Available online at www.sciencedirect.com
ScienceDirect
Theoretical Computer Science 367 (2006) 88–122
Theoretical Computer Science
www.elsevier.com/locate/tcs

On the semantics of Alice&Bob specifications of security protocols☆

Carlos Caleiro[a,*], Luca Viganò[b], David Basin[b]

[a]*Department of Mathematics, CLC and SQIG-IT, IST, Lisbon, Portugal*
[b]*Department of Computer Science, ETH Zurich, Switzerland*

Abstract

In the context of security protocols, the so-called Alice&Bob notation is often used to describe the messages exchanged between honest principals in successful protocol runs. While intuitive, this notation is ambiguous in its description of the actions taken by principals, in particular with respect to the conditions they must check when executing their roles and the actions they must take when the checks fail.

In this paper, we investigate the semantics of protocol specifications in Alice&Bob notation. We provide both a denotational and an operational semantics for such specifications, rigorously accounting for these conditions and actions. Our denotational semantics is based on a notion of incremental symbolic runs, which reflect the data possessed by principals and how this data increases monotonically during protocol execution. We contrast this with a standard formalization of the behavior of principals, which directly interprets message exchanges as sequences of atomic actions. In particular, we provide a complete characterization of the situations where this simpler, direct approach is adequate and prove that incremental symbolic runs are more expressive in general. Our operational semantics, which is guided by the denotational semantics, implements each role of the specified protocol as a sequential process of the pattern-matching spi calculus.

Keywords: Security protocols; Protocol models; Message sequences; Alice&Bob specifications; Spi calculus

1. Introduction

Context and motivation: *Message Sequence Charts* and related notations [23,30,33,37] describe communication between entities, e.g. objects in a distributed system, as a sequence of messages exchanged in an execution scenario. In the context of security protocols, these diagrams are written in a syntax commonly called *Alice&Bob notation* [36,42]. Here, the objects are principals participating in the protocol, like Alice and Bob, and communication is described by a

☆ This work was partially supported by FCT and EU FEDER through POCI (namely via the Project QuantLog POCI/MAT/55796/2004 of CLC), by the project "AVISPA: Automated Validation of Internet Security Protocols and Applications" (FET Open Project IST-2001-39252 and BBW Project 02.0431), and by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under Grant number 5005-67322. This paper generalizes and extends the preliminary results of [14]. We thank Sebastian Mödersheim and the referees for their useful comments.

* Corresponding author. Tel.: +351 21 8417138; fax: +351 21 8417048.
E-mail addresses: ccal@math.ist.utl.pt (C. Caleiro), vigano@inf.ethz.ch (L. Viganò).

0304-3975/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2006.08.041

linear sequence of message-exchange steps of the form

$$A \to B \ : \ M.$$

While Alice&Bob notation is compact and intuitive, it is informal and provides an incomplete description of the actions of principals engaged in the protocol execution. Namely, it describes only the actions taken in a complete protocol run between honest principals. In contrast, it does not describe what happens during unsuccessful runs, for example, with possibly dishonest participants. Moreover, it leaves implicit that even in successful runs, certain checks must be taken and execution will abort when the checks fail.

In the standard use of sequence diagrams, these alternative execution possibilities are unproblematic: sequence diagrams only specify requirements on *inter*-object communication, rather than specifying *intra*-object behavior. That is, they specify possible communication scenarios between objects rather than the inner-workings of the objects themselves. Intra-object behavior is instead described using an automata-like language, which gives a full behavioral specification of each object (see, for instance, [23] for a discussion on this). But in the security protocol community, Alice&Bob notation is often taken as defining the protocols themselves, that is, as a notation that can be more-or-less (e.g., perhaps with additional information about the types of data manipulated) translated to protocol automata for the principals involved.

Let us give a simple example, pointing to some of the problems. Consider a protocol with the step

$$A \to B \ : \ \{\!|M|\!\}^{\mathrm{s}}_K,$$

indicating that a principal in the role A sends the message M encrypted with the symmetric (hence the superscript s) key K to a principal in the role B. Intuitively, this describes steps that must be taken by both (principals in the roles of) A and B: A sends $\{\!|M|\!\}^{\mathrm{s}}_K$ and B receives $\{\!|M|\!\}^{\mathrm{s}}_K$. Moreover, to send the message, A must compose $\{\!|M|\!\}^{\mathrm{s}}_K$, e.g. A must possess $\{\!|M|\!\}^{\mathrm{s}}_K$, which he then directly sends, or he possesses the components M and K and then performs encryption first. What about B? His responsibility is to receive $\{\!|M|\!\}^{\mathrm{s}}_K$. Moreover, when he possesses both M and the corresponding decryption key (which we can assume to be K itself as the encryption is symmetric), then he should check that he received the expected message. He can do this by checking that decryption with K yields M. If this check fails, then B should abort the protocol execution. Equivalently, B could encrypt M with K, compare the result with the message he received, and again abort if the comparison fails. However, when B does not possess M and K, he cannot perform this check, although he should perform it later when (if at all) he accumulates enough data.

Hence, to reason about protocols where checks like the one we just described must occur, it is not sufficient to formalize the behavior of principals by directly interpreting message exchanges as sequences of atomic actions. Various researchers (e.g. [2,3,9,13,16,20,24,26,28,31,32]) have thus investigated ways of making explicit at least part of what is left implicit, or even unspecified, in Alice&Bob-style specifications. This includes the view that a principal has of a received message with respect to his current knowledge, the conditions a principal must check during protocol execution and the actions he must take when the checks fail, and, thus, also the conditions that must be satisfied by the principals in order to execute a protocol. For instance, the original strand spaces approach, e.g. [40], which is directly based on Alice&Bob-style protocol specifications, was extended in [20,26] to model such explicit checks.

Making such information explicit is essential for providing a suitable semantics for protocol specification languages, and also for correctly translating Alice&Bob protocol specifications into low-level languages that can be handled by automated analysis tools (including those cited above). In fact, many of the above works are directed towards protocol verification, and hence the solutions they propose incorporate additional information that ends up hiding some of the relevant modeling details, which are independent of the particular kind of formalism and the kinds of reasoning employed to carry out the analysis in each case. This paper aims at giving a precise description of these modeling details, thereby providing an abstract denotational semantics for Alice&Bob specifications. We also provide a corresponding operational semantics, which implements protocol roles as spi calculus processes.

Contributions: We see our contributions as follows. First, we provide a *denotational semantics* for security protocol specifications based on Alice&Bob notation. This semantics is based on a notion of *incremental symbolic runs*, which reflect the data possessed by principals and how this data increases monotonically during protocol execution.

Second, we provide a formal characterization of the "denotational semantic spectrum" of Alice&Bob specifications. Namely, we compare our denotational semantics with the simpler *direct interpretation* that formalizes the behavior of principals by directly interpreting message exchanges as sequences of atomic actions. We give a complete characterization of the situations where this simpler interpretation is adequate and prove that incremental symbolic runs are

more expressive in general. We call our denotational semantics the *fine interpretation* and we also introduce a *coarse interpretation*, which we show to be appropriate for those protocol descriptions where principals are not required to check messages that they previously received.

Third, we use our fine interpretation to precisely characterize the roles of an Alice&Bob protocol specification as processes in the sequential spi calculus. That is, guided by our abstract denotational semantics, we give an *operational semantics* for Alice&Bob specifications in terms of the *pattern-matching spi calculus* [28], where we interpret a security protocol as a collection of sequential processes, one for each protocol role, which includes all the machinery necessary to handle exceptional cases. Moreover, the processes obtained are effectively executable in the sense that both the analysis of received messages and the construction of sent messages are explicitly prescribed.

Together, the two semantics that we consider (the operational one and the denotational one based on incremental symbolic runs) give Alice&Bob specifications an unambiguous meaning, both in terms of operations performed by principals and their communication and knowledge at a more abstract level. We prove that these two semantics are equivalent in an appropriate sense.

Overall, our results are independent of the details of the formalism used for protocol analysis (e.g., multiset rewriting, distributed temporal logic, strand spaces, process algebras, trace-based models, and so on; see, for example, [3,4,7–10,15,22,32,34,35,38–41]), of the details of the intruder model (which could be the standard Dolev–Yao model or the model of an intruder with different capabilities), and of the particular security properties considered (such as secrecy of some data or different forms of authentication). Therefore, our results can be used to provide a formal footing for using Alice&Bob notation in different security protocol analysis tools and for guiding the translation from this notation into protocol automata or other descriptions suitable for direct analysis or for actual implementation.

The differences that we describe between the direct, coarse, and fine interpretations have a clear relevance for protocol analysis models and protocol correctness results. In fact, considering different interpretations may lead to conflicting results. We show that, in general, a protocol correctness result obtained for the direct interpretation is not very meaningful because the fine interpretation may introduce further behaviors that violate the relevant security properties. Similarly, an attack to a security property obtained by means of a coarse interpretation of the protocol may also not have a counterpart in terms of the behaviors allowed by the fine interpretation.

Organization: We proceed as follows. In Section 2, we introduce Alice&Bob protocol specifications and their direct interpretation. Afterwards, in Section 3, we introduce a more adequate denotational semantics, followed in Section 4 with a corresponding operational semantics. In Section 5, we discuss related work and draw conclusions. For the sake of readability, the full proofs of the technical results are given in the appendix.

2. The direct interpretation of Alice&Bob protocol specifications

2.1. Protocol specifications in Alice&Bob notation

Security protocols describe how principals exchange messages, built using cryptographic primitives, in order to obtain security guarantees. We begin our formalization of these protocols by introducing an algebra of messages and then define the actions that principals can take during protocol execution.

Protocol specifications are parametric in the sense that they prescribe a general recipe for communication that can be used by different principals playing in the different protocols roles (sender, responder, server, etc.). The messages transmitted are bit-strings, but, for our purposes, they can also be taken from any other appropriate set of values and our results are independent of such details. We just consider fixed a set *Princ* of *principal identifier* symbols and a set *Num* of "number" symbols. The elements of *Num* are used to model random data, like nonces and keys. We will use upper-case letters like $A, B, C, \ldots$ to denote principals, N and I to denote numbers, and M to denote an arbitrary message. All of these symbols may be annotated with subscripts or superscripts. The algebra of messages can then be defined as follows.

Definition 1. *Messages* are built inductively from *atomic messages* (identifiers and number symbols) by *pairing*, *encryption*, *inversion*, and *hashing*. For M, M_1, and M_2 messages, we write the pairing of M_1 and M_2 as $M_1; M_2$, the *asymmetric* encryption of M_1 by M_2 as $\{\!|M_1|\!\}^{\mathrm{a}}_{M_2}$, the *symmetric* encryption of M_1 by M_2 as $\{\!|M_1|\!\}^{\mathrm{s}}_{M_2}$, the asymmetric inversion of M by M^{-1}, and the application of a hash function H to M as $H(M)$.

For readability, we will often write K to denote a message intended to be used as an encryption key. Whenever the distinction between symmetric and asymmetric encryption is not important we will write $\{\!|M|\!\}_K$.

Observe that K^{-1} denotes the asymmetric inverse of K. Some approaches, e.g. [35], denote by K^{-1} the inverse of a symmetric key K, with $K^{-1} = K$. In this paper, following [7] and many other approaches, we consider a model in which messages are untyped and hence the inverse key cannot be determined from (the type of) the key. In our model, every message has an asymmetric inverse. Hence, principals (including dishonest ones) can compose a message from its submessages but can neither generate K^{-1} from K nor vice versa. The only ways in which a principal A can obtain the inverse of a key are to know it initially, to receive it in a message, or when it is the private key of an asymmetric key-pair that A has generated. Furthermore, note that we consider only a key K and its inverse K^{-1}, but not the inverse of the inverse $(K^{-1})^{-1}$; see, for instance, the message derivation rules in Definition 11.

Definition 2. The set $sub(M)$ of *submessages* of a message M is defined inductively by

$$sub(M) = \begin{cases} \{M\} & \text{if } M \text{ is atomic,} \\ \{M_1; M_2\} \cup sub(M_1) \cup sub(M_2) & \text{if } M = M_1; M_2, \\ \{\{\!|M_1|\!\}_K\} \cup sub(M_1) \cup sub(K) & \text{if } M = \{\!|M_1|\!\}_K, \\ \{K^{-1}\} \cup sub(K) & \text{if } M = K^{-1}, \\ \{H(M_1)\} \cup sub(M_1) & \text{if } M = H(M_1). \end{cases}$$

If S is a set of messages, then we will also write $sub(S)$ to denote the set of all submessages of messages in S, i.e., $sub(S) = \bigcup_{M \in S} sub(M)$. Note that, in this paper, we will work in the *free algebra*, where syntactically different terms denote different messages.

