	is the testing preorder, defined as $\forall R \ ((P R) \Downarrow \omega \implies (Q R) \Downarrow \omega)$	11
$P \simeq Q$	means that P and Q are testing equivalent, i.e., no test can distinguish between them ($P \simeq Q \triangleq (P \sqsubseteq Q) \wedge (Q \sqsubseteq P)$)	11
t, t'	range over traces	34
ϵ	is the empty trace	34
a, a'	range over trace symbols	34
$t.t'$	is trace t concatenated with trace t'	34
$a.t$	is the trace obtained by prepending symbol a to trace t	34
$t.a$	is the trace obtained by appending symbol a to trace t	34
π	is the process action label of an input or output transition	34
μ, ϕ	is a generic trace symbol	34
$\xRightarrow[t]{\epsilon}$	binary relation on the set of states of an ES-LTS such that $S_1 \xRightarrow[t]{\epsilon} S_2$ iff t is a trace starting from state S_1 and ending with state S_2	35
$\xRightarrow[\epsilon]{\tau}$	is the reflexive and transitive closure of $\xRightarrow[\tau]{\epsilon}$	35
$ctr(P, K)$	is the set of the concrete traces of a spi process P with an initial intruder knowledge K , i.e., $\{t \mid \exists P', K' \mid K \triangleright P \xRightarrow[t]{\epsilon} K' \triangleright P'\}$	35
$str(P, K)$	is the set of the symbolic traces of a spi process P with an initial intruder knowledge K , i.e., $\{t \mid \exists P', K', \Upsilon', \Lambda' \mid (K \triangleright P)_{\emptyset, \emptyset} \xRightarrow[t]{\epsilon} (K' \triangleright P')_{\Upsilon', \Lambda'}\}$	35

symbol	meaning	page
im_K^t	is $im(K')$, where K' is the knowledge that is reached starting from the initial knowledge K after the execution of t	36
Υ_K^t	is $\Upsilon'[K']$, where K' is the knowledge that is reached starting from the initial knowledge K after the execution of t	36
Λ_K^t	is $\Lambda'[K']$, where K' is the knowledge that is reached starting from the initial knowledge K after the execution of t	36
$n(\delta_K)$	is the set of names occurring in $\delta_K(\rho)$	36
$old(\delta_K)$	is the set of indexes removed from the intruder knowledge image in an output transition having environment label $\delta_K(\rho)$	36
λ_{δ_K}	is the substitution that converts any canonical representation $\theta[K']$ into $\theta[K]$, where K and K' are the knowledge functions before and after an output transition with environment label $\delta_K(\rho)$, respectively	36
$concr_K(t)$	is the set of concrete traces corresponding to a symbolic trace t with an initial intruder knowledge K	37,38
$S_{\Upsilon,\Lambda}^c$	is the subset of $S_{\Upsilon,\Lambda}$ that includes all the specializations that convert symbolic states into concrete states, i.e., $S_{\Upsilon,\Lambda}^c = \{\xi \in S_{\Upsilon,\Lambda} \mid \Upsilon\{\xi\} = \emptyset\}$	37
$act(t, \xi)$	is the concrete trace obtained by applying $\xi \in S_{\Upsilon,\Lambda}^c$ to symbolic trace t	37
$a\{\xi\}$	is trace symbol a updated according to specialization ξ	37
$b_t(\xi)$	is the specialization that must be applied to t' to obtain the concrete trace, where $t = t'.(\tau, \xi_1).a$, and $\xi \in S_{\Upsilon,\Lambda}^c$	37
$concr_K(str(P, K))$	is the whole set of concrete traces represented by the symbolic traces of P , i.e., $\{concr_K(t) \mid t \in str(P, K)\}$	38
$\langle t_1, \xi_1 \rangle \subseteq_K \langle t_2, \xi_2 \rangle$	means that all the concrete traces that are obtained from t_1 via specializations compatible with ξ_1 can also be obtained from t_2 via specializations compatible with ξ_2	38
$t_1 \subseteq_K t_2$	means that $\langle t_1, \top \rangle \subseteq_K \langle t_2, \top \rangle$	38
T	ranges over sets of traces	39
$T_1 \subseteq_K T_2$	means that $\forall t_1 \in T_1 \ \exists t_2 \in T_2 \mid t_1 \subseteq_K t_2$	39
$(K_1, R_1) \approx (K_2, R_2)$	means that R_1 and R_2 can be obtained by applying the substitutions $[K_1^{-1}]$ and $[K_2^{-1}]$, respectively, to a common expression R^*	44

symbol	meaning	page
$R_{t,K}$	is the spi description of an intruder that can distinguish between P and Q , where $K = fn(P) \cup fn(Q)$, and t is a symbolic trace of P , but not of Q	46
$R_t^*(im(K))$	is a spi process parameterized by free names that are elements of $im(K)$	46
$A(a, F)$	is the spi description of the intruder actions that correspond to label a , where F represents the image of the current intruder knowledge function	46
$D(a, F)$	is the list of spi calculus actions taken by the intruder after the input corresponding to label a	46
Ξ	is a knowledge representation function that represents, at each step, the set of all knowledge variables available to the spi test process, including the temporary ones, with their current values expressed in terms of the new knowledge variables	47

ACKNOWLEDGMENTS

We are grateful to the anonymous referees for their helpful suggestions. We are also grateful to Ivan Cibrario Bertolotti for his useful hints.