As notation for keys, we will write K_A to denote a public key of the principal A, with the corresponding private key K_A^{-1}, and we will write K_{AB} to denote a symmetric key that is shared by the principals A and B. As is often done in security protocol analysis, we follow the *perfect cryptography assumption*, which postulates that the cryptographic primitives themselves cannot be attacked, and hence the only way to decrypt a message is to possess the appropriate key. While the public key K_A of a principal A may be known by other principals, the key K_A^{-1} should only be known by A. Similarly, a shared key K_{AB} should not be known by principals other than A and B (except by trusted third parties when the protocol prescribes that).

We will use the following notation for sequences. We write $w = \langle w_1.w_2.w_3 \ldots \rangle$ to denote a (possibly infinite) sequence w composed of the elements $w_1, w_2, w_3, \ldots$. Further, we write $|w|$ to denote the length of the sequence w, where $|\langle\rangle| = 0$ for the empty sequence $\langle\rangle$ and $|w| = \infty$ whenever w is infinite. Finally, we write $w \cdot w'$ to denote the concatenation of two sequences, provided that the first sequence is finite, and we write $w|_i$ to denote the prefix of w of length i, i.e. $w|_i = \langle w_1 \ldots w_i \rangle$, provided that $0 \leqslant i \leqslant |w|$. Clearly, $w|_0 = \langle\rangle$.

Definition 3. An *Alice&Bob protocol specification* consists of a finite sequence $\langle \mathbf{prot}_1 \ldots \mathbf{prot}_m \rangle$ of message-exchange steps, each of the form

$$(\mathbf{prot}_q) \quad A_{\mathrm{s}} \to A_{\mathrm{r}} \ : \ (N_1, \ldots, N_t).\ M,$$

where A_{s} and A_{r} are distinct principal symbols and $N_1, \ldots, N_t$ are distinct number symbols.

Overall, an Alice&Bob protocol specification involves one principal symbol corresponding to each intended protocol participant playing a specific role. We write *Part* to denote the set of all *honest protocol participants*, i.e. those principals who send or receive messages in some step of the protocol.

The description also involves a collection of number symbols. The number symbols $N_1, \ldots, N_t$ at each step represent values that must be (pseudo-)randomly generated by the sender A_{s} just before the message M is sent to the receiver A_{r}. Hence, we assume that these values are *fresh*, i.e., that they do not occur in any message from the preceding steps of the protocol specification. [1]

[1] This means that the new numbers have not appeared before and thus they cannot be used to generate the inverse of any known key.

$$
\begin{array}{ll}
(\mathbf{nspk}_1) & A \to B : (N_1).\ \{\!|N_1; A|\!\}^{a}_{K_B} \\
(\mathbf{nspk}_2) & B \to A : (N_2).\ \{\!|N_1; N_2|\!\}^{a}_{K_A} \\
(\mathbf{nspk}_3) & A \to B : \qquad \{\!|N_2|\!\}^{a}_{K_B}
\end{array}
$$

Fig. 1. The simplified Needham–Schroeder Public-Key protocol (NSPK).

Example 4. As an example, consider the simplified Needham–Schroeder Public-Key protocol (NSPK) [18]. Fig. 1 shows the usual Alice&Bob specification of NSPK, where A and B are principal symbols that identify the roles played in one run of the protocol.

Let us from now on consider a fixed, arbitrary protocol specification

$$\langle \mathbf{prot}_1 \dots \mathbf{prot}_m \rangle,$$

where, for brevity, we will often omit "specification" and simply speak of "the protocol $\langle \mathbf{prot}_1 \dots \mathbf{prot}_m \rangle$". We define A-*role*, the role of an $A \in Part$, as the sequence of message-exchange steps

$$\langle \mathbf{prot}_1^A \dots \mathbf{prot}_n^A \rangle$$

that results by removing those steps from the protocol where A is neither the sender nor the receiver. Hence, each role prescribes the sequence of *actions* that are executed by the principal playing it in a successful run of the protocol.

Definition 5. The *actions* that can be executed by a principal playing a role in a protocol are:
- $\mathbf{s}(M, B)$—sending the message M to the principal B,
- $\mathbf{r}(M)$—receiving the message M, and
- $\mathbf{f}(N)$—generating the fresh number N.

These actions reflect the fact that the underlying network may be hostile: sending actions name the intended recipient but receiving actions do not name the message's sender. In fact, we can assume, as is standard, that the network is controlled by, and can be identified with, a *Dolev–Yao intruder* [25] who can compose, send, and intercept messages at will, but, following the perfect cryptography assumption, cannot break cryptography. Our results, however, are independent of the particular capabilities of the intruder.

Note that Alice&Bob specifications say little about the internal actions of principals. Aside from the two communication actions, we have modeled only one internal action $\mathbf{f}(N)$. It is often the case that protocol specifications in Alice&Bob notation do not make the generation of fresh nonces explicit, since one can always infer these nonces by collecting the new atomic data in the message being sent. In our case, we chose to represent fresh nonce generation explicitly, not only for technical simplicity, but also because nonces are security-relevant data. All other internal activities are abstracted away. For example, neither Alice&Bob specifications nor the denotational semantics that we will present shortly prescribe a specific way of constructing messages prior to sending them. However, we will do this carefully in our operational semantics, so that the resulting process descriptions are directly executable. This computation depends, in general, on the data collected from the messages received in preceding steps. Similarly, Alice&Bob specifications also do not specify how a recipient should process received messages.

2.2. The direct interpretation

We now introduce a direct interpretation of Alice&Bob protocol specifications, which we obtain by directly extracting a sequence of actions for each protocol participant. Formally, we proceed by defining the sequence of actions, or *run*, that characterizes the behavior of each of the participants in a complete, successful execution of the protocol. From now on, we will focus on the protocol role $A\text{-}role = \langle \mathbf{prot}_1^A \dots \mathbf{prot}_n^A \rangle$.

$$A \to B : (N).\ \{\!|N|\!\}^{a}_{K_A}$$
$$B \to A : \qquad N.$$

Fig. 2. A non-executable protocol specification.

$$A\text{-}run : \langle \mathbf{f}(N_1).\, \mathbf{s}(\{\!|N_1; A|\!\}^{a}_{K_B}, B).\, \mathbf{r}(\{\!|N_1; N_2|\!\}^{a}_{K_A}).\, \mathbf{s}(\{\!|N_2|\!\}^{a}_{K_B}, B)\rangle$$
$$B\text{-}run : \langle \mathbf{r}(\{\!|N_1; A|\!\}^{a}_{K_B}).\, \mathbf{f}(N_2).\, \mathbf{s}(\{\!|N_1; N_2|\!\}^{a}_{K_A}, A).\, \mathbf{r}(\{\!|N_2|\!\}^{a}_{K_B})\rangle$$

Fig. 3. The initiator and responder runs of the NSPK protocol.

Definition 6. Let $A \in Part$. In the *direct interpretation* of Alice&Bob protocol specifications, the *run* corresponding to the execution of A's role in the protocol is

$$A\text{-}run = seq(\mathbf{prot}^A_1) \cdot \cdots \cdot seq(\mathbf{prot}^A_n),$$

where *seq* is a function that translates a message-exchange step into a sequence of actions:

$$seq(A_{\mathrm{s}} \to A_{\mathrm{r}} : (N_1, \ldots, N_t).\, M) = \begin{cases} \langle \mathbf{f}(N_1) \ldots \mathbf{f}(N_t)\, .\, \mathbf{s}(M, A_r)\rangle & \text{if } A = A_{\mathrm{s}}, \\ \langle \mathbf{r}(M)\rangle & \text{if } A = A_{\mathrm{r}}. \end{cases}$$

Note that A-*run* is parametric and a concrete run is obtained by instantiating all the symbols occurring in the actions with concrete data.

While the above definition is straightforward, internal actions other than nonce generation are not represented. This turns out to have important consequences revolving around the notion of "possession". First, most protocols are designed so that the participants can actually execute all the steps prescribed for them in their concrete runs. But this is only the case when principals can construct all the messages they are supposed to send using the data they possess. For instance, an encrypted message $\{\!|M|\!\}_K$ can only be synthesized if one possesses both M and K. Similarly, a hashed message $H(M)$ can only be synthesized if one has M (assuming, as we do, that the hash function H is known by all parties).

Second, it is left implicit what internal actions a principal should take after receiving a message. In particular, if a principal is to gain access to any proper submessage of a received message M (for example, to test that it agrees with previously received or generated data, or to compose new messages from it), then it must be feasible for him to parse the received message, i.e., decompose and verify the message's structure. This also depends on the data he possesses. For example, an encrypted message $\{\!|M|\!\}^{\mathrm{a}}_K$ can only be feasibly parsed if one possesses both M and K, or the decryption key K^{-1}. A similar restriction applies to hashes: $H(M)$ can only be feasibly parsed given M.

We will give formal definitions of what it means for a protocol to be *executable* and an action to be *feasible* in Section 3 (and in Section 4 we will then introduce a constructive notion of implementable message). We explain here, and illustrate with examples, that the direct interpretation does not take these two notions into account.

Example 7. The toy protocol described in Fig. 2 exemplifies non-executability: B cannot send the fresh nonce N back to A as B is not supposed to know A's private key K_A^{-1}.

Moreover, received messages need not be feasibly parsable, even when the protocol itself is executable. The root of the problem is that these notions are *dynamic*: they depend on the participants' possessions, which monotonically grow with each protocol action taken. Handling this will require a data-sensitive view of runs, which we introduce later.

Example 8. As a simple example of an executable protocol, consider the NSPK protocol. There are two participants playing in two roles, an initiator A and a responder B. The two corresponding sequences of actions A-*run* and B-*run* are shown in Fig. 3.

These two sequences provide a high-level representation of the behavior of the participants of the NSPK protocol playing in the respective roles. Moreover, as the reader may easily check, given the information available at each step,

$$
\begin{array}{ll}
(\mathbf{or}_1) & A \to B : (N_1).\ I; A; B; \{\!|N_1; I; A; B|\!\}^s_{K_{AS}} \\
(\mathbf{or}_2) & B \to S : (N_2).\ I; A; B; \{\!|N_1; I; A; B|\!\}^s_{K_{AS}}; \{\!|N_2; I; A; B|\!\}^s_{K_{BS}} \\
(\mathbf{or}_3) & S \to B : (K).\ \ I; \{\!|N_1; K|\!\}^s_{K_{AS}}; \{\!|N_2; K|\!\}^s_{K_{BS}} \\
(\mathbf{or}_4) & B \to A : \qquad I; \{\!|N_1; K|\!\}^s_{K_{AS}}
\end{array}
$$

Fig. 4. The Otway–Rees Authentication/Key-Exchange protocol (OR).

$$
\begin{array}{l}
B\text{-}run : \langle \mathbf{r}(I; A; B; \{\!|N_1; I; A; B|\!\}^s_{K_{AS}}). \\
\qquad\quad \mathbf{f}(N_2). \\
\qquad\quad \mathbf{s}(I; A; B; \{\!|N_1; I; A; B|\!\}^s_{K_{AS}}; \{\!|N_2; I; A; B|\!\}^s_{K_{BS}}, S). \\
\qquad\quad \mathbf{r}(I; \{\!|N_1; K|\!\}^s_{K_{AS}}; \{\!|N_2; K|\!\}^s_{K_{BS}}). \\
\qquad\quad \mathbf{s}(I; \{\!|N_1; K|\!\}^s_{K_{AS}}, A)\rangle
\end{array}
$$

Fig. 5. The responder run of the OR protocol.

the principals can both parse and check the contents of the messages they receive (A and B should each possess their own private key), and construct the messages they need to send (by using the data obtained by decrypting previous messages). The NSPK protocol is thus indeed executable.

Example 9. As another example, in Fig. 4 we present the Alice&Bob specification of the Otway–Rees Authentication/Key-Exchange Protocol (OR) [18]. In this protocol, an initiator A and a responder B attempt to mutually authenticate each other and to exchange a shared key K_{AB} with the help of a server S, with whom they respectively share the keys K_{AS} and K_{BS}. I is a run identifier, contained in each of the four messages.

Directly interpreting the responder role of the Alice&Bob specification of the OR protocol yields the sequence of actions *B-run* shown in Fig. 5 (written vertically for the sake of readability). In contrast to the NSPK example, although the OR protocol is also executable, this sequence does not properly represent the feasible behavior of the responder. The problem concerns B's first action in the run, where he receives $I; A; B; \{\!|N_1; I; A; B|\!\}^s_{K_{AS}}$. As B is not supposed to possess the key K_{AS}, he cannot parse and thereby check that the fourth part of this message is indeed of the form $\{\!|N_1; I; A; B|\!\}^s_{K_{AS}}$. The problem here is not one with the notion of feasibility, but rather it reflects a limitation of the direct interpretation. In this protocol, we must interpret the submessage $\{\!|N_1; I; A; B|\!\}^s_{K_{AS}}$ differently. For B, this submessage is *opaque*: it represents a chunk of information that he cannot decompose but which he should simply forward to S in the second step. In fact, from B's point of view, any message of the form $I; A; B; M$ is acceptable at this point. For similar reasons, the submessage $\{\!|N_1; K_{AB}|\!\}^s_{K_{AS}}$ that B receives in the third step is also opaque and is intended to be forwarded too.

In this example, B will never have the chance to check if the message that he receives from A in his first receiving action is actually of the form prescribed by the protocol. Instead, it is S who must carry out this check later. In other protocols, it may be possible for a principal to receive enough information later on to analyze an opaque message himself.

In general, principals accumulate data incrementally during a protocol run and may only be able to tell if a message received has the required structure (and abort the protocol when this is not the case) at some future protocol step. It is easy to construct an example of this where a principal receives an encrypted message $\{\!|M|\!\}_K$ but only later M and K, or the decryption key. The following protocol provides a concrete example of this phenomenon, using hashes to compute commitments of the protocol participants.

Example 10. The ASW protocol is an optimistic fair-exchange protocol for contract signing proposed by Asokan et al. [5]. Fig. 6 displays (a slightly simplified version of) the Exchange Subprotocol of ASW; for brevity, we will

$$
\begin{array}{ll}
(\mathbf{asw}_1) & A \to B : (N_1).\ \{\!|K_A; K_B; M; H(N_1)|\!\}_{K_A^{-1}} \\
(\mathbf{asw}_2) & B \to A : (N_2).\ \{\!|\{\!|K_A; K_B; M; H(N_1)|\!\}_{K_A^{-1}}; H(N_2)|\!\}_{K_B^{-1}} \\
(\mathbf{asw}_3) & A \to B : \qquad N_1 \\
(\mathbf{asw}_4) & B \to A : \qquad N_2
\end{array}
$$

Fig. 6. The Exchange Subprotocol of the ASW protocol (simplified).

$$
\begin{array}{l}
B\text{-}run : \langle \mathbf{r}(\{\!|K_A; K_B; M; H(N_1)|\!\}_{K_A^{-1}}). \\
\qquad \mathbf{f}(N_2). \\
\qquad \mathbf{s}(\{\!|\{\!|K_A; K_B; M; H(N_1)|\!\}_{K_A^{-1}}; H(N_2)|\!\}_{K_B^{-1}}, A). \\
\qquad \mathbf{r}(N_1). \\
\qquad \mathbf{s}(N_2, A)\rangle
\end{array}
$$

Fig. 7. The responder run of the Exchange Subprotocol of the ASW protocol.

subsequently refer to this protocol as the ASW protocol. The idea is that if two honest principals execute this protocol, and there are neither network failures nor intruder intervention, then afterwards both will possess a valid contract. We write M to denote the contract text, and write $\{\!|M'|\!\}^{\mathrm{a}}_{K_A^{-1}}$ to denote the digital signature of a message M' by the principal A, whose public key for signature verification is K_A. The principals A and B generate nonces N_1 and N_2, which are called their respective *secret commitments* to the contract. Given these, they compute their *public commitments* by hashing these values, yielding $H(N_1)$ and $H(N_2)$, respectively. The protocol then proceeds in two rounds: in the first, each principal expresses his public commitment to the agreed-upon contract but does not disclose his secret commitment; in the second, they exchange their respective secret commitments. Each principal can then hash the secret commitment received and verify that it indeed corresponds to the public commitment from the first round. At the end of this exchange, each principal possesses a valid standard contract of the form

$$\{\!|K_A; K_B; M; H(N_1)|\!\}^{\mathrm{a}}_{K_A^{-1}};\ \{\!|\{\!|K_A; K_B; M; H(N_1)|\!\}^{\mathrm{a}}_{K_A^{-1}}; H(N_2)|\!\}^{\mathrm{a}}_{K_B^{-1}};\ N_1;\ N_2.$$

Fig. 7 displays the responder run *B-run* obtained by the direct interpretation of the ASW protocol. As with the OR protocol, although the ASW protocol is executable, this sequence does not properly represent the feasible behavior of the responder. The problem is that before carrying out the action $\mathbf{r}(N_1)$, the principal B cannot check the structure of the submessage $H(N_1)$, even though he knows the hash function H. Therefore, until N_1 is received, the submessage $H(N_1)$ is again just an opaque chunk of information to be stored. However, upon receiving N_1, B should hash it and abort the protocol execution if it does not coincide with the opaque submessage previously stored. In this case, he should not even execute the last sending action of the run. This kind of situation occurs for several protocols, in particular for those involving commitments to values by principals (e.g., the Zhou-Gollmann protocol, as discussed in [20]).

3. A feasible denotational semantics for Alice&Bob specifications

As illustrated by the examples of the previous section, the direct interpretation does not take into account all of the relevant notions implicit in Alice&Bob specifications, in particular the notions of executable protocol and feasible action. In this section, we will formalize these notions and then introduce a feasible *denotational semantics* for Alice&Bob specifications that accounts for the way that the data possessed by each principal evolves during protocol execution. We proceed as follows. In Section 3.1, we introduce background concepts. In Section 3.2, we present a denotational semantics for Alice&Bob specifications that we call the *fine interpretation*. We also employ the notion of message opacity to formally characterize the class of protocol specifications for which the direct interpretation is appropriate. In Section 3.3, we introduce a *coarse interpretation*, which we show to be appropriate for those protocol specifications

where the evolution of the message views does not require principals to check messages that they previously received. In Section 3.4, we sum up the results of this section by comparing the three interpretations that we have considered.

3.1. Data manipulation and datasets

We begin, as is standard, by defining the sets of messages that principals can *analyze* (decompose) and *synthesize* (compose).

Definition 11. Let S be a set of messages. The set $\mathit{analyz}(S)$ is the least superset of S closed under the rules

$$\frac{M_1;\, M_2}{M_1} \quad \frac{M_1;\, M_2}{M_2} \quad \frac{\{\!|M|\!\}^{\mathrm{a}}_{K} \quad K^{-1}}{M} \quad \frac{\{\!|M|\!\}^{\mathrm{a}}_{K^{-1}} \quad K}{M} \quad \frac{\{\!|M|\!\}^{\mathrm{s}}_{K} \quad K}{M},$$

and the set $\mathit{synth}(S)$ is the least superset of S closed under the rules

$$\frac{M_1 \quad M_2}{M_1;\, M_2} \quad \frac{M \quad K}{\{\!|M|\!\}_K} \quad \frac{M}{H(M)}.$$

The least superset of S closed under both the analysis and synthesis rules is denoted by $\mathit{close}(S)$. Whenever $S = \mathit{close}(S)$, we will call S a *dataset*.

Since we do not consider inverses of inverses $(K^{-1})^{-1}$, as is customary, we have two unambiguous analysis rules for asymmetric encryption: one for decrypting with a private key K^{-1} a message M that has been asymmetrically encrypted with a public key K, and one for decrypting with a public key K a message M that has been asymmetrically encrypted with a private key K^{-1} (as usual, this corresponds to verifying with key K a message that has been signed with key K^{-1}).

It is well-known that if one forbids encryption using composed messages as keys (i.e., if all keys are atomic), then $\mathit{close}(S) = \mathit{synth}(\mathit{analyz}(S))$. In general, however, $\mathit{synth}(\mathit{analyz}(S)) \subsetneq \mathit{close}(S)$, that is, the inclusion is proper. For example, if S contains just the message $M_1; \{\!|M|\!\}^{\mathrm{a}}_{(M_1;M_1)^{-1}}$ then $M \in \mathit{close}(S)$ but $M \notin \mathit{synth}(\mathit{analyz}(S))$. In any case, as shown for instance in [17,19], $\mathit{close}(S)$ is decidable for finite S.

For a given $A \in \mathit{Part}$, assume now that $A\text{-}\mathit{run} = \langle \mathbf{a}_1 \dots \mathbf{a}_s \rangle$. To formalize how the data A possesses evolves along an execution of this run, we must account for both the protocol-relevant data that A starts with and the data that he generates or receives with each protocol action. Since our protocol actions do not model the way that principals synthesize and analyze messages, we will use the datasets to represent the *possessions* of each principal after each action.

Let D^i_A denote the data that A possesses after executing the first i actions of the run. We can then visualize the evolution of A's dataset as follows:

$$D^0_A \xrightarrow{\mathbf{a}_1} D^1_A \xrightarrow{\mathbf{a}_2} D^2_A \xrightarrow{\mathbf{a}_3} \cdots \xrightarrow{\mathbf{a}_{s-1}} D^{s-1}_A \xrightarrow{\mathbf{a}_s} D^s_A.$$

Here D^0_A contains A's initial knowledge. Each D^{i+1}_A is the extension of D^i_A with the new data that A obtains by executing the action $\mathbf{a}_{i+1}$ and taking the closure of the result. When the action is the generation of a number or the reception of a message, then A learns the number generated or the message received. In contrast, A learns nothing new just by sending a message.

Definition 12. The data collected by a principal executing an action $\mathbf{a}$ is the set $\mathit{gets}(\mathbf{a})$ defined by

$$\mathit{gets}(\mathbf{a}) = \begin{cases} \emptyset & \text{if } \mathbf{a} = \mathbf{s}(M, B), \\ \{M\} & \text{if } \mathbf{a} = \mathbf{r}(M), \\ \{N\} & \text{if } \mathbf{a} = \mathbf{f}(N). \end{cases}$$

Given an initial dataset D^0_A, the datasets D^i_A for $1 \leqslant i \leqslant s$ are inductively defined by

$$D^i_A = \mathit{close}(D^{i-1}_A \cup \mathit{gets}(\mathbf{a}_i)).$$

Hence, for every $i > 0$, we have that

$$D_A^i = \begin{cases} D_A^{i-1} & \text{if } \mathbf{a}_i = \mathbf{s}(M, B), \\ close(D_A^{i-1} \cup \{M\}) & \text{if } \mathbf{a}_i = \mathbf{r}(M), \\ close(D_A^{i-1} \cup \{N\}) & \text{if } \mathbf{a}_i = \mathbf{f}(N), \end{cases}$$

and each run defines a (non-strictly) increasing chain $D_A^0 \subseteq D_A^1 \subseteq D_A^2 \subseteq \ldots \subseteq D_A^{s-1} \subseteq D_A^s$ of datasets.

While the messages A sends play no role in constructing the sets D_A^i, it must be the case that A can construct each of these messages using the data available to him. This assumption, which is often left implicit in protocol analysis approaches, can be formalized in the current setting once we fix D_A^0. The initial dataset need only contain the protocol-relevant data that the principal A requires to execute his role of the protocol, such as the identities of relevant participants, their public keys, and A's own private and shared keys, as well as any protocol-specific data such as numbers (e.g., run identifiers as in the Otway–Rees protocol) used as parameters in the protocol run. A precise description of this data is usually missing from Alice&Bob-style protocol specifications. However, it can generally be inferred from the description of the roles themselves. Here we specify requirements on this dataset. [2]

Definition 13. An *initial dataset* for a principal A is a dataset

$$close(\{A, K_A, K_A^{-1}\} \cup Data_A),$$

where $Data_A = Part_A \cup Key_A \cup Num_A$ and
- $Part_A \subseteq Part$,
- $Key_A \subseteq \{K_B, K_{AB} \mid B \in Part_A\}$, and
- Num_A is a set of non-fresh symbols occurring in the specification.

Given these requirements, we can now define executability.

Definition 14. The *role* of A is *executable* provided that there exists an initial dataset D_A^0 for A such that, for every $1 \leqslant i \leqslant s$, if $\mathbf{a}_i = \mathbf{s}(M, B)$ then $M, B \in D_A^{i-1}$. A *protocol specification* is *executable* when all of its roles are.

In the remainder of this paper, we will assume that the protocol specifications that we consider are all executable (for brevity, we will often omit "specification" and simply speak of "an executable protocol"). We will also assume fixed a protocol participant A and an initial dataset D_A^0 that makes A's role executable.