REFERENCES

- ABADI, M. AND GORDON, A. D. 1998. A bisimulation method for cryptographic protocols. *Nordic Journal of Computing* 5, 4, 267–303.
- ABADI, M. AND GORDON, A. D. 1999. A calculus for cryptographic protocols: The spi calculus. *Inf. Comput.* 148, 1, 1–70.
- AMADIO, R. M. AND LUGIEZ, D. 2000. On the reachability problem in cryptographic protocols. In *Proceedings of the 11th International Conference on Concurrency Theory (CONCUR 2000)*. Lecture Notes in Computer Science, vol. 1877. Springer-Verlag, Heidelberg, 380–394.
- BOREALE, M. 2001. Symbolic trace analysis of cryptographic protocols. In *Proceedings of the 28th International Colloquium on Automata, Languages, and Programming (ICALP 2001)*. Lecture Notes in Computer Science, vol. 2076. Springer-Verlag, Heidelberg, 667–681.
- BOREALE, M. AND BUSCEMI, M. G. 2002. A framework for the analysis of security protocols. In *Proceedings of the 13th International Conference on Concurrency Theory (CONCUR 2002)*. Lecture Notes in Computer Science, vol. 2421. Springer-Verlag, Heidelberg, 483–498.
- BOREALE, M. AND DE NICOLA, R. 1996. A symbolic semantics for the π -calculus. *Inf. Comput.* 126, 1, 34–52.
- BOREALE, M., DE NICOLA, R., AND PUGLIESE, R. 2002. Proof techniques for cryptographic processes. *SIAM J. Comput.* 31, 3, 947–986.
- CLARKE, E. M., JHA, S., AND MARRERO, W. 1998. Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In *Proceedings of the IFIP Working Conference on Programming Concepts and Methods (PROCOMET 1998)*. Chapman & Hall, London, 87–106.
- CLARKE, E. M., JHA, S., AND MARRERO, W. 2000. Verifying security protocols with Brutus. *ACM Trans. Softw. Eng. Meth.* 9, 4, 443–487.

- DE NICOLA, R. AND HENNESSY, M. C. B. 1984. Testing equivalence for processes. *Theor. Comput. Sci.* 34, 1-2, 84–133.
- DURANTE, L., SISTO, R., AND VALENZANO, A. 2000. A state-exploration technique for spicalculus testing equivalence verification. In *Proceedings of the IFIP International joint conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XIII) and Protocol Specification, Testing and Verification (PSTV XX)*. Kluwer Academic Publishers, Dordrecht, 155–170.
- FIGLIORE, M. AND ABADI, M. 2001. Computing symbolic models for verifying cryptographic protocols. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop (CSFW 2001)*. IEEE Computer Society Press, Washington, 160–173.
- HENNESSY, M. AND LIN, H. 1995. Symbolic bisimulations. *Theor. Comput. Sci.* 138, 2, 353–389.
- HUIMA, A. 1999. Efficient infinite-state analysis of security protocols. In *Proceedings of the FLOC Workshop on Formal Methods and Security Protocols*.
- LOWE, G. 1996. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of the 2nd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 1996)*. Lecture Notes in Computer Science, vol. 1055. Springer-Verlag, Heidelberg, 147–166.
- LOWE, G. 1997. Casper: a compiler for the analysis of security protocols. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop (CSFW 1997)*. IEEE Computer Society Press, Washington, 18–30.
- LOWE, G. 1999. Towards a completeness result for model checking security protocols. *Journal of Computer Security* 7, 2-3, 89–146.
- LOWE, G. AND ROSCOE, B. 1997. Using CSP to detect errors in the TMn protocol. *IEEE Trans. Softw. Eng.* 23, 10, 659–669.
- MILLEN, J. K., CLARK, S. C., AND FREEDMAN, S. B. 1987. The Interrogator: Protocol security analysis. *IEEE Trans. Softw. Eng.* 13, 2, 274–288.
- MILNER, R., PARROW, J., AND WALKER, D. 1992. A calculus of mobile processes, parts I and II. *Inf. Comput.* 100, 1, 1–77.
- PAULSON, L. C. 1998. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security* 6, 85–128.
- PRAWITZ, D. 1965. *Natural Deduction: A Proof-Theoretical Study*. Almqvist & Wiksell, Stockholm.
- SCHNEIDER, S. 1998. Verifying authentication protocols in CSP. *IEEE Trans. Softw. Eng.* 24, 9, 741–758.
- SONG, D. X. 1999. Athena: a new efficient automatic checker for security protocol analysis. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop (CSFW 1999)*. IEEE Computer Society Press, Washington, 192–202.
- THAYER, J., HERZOG, J., AND GUTTMAN, J. 1998. Strand spaces: Why is a security protocol correct? In *Proceedings of the 19th IEEE Symposium on Security and Privacy (S&P 1998)*. IEEE Computer Society Press, Washington, 160–171.

Received M Y; revised M Y; accepted M Y