Example 15. For our example protocols NSPK, OR, and ASW, all of which are executable, we fix the initial datasets generated (according to Definition 13) using the following data:
NSPK protocol: $Data_A = \{B, K_B\}$ and $Data_B = \{A, K_A\}$.
OR protocol: $Data_A = \{B, S, K_{AS}, I\}$, $Data_B = \{S, K_{BS}\}$, and $Data_S = \{A, B, K_{AS}, K_{BS}\}$.
ASW protocol: $Data_A = \{B, K_B, M\}$ and $Data_B = \{A, K_A\}$.

3.2. The fine interpretation of Alice&Bob specifications

We now introduce a denotational semantics for Alice&Bob specifications that we call the fine interpretation. Given that each message that is sent in a protocol run must actually be constructed by its sender, we begin by considering the question of how the evolving datasets affect the way that principals parse the messages they receive. As we saw in the examples at the end of the previous section, this data-sensitive inspection must be feasible and plays an essential role in the correct interpretation of Alice&Bob specifications.

Since principals cannot always fully parse the messages they receive, we introduce a set of symbols that represent place-holders: each symbol marks the occurrence of an unparsable message until it becomes parsable. Formally,

[2] This definition can be generalized straightforwardly, e.g., to consider the case where A initially possesses more than one asymmetric key-pair.

we introduce a new atomic *ghost symbol* γ_M for each unparsable message M. These symbols will be used to mark the (sub)messages that are *opaque* to a principal, in the sense that the currently available data does not allow him to parse these (sub)messages and thereby ascertain their precise form and extract their content.

Definition 16. The *view* $v_D(M)$ that a principal has of a message M according to a given dataset D is defined inductively as follows:

- $v_D(M) = M$, if M is atomic;
- $v_D(M_1; M_2) = v_D(M_1); v_D(M_2)$;
- $v_D(\{\!|M_1|\!\}^{\mathrm{a}}_K) = \{\!|v_D(M_1)|\!\}^{\mathrm{a}}_{v_D(K)}$, if $M_1, K \in D$, else
  $v_D(\{\!|M_1|\!\}^{\mathrm{a}}_K) = \{\!|v_D(M_1)|\!\}^{\mathrm{a}}_{\gamma_K}$, if $K^{-1} \in D$, else
  $v_D(\{\!|M_1|\!\}^{\mathrm{a}}_K) = \gamma_M$ with $M = \{\!|M_1|\!\}^{\mathrm{a}}_K$;
- $v_D(\{\!|M_1|\!\}^{\mathrm{a}}_{K^{-1}}) = \{\!|v_D(M_1)|\!\}^{\mathrm{a}}_{v_D(K^{-1})}$, if $M_1, K^{-1} \in D$ or $K \in D$, else
  $v_D(\{\!|M_1|\!\}^{\mathrm{a}}_{K^{-1}}) = \gamma_M$ with $M = \{\!|M_1|\!\}^{\mathrm{a}}_{K^{-1}}$;
- $v_D(\{\!|M_1|\!\}^{\mathrm{s}}_K) = \{\!|v_D(M_1)|\!\}^{\mathrm{s}}_{v_D(K)}$, if $K \in D$, else
  $v_D(\{\!|M_1|\!\}^{\mathrm{s}}_K) = \gamma_M$ with $M = \{\!|M_1|\!\}^{\mathrm{s}}_K$;
- $v_D(K^{-1}) = v_D(K)^{-1}$, if $K \in D$, else
  $v_D(K^{-1}) = \gamma_K^{-1}$, if $\{\!|M_1|\!\}^{\mathrm{a}}_K \in D$ for some M_1, else
  $v_D(K^{-1}) = \gamma_M$ with $M = K^{-1}$;
- $v_D(H(M_1)) = H(v_D(M_1))$, if $M_1 \in D$, else
  $v_D(H(M_1)) = \gamma_M$ with $M = H(M_1)$.

A message M is *D-transparent* if $v_D(M) = M$ and *D-opaque* if $v_D(M) = \gamma_M$. We extend v_D to actions and sequences of actions in the obvious way.

This definition deserves some explanation. First of all, the definition of $v_D(M)$ will be used in the context of a given protocol specification, where D is the dataset of a principal at some point in the execution of his role of the protocol, and M is (a submessage of) a message that he is about to send or has just received, that is, $M \in D$. Moreover, the computation of views is closely related to the synthesis and analysis of messages. Indeed, if M can be synthesized from simpler messages that are in D, then the view of M should correspond to the construction of M from the views of these submessages. This fact explains the cases when M is an atomic message, a pair of messages, a symmetric encryption, or the hash of a message. Namely:

- A pair $M_1; M_2$ can always be analyzed, the views $v_D(M_1)$ and $v_D(M_2)$ of its components computed, and $v_D(M_1; M_2)$ synthesized from them.
- A symmetric encryption $\{\!|M_1|\!\}^{\mathrm{s}}_K$ can only be analyzed by a principal when he knows K; but then he also obtains the message M_1 in the body of the encryption. Thus, the views $v_D(M_1)$ and $v_D(K)$ can be computed and $v_D(\{\!|M_1|\!\}^{\mathrm{s}}_K)$ synthesized from them. Otherwise, the principal's view of the message M is simply the ghost symbol γ_M, i.e., $\gamma_{\{\!|M_1|\!\}^{\mathrm{s}}_K}$.
- Hashed messages are treated similarly, since $H(M_1)$ can only be analyzed by a principal if he has M_1. Then he can compute $v_D(M_1)$ and hash it to obtain $H(v_D(M_1))$. Otherwise, the principal's view of the message is again simply a ghost symbol.

The cases when M is an asymmetric encryption or an inverse key are more complex and are interrelated:

- The case when M is $\{\!|M_1|\!\}^{\mathrm{a}}_{K^{-1}}$ is similar to the cases above when the principal has both M_1 and K^{-1}. In this case, he computes $v_D(M_1)$ and $v_D(K^{-1})$, and then synthesizes $v_D(\{\!|M_1|\!\}^{\mathrm{a}}_{K^{-1}})$ as $\{\!|v_D(M_1)|\!\}^{\mathrm{a}}_{v_D(K^{-1})}$. If the principal only has K, then he can still use it to decrypt the message and obtain M_1. Of course, he cannot be sure about M_1 but he can compute its view $v_D(M_1)$. He can also compute his view of K and use it to compute the view of K^{-1}, that is, $v_D(K^{-1}) = v_D(K)^{-1}$. Consequently, we define $v_D(\{\!|M_1|\!\}^{\mathrm{a}}_{K^{-1}}) = \{\!|v_D(M_1)|\!\}^{\mathrm{a}}_{v_D(K^{-1})} = \{\!|v_D(M_1)|\!\}^{\mathrm{a}}_{v_D(K)^{-1}}$. Otherwise, we again use a ghost symbol.
- The case when M is $\{\!|M_1|\!\}^{\mathrm{a}}_K$ is treated similarly to the cases above if the principal's dataset contains M_1 and K. When this holds, the principal can compute $v_D(M_1)$ and $v_D(K)$, and then synthesize $v_D(\{\!|M_1|\!\}^{\mathrm{a}}_K)$ as $\{\!|v_D(M_1)|\!\}^{\mathrm{a}}_{v_D(K)}$. Even if the principal does not have M_1 and K, he may still be able to view something in $\{\!|M_1|\!\}^{\mathrm{a}}_K$ if he has the

key to decrypt it, i.e., $K^{-1} \in D$. Then, he can decrypt $\{\!|M_1|\!\}^{\mathrm{a}}_K$ to obtain M_1. Of course, he can still not grasp the structure of K, and also he cannot be sure about M_1 unless he has a clear view of it. So, he can compute $v_D(M_1)$, and recognize that $v_D(\{\!|M_1|\!\}^{\mathrm{a}}_K)$ should be $v_D(M_1)$ encrypted under "something". We thus define $v_D(K^{-1}) = \gamma_K^{-1}$ and $v_D(\{\!|M_1|\!\}^{\mathrm{a}}_K) = \{\!|v_D(M_1)|\!\}^{\mathrm{a}}_{\gamma_K}$. If none of these two subcases apply, we again use a ghost symbol to define the view.

It should be clear that the cases of $\{\!|M_1|\!\}^{\mathrm{a}}_K$ and $\{\!|M_1|\!\}^{\mathrm{a}}_{K^{-1}}$ are different when $K^{-1} \in D$ and $K \in D$, respectively. The difference is due to the fact that $v_D(K)$ may be used to compute $v_D(K^{-1})$, since K is a submessage of K^{-1}, but not vice versa. For the sake of clarity, let us consider again what happens with messages of the form K^{-1}.

- If $K \in D$, then the principal can encrypt any message M_1 with K and then decrypt again, using K^{-1}, to recover M_1. Hence, he can confirm that K^{-1} is indeed the inverse of K, and so we define $v_D(K^{-1}) = v_D(K)^{-1}$. If $K \notin D$, then the principal can still use K^{-1} to decrypt any message $\{\!|M_1|\!\}^{\mathrm{a}}_K \in D$. However, he will not be able to analyze K. Hence, we define $v_D(K^{-1}) = \gamma_K^{-1}$ (and also $v_D(\{\!|M_1|\!\}^{\mathrm{a}}_K) = \{\!|v_D(M_1)|\!\}^{\mathrm{a}}_{\gamma_K}$). Otherwise, we again use the corresponding ghost symbol.

It is interesting to note that if we had considered a closure rule like

$$\frac{K^{-1}}{K}$$

then we would simply have $v_D(K^{-1}) = v_D(K)^{-1}$ in all cases. Moreover, we would also have $v_D(\{\!|M_1|\!\}^{\mathrm{a}}_K) = \{\!|v_D(M_1)|\!\}^{\mathrm{a}}_{v_D(K)}$ whenever $\{\!|M_1|\!\}^{\mathrm{a}}_K$ is not D-opaque. We have not considered such a rule, however, because it is admissible, in general, that one can get hold of K^{-1} and not know K.

To illustrate this definition further, let us consider two examples. First, let $D = \mathit{close}(\{\{\!|M_1|\!\}^{\mathrm{a}}_{(N;N)^{-1}}, N; N\})$ for some message M_1 and some number N. Then, the views of the two messages in D are $v_D(\{\!|M_1|\!\}^{\mathrm{a}}_{(N;N)^{-1}}) = \{\!|M_1|\!\}^{\mathrm{a}}_{(N;N)^{-1}}$ and $v_D(N; N) = N; N$. Let now $D' = \mathit{close}(\{\{\!|M_1|\!\}^{\mathrm{a}}_{N;N}, (N; N)^{-1}\})$. The corresponding views are $v_D(\{\!|M_1|\!\}^{\mathrm{a}}_{N;N}) = \{\!|M_1|\!\}^{\mathrm{a}}_{\gamma_{N;N}}$ and $v_D((N; N)^{-1}) = \gamma^{-1}_{N;N}$, since we have no way of viewing the structure of $N; N$. Hence, although $N; N$ by itself can be analyzed, $N; N \notin D$ and thus our view of the messages uses the place-holder $\gamma_{N;N}$ instead.

Note that M is D-opaque when $M = \{\!|M_1|\!\}_K$, $\{M_1, K\} \not\subseteq D$, and the decryption key is also not in D, or when $M = K^{-1}$ and no message $\{\!|M_1|\!\}^{\mathrm{a}}_K$ is in D, or else when $M = H(M_1)$ and $M_1 \notin D$. That is, M is either an encrypted message that cannot be decrypted or constructed, an inverse that cannot be used for asymmetric decryption, or the hash of an unknown message. Clearly, if K^{-1} is D-opaque then $K \notin D$, or otherwise one would also have $\{\!|K|\!\}^{\mathrm{a}}_K \in D$. The converse does not hold in general, that is, $K \notin D$ does not imply that K^{-1} is D-opaque. It is also clear that if a ghost symbol γ_M occurs in $v_D(M')$ then M is a D-opaque submessage of M', or else $M \notin D$ and γ_M occurs only in non-analyzable positions in M', that is, in keys of encrypted messages or in inverse submessages. Note also that views under each of the evolving datasets of a protocol participant can be computed given that these datasets are generated by the closure of finite sets of messages.

Before we define feasibility, and in order to clarify the distinction with respect to executability, we should stress the fact that belonging to the dataset D and being D-transparent are two independent properties of messages. We already know that a message $M \in D$ does not have to be D-transparent, but note also that a D-transparent message does not have to be in D. Namely, $\{\!|M|\!\}^{\mathrm{s}}_K$ is $\mathit{close}(\{K\})$-transparent whenever M is, even if $\{\!|M|\!\}^{\mathrm{s}}_K$ is not in $\mathit{close}(\{K\})$.

Definition 17. Let D be a dataset, $\mathbf{a}$ an action, and $D' = \mathit{close}(D \cup \mathit{gets}(\mathbf{a}))$. We say that $\mathbf{a}$ is D-*feasible* if $v_{D'}(\mathbf{a}) = \mathbf{a}$.

For example, $\mathbf{r}(\{\!|M|\!\}^{\mathrm{a}}_K)$ is not $\emptyset$-feasible, simply because $\{\!|M|\!\}^{\mathrm{a}}_K$ is not $\{\!|M|\!\}^{\mathrm{a}}_K$-transparent.

Clearly, $A\text{-}\mathit{role} = \langle \mathbf{a}_1 \ldots \mathbf{a}_s \rangle$ can be executable even if, for some i, $\mathbf{a}_i$ is not D_A^{i-1}-feasible. This is the case for both the OR and the ASW protocols. Hence, to provide a feasible semantics for Alice&Bob protocol specifications, we must extend the direct interpretation by incorporating this dynamic, data-sensitive notion of feasibility.

Of course, if $D \subseteq D'$ then D'-opaque messages are also D-opaque. The converse, however, is false: a D-opaque message may be parsable (at least partially) using the additional data in D'. This is the key insight behind the interpretation we now suggest. Instead of directly interpreting the run $A\text{-}\mathit{run} = \langle \mathbf{a}_1 \ldots \mathbf{a}_s \rangle$, we will view the execution of A's protocol role as proceeding as described by the diagram in Fig. 8. Hence, $v_{D_A^{k+1}}(\mathbf{a}_i)$ is obtained from $v_{D_A^k}(\mathbf{a}_i)$ by

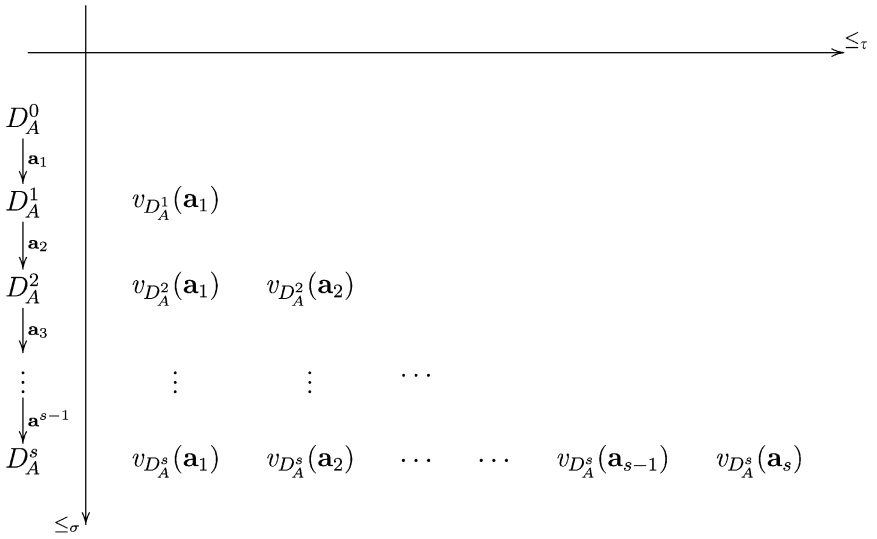


Fig. 8. The execution of A's protocol role.

uniformly replacing all occurrences of ghost symbols γ_M for $M \in D_A^{k+1}$ by $v_{D_A^{k+1}}(M)$. The substitution has no effect when M remains opaque, that is, $v_{D_A^{k+1}}(M) = \gamma_M$.

The sequence composed by the rows between the two axes in the diagram of Fig. 8, i.e., the sequence whose kth element is the sequence

$$\langle v_{D_A^k}(\mathbf{a}_1) \,.\, v_{D_A^k}(\mathbf{a}_2) \ldots v_{D_A^k}(\mathbf{a}_{k-1}) \,.\, v_{D_A^k}(\mathbf{a}_k)\rangle,$$

will be called an *incremental symbolic run* (cf. Definition 18).

At this point, some further explanation is in order. First, the diagram illustrates how actions both serve to extend sequences and instantiate ghost variables. From A's viewpoint, a successful execution of the first step of the protocol corresponds to executing $v_{D_A^1}(\mathbf{a}_1)$. In the resulting sequence, $v_{D_A^1}(\mathbf{a}_1)$ may differ from $\mathbf{a}_1$ in that opaque submessages are replaced by ghost symbols. After execution of the second step, the protocol corresponds to executing the actions $v_{D_A^2}(\mathbf{a}_1)$ and $v_{D_A^2}(\mathbf{a}_2)$ and it is possible that $v_{D_A^2}(\mathbf{a}_1)$ differs from $v_{D_A^1}(\mathbf{a}_1)$ in that some of the ghost symbols occurring in $v_{D_A^1}(\mathbf{a}_1)$ may disappear in $v_{D_A^2}(\mathbf{a}_1)$, having been replaced by more structured patterns. This happens whenever the execution of the second action results in data that allows A to view a previously opaque message in a different way. Hence, the rationale is that, upon executing the second action, A should pick from his backlog of ghost symbols those that correspond to messages that are no longer opaque, and check whether the concretization of each relevant ghost symbol agrees with his new view of the message. When this is not the case, he should abort the protocol. Otherwise, he will have executed the first two actions as $v_{D_A^2}(\mathbf{a}_1)$ and $v_{D_A^2}(\mathbf{a}_2)$. This process is repeated for each executed action.

Formally, upon executing the $(k+1)$th action of his role, for each $M \in D_A^{k+1}$, A should pick each of the γ_M ghost symbols occurring in $\langle v_{D_A^k}(\mathbf{a}_1) \ldots v_{D_A^k}(\mathbf{a}_k)\rangle$ and check if the relevant concrete instantiations of γ_M and $v_{D_A^{k+1}}(M)$ coincide, and abort the protocol when this is not the case. Of course, there is nothing to check if M remains opaque.

Second, our use of the term "symbolic" in naming these runs stems from the ghost symbols, which are not present in the Alice&Bob protocol specifications and hence are also absent from the runs obtained via the direct interpretation. Of course, the runs obtained by the direct interpretation are also symbolic, in the sense that they contain symbols corresponding to parameters that must be instantiated in order to obtain concrete runs. However, the structure of the terms with which they are instantiated is irrelevant, whereas ghost symbols can be instantiated with concrete messages whose inner structure is protocol relevant. In this paper, we will use the term symbolic to refer to terms and runs that may contain ghost symbols.

$B\text{-}isrun$:

$$\begin{array}{l}
\langle \mathbf{r}(\{\!|K_A; K_B; M; \gamma|\!\}^a_{K_A^{-1}})\rangle \\
\langle \mathbf{r}(\{\!|K_A; K_B; M; \gamma|\!\}^a_{K_A^{-1}}) . \quad \mathbf{f}(N_2)\rangle \\
\langle \mathbf{r}(\{\!|K_A; K_B; M; \gamma|\!\}^a_{K_A^{-1}}) . \quad \mathbf{f}(N_2) .\mathbf{s}(\{\!|\{\!|K_A; K_B; M; \gamma|\!\}^a_{K_A^{-1}}; \quad H(N_2)|\!\}^a_{K_B^{-1}}, A)\rangle \\
\langle \mathbf{r}(\{\!|K_A; K_B; M; H(N_1)|\!\}^a_{K_A^{-1}}) . \quad \mathbf{f}(N_2) .\mathbf{s}(\{\!|\{\!|K_A; K_B; M; H(N_1)|\!\}^a_{K_A^{-1}}; \quad H(N_2)|\!\}^a_{K_B^{-1}}, A) .\mathbf{r}(N_1)\rangle \\
\langle \mathbf{r}(\{\!|K_A; K_B; M; H(N_1)|\!\}^a_{K_A^{-1}}) . \quad \mathbf{f}(N_2) .\mathbf{s}(\{\!|\{\!|K_A; K_B; M; H(N_1)|\!\}^a_{K_A^{-1}}; \quad H(N_2)|\!\}^a_{K_B^{-1}}, A) .\mathbf{r}(N_1) . \quad \mathbf{s}(N_2, A)\rangle
\end{array}$$

Fig. 9. The incremental symbolic responder runs of the Exchange Subprotocol of the ASW protocol.

$B\text{-}isrun$:

$$\begin{array}{l}
\langle \mathbf{r}(\{\!|N_1; A|\!\}^a_{K_B})\rangle \\
\langle \mathbf{r}(\{\!|N_1; A|\!\}^a_{K_B}) . \quad \mathbf{f}(N_2)\rangle \\
\langle \mathbf{r}(\{\!|N_1; A|\!\}^a_{K_B}) . \quad \mathbf{f}(N_2) .\mathbf{s}(\{\!|N_1; N_2|\!\}^a_{K_A}, A)\rangle \\
\langle \mathbf{r}(\{\!|N_1; A|\!\}^a_{K_B}) . \quad \mathbf{f}(N_2) .\mathbf{s}(\{\!|N_1; N_2|\!\}^a_{K_A}, A) . \quad \mathbf{r}(\{\!|N_2|\!\}^a_{K_B})\rangle
\end{array}$$

Fig. 10. The incremental symbolic responder runs of the NSPK protocol.

Finally, the notion of symbolic sequences being extended and instantiated by actions can be formally described in terms of orderings: an *instantiation ordering* $\leqslant_\sigma$, a *prefix ordering* $\leqslant_\tau$, and their *composition* $\leqslant_{\tau\sigma}$. Given two symbolic sequences u and v, we say that $u \leqslant_\sigma v$ iff v is a substitution instance of u, i.e., there exists a substitution σ such that $\sigma(u) = v$. Moreover, we say that $u \leqslant_\tau v$ iff u is a prefix of v, i.e., the sequence v extends u. Finally, we define $\leqslant_{\tau\sigma} = \leqslant_\tau \circ \leqslant_\sigma$, i.e., $u \leqslant_{\tau\sigma} v$ iff there is a symbolic sequence w such that $u \leqslant_\sigma w$ and $w \leqslant_\tau v$. In other words, there is a substitution instance w of u that can be extended to v, or simply, v extends a substitution instance of u. Fig. 8 shows the two dimensions induced by the instantiation and prefix orderings: the symbolic runs grow left-to-right according to $\leqslant_\tau$ and top-down according to $\leqslant_\sigma$. That is, each symbolic sequence extends its predecessor and possibly instantiates some of its ghost symbols. Hence, in an incremental symbolic run $\langle t_1 . t_2 \ldots t_n\rangle$, we have $t_i \leqslant_{\tau\sigma} t_{i+1}$, for all i with $1 \leqslant i < n$.

We now introduce the fine interpretation of Alice&Bob protocol specifications in terms of incremental symbolic runs that characterize the possible behavior of each of the participants in a (possibly incomplete) protocol execution.

Definition 18. Let $A \in Part$. In the *fine interpretation* of Alice&Bob protocol specifications, the *incremental symbolic run* corresponding to the execution of A's role in the protocol is

$$A\text{-}isrun = \langle A\text{-}isrun^1 \ldots A\text{-}isrun^s\rangle,$$

where $A\text{-}isrun^i = v_{D_A^i}(A\text{-}run|_i)$ for $1 \leqslant i \leqslant s = |A\text{-}run|$.

Example 19. Under the fine interpretation, the responder run of the ASW protocol is, instead of the $B\text{-}run$ of Fig. 7, the incremental symbolic run $B\text{-}isrun$ (i.e., the sequence of subruns, of growing length) displayed in Fig. 9. Here, for simplicity, we write γ instead of $\gamma_{H(N_1)}$.

This example illustrates how the new data available to the principal executing a role may turn opaque elements into non-opaque ones. That happens if $\mathbf{a}_i$ is a receiving action taken by the protocol participant B, γ_M occurs in $v_{D_B^{i-1}}(\mathbf{a}_j)$ for some $j < i$, and γ_M does not occur in $v_{D_B^i}(\mathbf{a}_j)$. When this happens, B should abort the execution of his role of the protocol whenever the actual value of γ_M does not match $v_{D_B^i}(M)$.

Note, however, that, in cases like NSPK, the direct interpretation appears to be appropriate. Indeed, as shown in Fig. 10, the direct runs are *representative* in the sense that each sequence of the fine interpretation coincides with the prefix of corresponding length of the direct run, that is, each $A\text{-}isrun^i = A\text{-}run|_i$. This happens because the direct run is feasible since all the messages involved are transparent, and hence ghost symbols and corresponding checks are

B-*isrun* :

$\langle \mathbf{r}(I;A;B;\gamma_1)\rangle$
$\langle \mathbf{r}(I;A;B;\gamma_1)\,.\mathbf{f}(N_2)\rangle$
$\langle \mathbf{r}(I;A;B;\gamma_1)\,.\mathbf{f}(N_2)\,.\ \ \mathbf{s}(I;A;B;\gamma_1;\{\!|N_2;I;A;B|\!\}^s_{K_{BS}},S)\rangle$
$\langle \mathbf{r}(I;A;B;\gamma_1)\,.\mathbf{f}(N_2)\,.\ \ \mathbf{s}(I;A;B;\gamma_1;\{\!|N_2;I;A;B|\!\}^s_{K_{BS}},S)\,.\mathbf{r}(I;\gamma_2;\{\!|N_2;K|\!\}^s_{K_{BS}})\rangle$
$\langle \mathbf{r}(I;A;B;\gamma_1)\,.\mathbf{f}(N_2)\,.\ \ \mathbf{s}(I;A;B;\gamma_1;\{\!|N_2;I;A;B|\!\}^s_{K_{BS}},S)\,.\mathbf{r}(I;\gamma_2;\{\!|N_2;K|\!\}^s_{K_{BS}})\,.\ \ \mathbf{s}(I;\gamma_2,A)\rangle$

Fig. 11. The incremental symbolic responder runs of the OR protocol.

not necessary. We can give a precise characterization of the protocol descriptions for which the direct interpretation is appropriate:

Proposition 20. *The sequence A-run is representative iff every received message is transparent when it is received, i.e., if* $\mathbf{a}_i = \mathbf{r}(M)$, *then M is* D^i_A*-transparent.*

3.3. The coarse interpretation of Alice&Bob specifications

While the ASW protocol clearly demonstrates the importance of the fine interpretation, cases such as that of the OR protocol (for which the direct interpretation is not feasible) still maintain some of the regularity of the simplest cases. Indeed, although ghost symbols are necessary, there is no need for abortion checks since opaque messages always remain opaque. Hence, the sequence $\langle v_{D^1_A}(\mathbf{a}_1)\ldots v_{D^s_A}(\mathbf{a}_s)\rangle$ in the diagonal of the incremental symbolic run obtained by the fine interpretation, besides being still feasible, is also representative. Namely, each $A\text{-}isrun^i = \langle v_{D^1_A}(\mathbf{a}_1)\ldots v_{D^i_A}(\mathbf{a}_i)\rangle$. For these cases, a coarse interpretation in terms of a unique sequence of actions with ghost symbols but no checks for previous messages seems to be appropriate. We will call such a sequence a *symbolic run*, in contrast to "incremental" symbolic runs.

Definition 21. Let $A \in Part$ and $A\text{-}run = \langle \mathbf{a}_1 \ldots \mathbf{a}_s\rangle$. In the *coarse interpretation* of Alice&Bob protocol specifications, the *symbolic run* corresponding to the execution of A's role in the protocol is

$$A\text{-}srun = \langle v_{D^1_A}(\mathbf{a}_1)\ldots v_{D^s_A}(\mathbf{a}_s)\rangle\,.$$

Fig. 11 shows the incremental symbolic responder runs B-*isrun* of the OR protocol, whose diagonal is the symbolic run B-*srun*, where, for simplicity, we simply write γ_1 and γ_2 instead of $\gamma_{\{\!|N_1;I;A;B|\!\}^s_{K_{AS}}}$ and $\gamma_{\{\!|N_1;K|\!\}^s_{K_{AS}}}$, respectively.

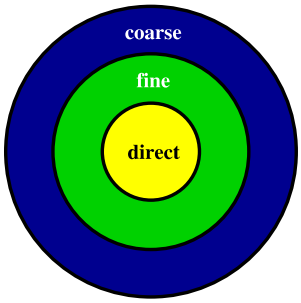
A precise characterization of the protocol specifications for which the coarse interpretation is appropriate can also be obtained (by considering the different possible actions):

Proposition 22. *The sequence A-srun is representative iff every received message preserves the ghost symbols that occur in the views of previously received messages, i.e., if* $1 \leqslant j < i \leqslant s$, $\mathbf{a}_j$ *and* $\mathbf{a}_i$ *are receiving actions, and* γ_M *occurs in* $v_{D^{i-1}_A}(\mathbf{a}_j)$, *then* γ_M *also occurs in* $v_{D^i_A}(\mathbf{a}_j)$.

Note that in this case, only message forwarding may be necessary. The precise meaning of *forwarding* is now well understood: if a sent message contains an opaque submessage M then M must also occur, and be opaque, in some previously received message. Note also that Proposition 20 can be seen as a corollary of Proposition 22.

3.4. Summary: the denotational semantic spectrum of Alice&Bob specifications

Fig. 12 sums up the results of this section, displaying, in general, the relationship between the sets of all possible concrete behaviors for each protocol participant determined by the three interpretations: direct, fine, and coarse. In the direct and coarse interpretations, the allowed (possibly incomplete) behaviors for each participant correspond to all prefixes of all concrete instantiations of the run or symbolic run, respectively. In the fine interpretation, the allowed behaviors correspond to all concrete instantiations of each of the incremental symbolic runs. Hence, the direct



Semantics	Interpretation domain	Interpretation function
Direct interpretation	runs	A-run
Fine interpretation	incremental symbolic runs	A-$isrun$
Coarse interpretation	symbolic runs	A-$srun$

Fig. 12. Direct, fine and coarse interpretations.

interpretation is the strictest, although it is not always feasible. Clearly, all the allowed behaviors obtained by the direct interpretation can be seen as particular cases of the fine interpretation where all the ghost symbols are concretized exactly by the intended messages, in which case all the checks involved will succeed. The inclusion is, in general, not an equality (cf. the OR and ASW protocols), but the two coincide precisely for those protocols that, like NSPK, meet the conditions of Proposition 20. The allowed behaviors obtained by the fine interpretation can also be obtained for the coarse interpretation if one just ignores all the checks. Again the inclusion is not, in general, an equality (cf. the ASW protocol) because some of the checks that are ignored may actually lead to abortion in the fine interpretation. The two coincide for those protocols, like OR, that fulfill the conditions of Proposition 22.

The above differences have a clear relevance for protocol analysis models and protocol correctness results since considering different interpretations may lead to conflicting results. In general, a protocol correctness result obtained for the direct interpretation is not very meaningful because the fine interpretation may introduce further behaviors that violate the relevant security properties. Similarly, an attack to a security property obtained by means of a coarse interpretation of the protocol may also not have a counterpart in terms of the behaviors allowed by the fine interpretation. In contrast, attacks obtained under the direct interpretation, as well as correctness results obtained under the coarse interpretation, do carry over to the fine interpretation.

4. An operational semantics for Alice&Bob specifications

In this section, we formalize an executable operational counterpart of the fine interpretation of Alice&Bob protocol specifications in terms of incremental symbolic runs. To this end, we will use (a fragment of) the *pattern-matching spi calculus* [28], which is an extension of the spi calculus [1] that includes pattern matching as a primitive. We could have used other formalisms, e.g., formalisms based on process-algebra such as LySa [9] or suitable extensions of CASPER [32], or formalisms based on multiset-rewriting as in [31], or even conventional programming languages. However, explicit pattern matching makes the pattern-matching spi calculus particularly well-suited for our purposes: pattern matching allows us to formalize that principals accept messages they receive only when these messages match the specified patterns.

4.1. Pattern-matching spi calculus

Strictly speaking, we will actually use a slight variant of the pattern-matching spi calculus. In [28], as is usual in spi calculi, messages are modeled as elements of an algebraic datatype. Since there is no essential difference, we will simplify matters by sticking to our previous definition of messages (cf. Section 2). Moreover, we will also assume that there exists an untrusted communication channel net_A for each principal A, instead of a global untrusted channel net. The idea is that whenever a honest principal A sends a message to another principal B, A should deliver it to the channel net_B. Also, if A is honest, then A will only receive messages from net_A. Furthermore, the pattern-matching spi calculus is a typed language, but for our purposes we will consider just a type Num of numbers (without differentiating between numbers that are keys, nonces, etc.) and a generic type Msg of messages.

Formally, we will work with the following fragment of the full pattern-matching spi calculus of [28]. This fragment defines a set of sequential *processes* that we will use to *implement* each of the roles of a given Alice&Bob specification.

$P ::=$	processes
$out\ N\ M;\ P$	asynchronous output of message M on message N
$inp\ N\ T;\ P$	input from message N against pattern T
$new\ N : T;\ P$	generation of name N of type T
0	inactivity

Patterns T are of the form

$$(\exists \overline{X})\ M\,,$$

where M is a message and $\overline{X} = \{X_1, \ldots, X_n\}$ is a set of *symbols*. We will often use X, Y, and Z to denote arbitrary symbols. Note also that, as is usual, we will typically omit the 0 at the end of a sequential composition and simply write P instead of $P;\ 0$.[3]

The meaning of these constructs, as explained in [28], is as follows:

- $out\ N\ M;\ P$ is a process that asynchronously outputs the message M on (the channel) N, and proceeds as P.
- $inp\ N\ (\exists \overline{X})\ M;\ P$ is a process that inputs from (the channel) N a message matching the pattern M (where the symbols in $\overline{X}$ occur), and proceeds as P, where the symbols in $\overline{X}$ are instantiated with their actual values according to the pattern-matched input message.
- $new\ N : T;\ P$ is a process that creates a new name N of type T, which is bound in the continuation P.

As notation, we denote by $[X_1 := M_1, \ldots, X_n := M_n]$ the *substitution function* that uniformly replaces every occurrence of a symbol X_i with M_i, where the trivial substitution [] is the identity and we denote composition of substitutions by $\circ$. We can then explain more rigorously the input–output behavior of the spi processes: if processes

$$out\ N\ M[X_1 := M_1, \ldots, X_n := M_n];\ P \quad \text{and} \quad inp\ N\ (\exists X_1, \ldots, X_n)\ M;\ Q$$

coexist, in parallel, then they can both evolve respectively into

$$P \quad \text{and} \quad Q[X_1 := M_1, \ldots, X_n := M_n]\,.$$

In [28], a number of syntactic restrictions are imposed on the messages and patterns involved. In $out\ N\ M$, it must be the case that M is an *implementable message*. Similarly, in $inp\ N\ (\exists \overline{X})\ M$, it must be the case that $(\exists \overline{X})\ M$ is an *implementable pattern*. Also, the N, in both notions, must be an implementable message (in our case, it will always be a channel or a locally bounded message symbol). These notions of implementability are closely related to our notions of executability and feasibility (cf. Section 3), respectively, as some of the underlying ideas are the same. However, while implementability is context-independent and depends only on the message or pattern at hand, our notions depend in an essential way on the context where the message (or pattern) appears. Hence, rather than following [28], we will introduce our own notions, based on adapting the notions of executability and feasibility to spi-calculus processes.

From here on, we will only consider processes that are built by sequential composition of blocks of the following four kinds.

new-block: $new\ N : Num$.
out-block: $out\ net_B\ M$.
inp-block: $inp\ net_A\ (\exists \overline{X})\ M$.
chk-block: $chk(Y_1; \ldots; Y_m \mid (\exists \overline{X})\ M_1; \ldots; M_m) = new\ tmp : Msg;\ out\ tmp\ Y_1; \ldots; Y_m;\ inp\ tmp\ (\exists \overline{X})\ M_1; \ldots; M_m$.

The first three kinds of block are straightforward, while the fourth kind requires more explanation. To implement the symbolic checks for abortion (namely, to check if one or more patterns matches one or more messages), we will use processes of the form $chk(Y_1; \ldots; Y_m \mid (\exists \overline{X})\ M_1; \ldots; M_m)$, where $Y_1, \ldots, Y_m$ are symbols that do not occur in $M_1; \ldots; M_m$. This process executes successfully whenever each Y_i matches M_i. Note that *tmp* is just a local name

[3] Usually, in spi calculi, output processes are not allowed to have a continuation by sequential composition. However, we are adopting here the standard convention that $out\ N\ M;\ P$ is indeed the parallel composition of $out\ N\ M$ and P (cf. [27]). Although we do not explicitly introduce it, parallel composition is another construct of spi calculi.

through which the matching will take place. If the match fails, the process will just deadlock (due to the design of the process calculus, there is no explicit way of raising an abortion).

We now introduce our notion of implementability. As previously noted, implementability is not a static notion as it depends in an essential way on the basic set of messages possessed by the principal at each point in the execution of his process. We assume that, starting from the initial dataset that is given as a parameter of the process, these possessions grow during execution by collecting every newly generated number, as well as all existentially quantified symbols and inverse submessages appearing in analyzable positions in blocks of kind chk-block. Note that by an *analyzable position* we mean an occurrence in a submessage that can be reached using the analysis rules, that is, we do not count the symbols occurring in K in submessages of the form $\{\!|M|\!\}_K$ or K^{-1}. We will call such a set of messages a *basic set*. Note also that, by definition, a basic set is always closed for analysis, and therefore its closure can be obtained just by closing it under the synthesis rules. It is important to stress that pattern-matches mark precisely the learning process of the principal, together with the collection of all freshly generated numbers.

Let χ be a symbol set. We define an *implementable pattern* to be a pattern that is transparent with respect to χ, corresponding to a feasible receiving action and where all symbols not in χ are existentially quantified and thus added to χ as long as they occur in analyzable positions. This means that, given a transparent message M, the unique corresponding implementable pattern is $(\exists X_1, \ldots, X_n)\ M$, where $X_1, \ldots, X_n$ are the symbols not in χ that occur in M. For brevity, we may simply write $(\exists_\chi)M$ for such an $(\exists X_1, \ldots, X_n)\ M$. Note that, although we do not quantify over them, all inverse messages that occur in M in analyzable positions must also be stored.

Moreover, we adopt a notion of implementable message that is explicitly *constructive*. In general, when a principal sends a message, he must be able to build it. This is what we have called executability. However, executability itself does not immediately yield a recipe to actually produce the message since, in general, there may be many ways of doing it. We provide a solution to this problem, obtaining an executable process that precisely prescribes how to build all the messages that need to be sent. In general, given χ, an *implementable message* is one that can be synthesized directly from χ.

Note that, by pattern matching, a symbol that appeared earlier in the process may be instantiated with a more concrete message. We will also build the corresponding substitution incrementally, starting from the trivial substitution []. We further assume that as soon as A knows the name of another principal B, then A also knows the channel net_B, for communicating with B.

For our purposes, it is convenient to use as symbols the atomic messages (identifiers and number symbols) and the ghost symbols γ_M. Hence, let us fix a process $P_A(Data_A) = Q_1; \ldots; Q_k$ for principal A, where each Q_i is a process of kind new-block, out-block, inp-block, or chk-block. We will define the incremental sequence of symbol sets $\chi_0, \chi_1, \ldots, \chi_k$ possessed by A during the execution of the process, along with the corresponding sequence of substitutions $\sigma_0, \sigma_1, \ldots, \sigma_k$, and use them to state the necessary implementability conditions for the process. The rationale is that if a principal A possesses a symbol set χ_{i-1}, with corresponding substitution σ_{i-1}, and executes Q_i, then he obtains a symbol set χ_i and substitution σ_i, but only provided that the implementability condition is fulfilled. More specifically, we have the following cases, depending on Q_i:

- Initially, $\chi_0 = Data_A$ and $\sigma_0 = []$;
- For each $i = 1, \ldots, k$ we have:
  - If $Q_i = new\ N : Num$, then $\chi_i = \chi_{i-1} \cup \{N\}$ and $\sigma_i = \sigma_{i-1}$. The condition for implementability is $N \notin sub(\chi_{i-1})$.
  - If $Q_i = out\ net_B\ M$, then $\chi_i = \chi_{i-1}$ and $\sigma_i = \sigma_{i-1}$. The condition for implementability is $B, M \in synth(\chi_{i-1})$.
  - If $Q_i = inp\ net_A\ (\exists \overline{X})M$, then $\chi_i = \chi_{i-1} \cup \{X \in \overline{X} \mid X \text{ analyzable in } M\} \cup \{K_1^{-1}, \ldots, K_j^{-1}\}$, where $K_1^{-1}, \ldots, K_j^{-1}$ are the inverse messages occurring in analyzable positions of M, and $\sigma_i = \sigma_{i-1}$. The conditions for implementability are that M is $synth(\chi_i)$-transparent and that $\overline{X}$ consists of the new symbols in M determined according to χ_{i-1}.
  - If $Q_i = chk(Y_1; \ldots; Y_m \mid (\exists \overline{X})\ M_1; \ldots; M_m)$, then $\chi_i = \chi_{i-1} \cup \{X \in \overline{X} \mid X \text{ analyzable in } M_1; \ldots; M_m\} \cup \{K_1^{-1}, \ldots, K_j^{-1}\}$, where $K_1^{-1}, \ldots, K_j^{-1}$ are the inverse messages occurring in analyzable positions of M, and $\sigma_i = [Y_1 := \sigma_{i-1}(M_1), \ldots, Y_m := \sigma_{i-1}(M_m)] \circ \sigma_{i-1}$. The conditions for implementability are that $M_1; \ldots; M_m$ is $synth(\chi_i)$-transparent and $\overline{X}$ consists of the new analyzable symbols in $M_1; \ldots; M_m$ determined according to χ_{i-1}.

4.2. Spi-calculus processes for Alice&Bob specifications

We now formalize an executable operational semantics for Alice&Bob protocol specifications based on the pattern-matching spi calculus that corresponds to the denotational semantics provided by our fine interpretation. In this semantics, we assign to each protocol role a sequential process that provides an actual implementation of the corresponding incremental symbolic run. The full protocol process is then obtained by the parallel composition of all these sequential processes with a suitable environment process.

To guarantee that all sent messages can be synthesized from the symbols collected along the execution of the protocol process, it is essential that principals engage in a careful analysis of the messages they receive. We now formalize how received messages must be parsed and their analyzable constituents stored for future use. First of all, a message has what we call a *facial pattern*, where encrypted, inverted, or hashed submessages are not parsed, but simply "stored" in a suitable ghost symbol, which serves as a macro for them.

Definition 23. The *facial pattern* $face(M)$ that a principal A sees of a message M is defined inductively as follows:
- $face(M) = M$, if M is atomic;
- $face(M_1; M_2) = face(M_1); face(M_2)$; and
- $face(M) = \gamma_M$, otherwise.

When the messages corresponding to each of the ghost symbols introduced in the definition of *face* are not opaque, one can then proceed stepwise and parse their content. We postpone the details of this parsing (to Definition 25) and define now the constructive form of a constructible message to be sent, with respect to the current set of symbols.

Definition 24. Given a set of symbols χ, the *constructive form* $cf_\chi(M)$ that a principal has of synthesizing (his view of) a message M is defined inductively as follows:
- $cf_\chi(M) = M$, if M is atomic;
- $cf_\chi(M_1; M_2) = cf_\chi(M_1); cf_\chi(M_2)$;
- $cf_\chi(\{\!|M_1|\!\}_K) = \{\!|cf_\chi(M_1)|\!\}_{cf_\chi(K)}$, if $cf_\chi(M_1), cf_\chi(K) \in synth(\chi)$, else
  $cf_\chi(\{\!|M_1|\!\}_K) = \gamma_M$ with $M = \{\!|M_1|\!\}_K$;
- $cf_\chi(K^{-1}) = face(K)^{-1}$, if $face(K)^{-1} \in \chi$, else
  $cf_\chi(K^{-1}) = \gamma_K^{-1}$, if $\gamma_K^{-1} \in \chi$, else
  $cf_\chi(K^{-1}) = \gamma_M$ with $M = K^{-1}$;
- $cf_\chi(H(M_1)) = H(cf_\chi(M_1))$, if $cf_\chi(M_1) \in synth(\chi)$, else
  $cf_\chi(H(M_1)) = \gamma_M$ with $M = H(M_1)$.

The constructive form of a message with respect to a set of symbols is closely related to message synthesis. If M can be constructed from simpler messages that are in $synth(\chi)$, then $cf_\chi(M)$ should correspond to the synthesis of M from (the constructive form of) these submessages. Otherwise, we use the corresponding ghost symbol γ_M. Note, in particular, that $cf_\chi(K^{-1})$ is given, when available, by the best possible representative of the message in χ, that is, $face(K)^{-1}$ or γ_K^{-1} if the former is not available. This explains why we need to store some inverse messages in the basic sets, given that inverse messages cannot be synthesized.

In order to check the content of a given symbol γ_M against M, we define now a constructive version of *view*.

Definition 25. Given a symbol set χ, an encrypted, inverse, or hashed message M can be *parsed* by a principal A whenever the corresponding *inner facial pattern* $inface_\chi(M)$ is defined, where
- $inface_\chi(\{\!|M_1|\!\}^{\mathrm{a}}_K) = \{\!|cf_\chi(M_1)|\!\}^{\mathrm{a}}_{cf_\chi(K)}$, if $cf_\chi(M_1), cf_\chi(K) \in synth(\chi)$, else
  $inface_\chi(\{\!|M_1|\!\}^{\mathrm{a}}_K) = \{\!|face(M_1)|\!\}^{\mathrm{a}}_{face(K)}$, if $face(K)^{-1} \in \chi$, else
  $inface_\chi(\{\!|M_1|\!\}^{\mathrm{a}}_K) = \{\!|face(M_1)|\!\}^{\mathrm{a}}_{\gamma_K}$, if $\gamma_K^{-1} \in \chi$, else
  $inface_\chi(\{\!|M_1|\!\}^{\mathrm{a}}_K)$ is undefined;

- $\mathit{inface}_\chi(\{\!|M_1|\!\}^{\mathrm{a}}_{K^{-1}}) = \{\!|\mathit{cf}_\chi(M_1)|\!\}^{\mathrm{a}}_{\mathit{cf}_\chi(K^{-1})}$, if $\mathit{cf}_\chi(M_1), \mathit{cf}_\chi(K^{-1}) \in \mathit{synth}(\chi)$, else
  $\mathit{inface}_\chi(\{\!|M_1|\!\}^{\mathrm{a}}_{K^{-1}}) = \{\!|\mathit{face}(M_1)|\!\}^{\mathrm{a}}_{\mathit{cf}_\chi(K)^{-1}}$, if $\mathit{cf}_\chi(K) \in \mathit{synth}(\chi)$, else
  $\mathit{inface}_\chi(\{\!|M_1|\!\}^{\mathrm{a}}_{K^{-1}})$ is undefined;
- $\mathit{inface}_\chi(\{\!|M_1|\!\}^{\mathrm{s}}_{K}) = \{\!|\mathit{face}(M_1)|\!\}^{\mathrm{s}}_{\mathit{cf}_\chi(K)}$, if $\mathit{cf}_\chi(K) \in \mathit{synth}(\chi)$, else
  $\mathit{inface}_\chi(\{\!|M_1|\!\}^{\mathrm{s}}_{K})$ is undefined;
- $\mathit{inface}_\chi(K^{-1}) = \mathit{cf}_\chi(K)^{-1}$, if $\mathit{cf}_\chi(K) \in \mathit{synth}(\chi)$, else
  $\mathit{inface}_\chi(K^{-1})$ is undefined;
- $\mathit{inface}_\chi(H(M_1)) = H(\mathit{cf}_\chi(M_1))$, if $\mathit{cf}_\chi(M_1) \in \mathit{synth}(\chi)$, else
  $\mathit{inface}_\chi(H(M_1))$ is undefined.

Clearly, $\mathit{inface}_\chi(M)$ mimics the view $v_D(M)$ of M, but using now a basic set and what can be synthesized from it, instead of a dataset. If the message cannot be analyzed given the information available, the result will be undefined. Otherwise, it will result in the immediate inner pattern of M, which is thus opened for further analysis.

Now we can finally define the spi-calculus implementation of each of the roles of an Alice&Bob protocol specification. Each received message is decomposed according to the definition of *face*, whose ghost symbols are stored for further checking. Then, each available ghost symbol is checked, in a stepwise manner, according to the definition of *inface*. Finally, the messages to be sent are then synthesized from the data items available, in accordance with the definition of *cf*. Let $A\text{-}run = \langle \mathbf{a}_1 \dots \mathbf{a}_s \rangle$ be obtained for a participant A in an Alice&Bob protocol specification as defined in Section 2. Since the translation will be inductive, we can rely on the symbol sets $\chi_0, \dots, \chi_{\mathrm{s}}$ collected along the execution of the process, where χ_k is collected up to the block corresponding to action $\mathbf{a}_k$.

Definition 26. Let $A \in \mathit{Part}$. The spi-calculus sequential process $A\text{-}proc$ that *implements* the execution of A's role for a parameter set Data_A is defined inductively by

$$P_A(\mathit{Data}_A) = \mathit{spi}^A_1(\mathbf{a}_1);\ \dots\ ;\ \mathit{spi}^A_s(\mathbf{a}_s)\,,$$

with
- $\mathit{spi}^A_{k+1}(\mathbf{f}(N)) = \mathit{new}\ N : \mathit{Num}$,
- $\mathit{spi}^A_{k+1}(\mathbf{s}(M, B)) = \mathit{out}\ \mathit{net}_B\ \mathit{cf}_{\chi_k}(M)$, and
- $\mathit{spi}^A_{k+1}(\mathbf{r}(M)) = \mathit{inp}\ \mathit{net}_A\ (\exists_{\chi_k})\ \mathit{face}(M);\ \mathit{chk}_1;\ \dots\ ;\ \mathit{chk}_t$,

where, using χ^i_k to denote the basic set that extends χ_k with the analyzable existentially quantified symbols and inverse messages collected and updated up to $\mathit{inp}\ \mathit{net}_A\ (\exists_{\chi_k})\ \mathit{face}(M);\ \dots\ ;\ \mathit{chk}_{i-1}$, we have that
- for each i, either
  - $\mathit{chk}_i = \mathit{chk}(\gamma_{M'} \mid (\exists_{\chi^i_k})\ \mathit{inface}_{\chi^i_k}(M'))$, for some previously unchecked symbol $\gamma_{M'} \in \chi^i_k$ such that M' can be parsed, or
  - $\mathit{chk}_i = \mathit{chk}(\gamma_{K^{-1}}; \gamma_{\{\!|M'|\!\}^{\mathrm{a}}_K} \mid (\exists_{\chi^i_k})\ \gamma_K^{-1}; \{\!|\mathit{face}(M')|\!\}^{\mathrm{a}}_{\gamma_K})$, for some previously unchecked symbols $\gamma_{K^{-1}}, \gamma_{\{\!|M'|\!\}^{\mathrm{a}}_K} \in \chi^i_k$, or
  - $\mathit{chk}_i = \mathit{chk}(\gamma_K \mid (\exists_{\chi^i_k})\ \mathit{cf}_{\chi^i_k}(K))$, for some previously unchecked symbol $\gamma_K \notin \chi^i_k$ such that $\gamma_K^{-1} \in \chi^i_k$ and $\mathit{cf}_{\chi^i_k}(K) \in \mathit{synth}(\chi^i_k)$, and
- no further checks are possible.

Let us explain the three kinds of possible checks. In the first case, we are checking $\gamma_{M'}$, where M' is an encrypted, hashed, or inverse message, according to its inner face pattern, whenever it is parsable. The second case corresponds to the situation where a principal has $\{\!|M'|\!\}^{\mathrm{a}}_K$ and K^{-1} and none of these messages can be parsed given the rest of the available information, although they can clearly be parsed jointly since one is the key to decrypt the other, thus yielding the check that we introduce. Finally, checks of the third kind serve to refine the view that a principal has of a key K, if it ever becomes available, after he gets K^{-1}.

Our running examples (the NSPK, OR, and ASW protocols) only require very simple checks of the first kind (see Example 29). Hence, we now introduce some small, artificial examples that illustrate checks of the first kind on nested messages, as well as checks of the second and third kinds.

First, suppose that the initial dataset of principal A contains K_A^{-1} and K_B and his role is of the form $\langle \mathbf{r}(\{\!|\{\!|N|\!\}^{\mathrm{a}}_{K_B^{-1}}; H(N)|\!\}^{\mathrm{a}}_{K_A}), \dots \rangle$. The corresponding (fragment of the) *A-proc* is given by

$$\begin{aligned} P_A(Data_A) = {} & \mathit{inp}\ \mathit{net}_A\ (\exists\gamma)\ \gamma; \\ & \mathit{chk}(\gamma \mid (\exists\delta, \theta)\ \{\!|\delta; \theta|\!\}^{\mathrm{a}}_{K_A}); \\ & \mathit{chk}(\delta \mid (\exists N)\ \{\!|N|\!\}^{\mathrm{a}}_{K_B^{-1}}); \\ & \mathit{chk}(\theta \mid H(N)); \dots, \end{aligned}$$

where all checks are of the first kind and, for simplicity, we used γ, δ, and θ instead of $\gamma_{\{\!|\{\!|N|\!\}^{\mathrm{a}}_{K_B^{-1}}; H(N)|\!\}^{\mathrm{a}}_{K_A}}$, $\gamma_{\{\!|N|\!\}^{\mathrm{a}}_{K_B^{-1}}}$, and $\gamma_{H(N)}$, respectively.

Suppose now that, in a protocol, some principal generates a fresh number N and uses it to produce a key $K = N; H(N)$ and its inverse $K^{-1} = (N; H(N))^{-1}$. Then, later in the protocol, some other principal A receives, in three different messages, first $\{\!|M|\!\}^{\mathrm{a}}_K$, then K^{-1} in order to obtain M, and finally N itself so that he can check the key K. This corresponds to a role for A of the form $\langle \dots \mathbf{r}(\{\!|M|\!\}^{\mathrm{a}}_{N;H(N)}) \dots \mathbf{r}((N; H(N))^{-1}) \dots \mathbf{r}(N) \dots \rangle$. The corresponding (fragment of the) *A-proc* is given by

$$\begin{aligned} P_A(Data_A) = {} & \dots; \mathit{inp}\ \mathit{net}_A\ (\exists\gamma)\ \gamma; \\ & \dots; \mathit{inp}\ \mathit{net}_A\ (\exists\delta)\ \delta; \mathit{chk}(\delta; \gamma \mid (\exists M, \theta)\ \theta^{-1}; \{\!|M|\!\}^{\mathrm{a}}_{\theta}); \\ & \dots; \mathit{inp}\ \mathit{net}_A\ (\exists N)\ N; \mathit{chk}(\theta \mid\ N; (H(N))); \dots, \end{aligned}$$

where the check in the second line is of the second kind, and the check in the third line is of the third kind. Note that, for simplicity, we used γ, δ, and θ instead of $\gamma_{\{\!|M|\!\}^{\mathrm{a}}_{N;H(N)}}$, $\gamma_{(N;H(N))^{-1}}$, and $\gamma_{N;H(N)}$, respectively. Note too that, although θ is existentially quantified in the check in the second line, θ itself will not be collected into the principal's basic set because it does not occur in an analyzable position. Hence, the principal will not be able to compose messages using θ. Actually, θ^{-1} will be collected instead, which explains the last check.

Given an executable protocol specification, the spi-calculus processes that implement its roles are all implementable. To prove this, one needs to draw a parallel between the evolving datasets of the previous section and the symbol sets resulting from the corresponding spi process. By induction, it follows that

Proposition 27. *Let $D_A^0, \dots, D_A^s$ be the evolving datasets corresponding to A's run of the protocol. For every i such that $0 \leqslant i \leqslant s$, if $M \in D_A^i$ then*

(1) $cf_{\chi_i}(M) \in \mathit{synth}(\chi_i)$,
(2) $\sigma_k(cf_{\chi_i}(M)) = v_{D_A^k}(M)$, *for each* $i \leqslant k \leqslant s$, *and*
(3) *if* $\mathit{face}(M) \in \mathit{synth}(\chi_i)$ *then* $\sigma_k(\mathit{face}(M)) = v_{D_A^k}(M)$, *for each* $i \leqslant k \leqslant s$.

As a direct consequence of (1) in Proposition 27, and since the transparency of all the patterns checked in the process is guaranteed by Definitions 23, 25, and 26, we obtain the following result.

Proposition 28. *For every participant A of an executable Alice&Bob protocol specification, the process A-proc is implementable.*

Example 29. We give processes for the roles of the protocol examples that we have previously considered, where, for simplicity, $\gamma, \delta, \theta, \mu, \dots$ name the symbols involved in the processes. The processes for the two NSPK roles are:

$$\begin{aligned} \mathit{NSPK}_A(B, K_B) = {} & \mathit{new}\ N_1 : \mathit{Num}; \\ & \mathit{out}\ \mathit{net}_B\ \{\!|N_1; A|\!\}^{\mathrm{a}}_{K_B}; \\ & \mathit{inp}\ \mathit{net}_A\ (\exists\gamma)\ \gamma; \\ & \mathit{chk}(\gamma \mid (\exists N_2)\ \{\!|N_1; N_2|\!\}^{\mathrm{a}}_{K_A}); \\ & \mathit{out}\ \mathit{net}_B\ \{\!|N_2|\!\}^{\mathrm{a}}_{K_B}. \end{aligned}$$

$$\begin{aligned} NSPK_B(A, K_A) = {} & inp\ net_B\ (\exists\gamma)\ \gamma; \\ & chk(\gamma \mid (\exists N_1)\ \{\!|N_1; A|\!\}^{a}_{K_B}); \\ & new\ N_2 : Num; \\ & out\ net_A\ \{\!|N_1; N_2|\!\}^{a}_{K_A}; \\ & inp\ net_B\ (\exists\delta)\ \delta; \\ & chk(\delta \mid \{\!|N_2|\!\}^{a}_{K_B}). \end{aligned}$$

The processes for the three OR roles are:

$$\begin{aligned} OR_A(B, S, K_{AS}, I) = {} & new\ N_1 : Num; \\ & out\ net_B\ I; A; B; \{\!|N_1; I; A; B|\!\}^{s}_{K_{AS}}; \\ & inp\ net_A\ (\exists\gamma)\ I; \gamma; \\ & chk(\gamma \mid (\exists K)\ \{\!|N_1; K|\!\}^{s}_{K_{AS}}). \end{aligned}$$

$$\begin{aligned} OR_B(S, K_{BS}) = {} & inp\ net_B\ (\exists I, A, \gamma)\ I; A; B; \gamma; \\ & new\ N_2 : Num; \\ & out\ net_S\ I; A; B; \gamma; \{\!|N_2; I; A; B|\!\}^{s}_{K_{BS}}; \\ & inp\ net_B\ (\exists\delta, \theta)\ I; \delta; \theta; \\ & chk(\theta \mid (\exists K)\ \{\!|N_2; K|\!\}^{s}_{K_{BS}}); \\ & out\ net_A\ I; \delta. \end{aligned}$$

$$\begin{aligned} OR_S(A, B, K_{AS}, K_{BS}) = {} & inp\ net_S\ (\exists I, \gamma, \delta)\ I; A; B; \gamma; \delta; \\ & chk(\gamma \mid (\exists N_1)\ \{\!|N_1; I; A; B|\!\}^{s}_{K_{AS}}); \\ & chk(\delta \mid (\exists N_2)\ \{\!|N_2; I; A; B|\!\}^{s}_{K_{BS}}); \\ & new\ K : Num; \\ & out\ net_B\ I; \{\!|N_1; K|\!\}^{s}_{K_{AS}}; \{\!|N_2; K|\!\}^{s}_{K_{BS}}. \end{aligned}$$

Finally, the processes for the two roles of the ASW protocol include not only ghost symbols but also abortion checks (which we expand for the sake of illustration):

$$\begin{aligned} ASW_A(B, K_B, M) = {} & new\ N_1 : Num; \\ & out\ net_B\ \{\!|K_A; K_B; M; H(N_1)|\!\}^{a}_{K_A^{-1}}; \\ & inp\ net_A\ (\exists\gamma)\ \gamma; \\ & chk(\gamma \mid (\exists\delta, \theta)\ \{\!|\delta; \theta|\!\}^{a}_{K_B^{-1}}); \\ & chk(\delta \mid (\exists\mu)\ \{\!|K_A; K_B; M; \mu|\!\}^{a}_{K_A^{-1}}); \\ & chk(\mu \mid H(N_1)); \\ & out\ net_B\ N_1; \\ & inp\ net_A\ (\exists N_2)\ N_2; \\ & chk(\theta \mid H(N_2)). \end{aligned}$$

$$\begin{aligned} ASW_B(A, K_A) = {} & inp\ net_B\ (\exists\gamma)\ \gamma; \\ & chk(\gamma \mid (\exists M, \delta)\ \{\!|K_A; K_B; M; \delta|\!\}^{a}_{K_A^{-1}}); \\ & new\ N_2 : Num; \\ & out\ net_A\ \{\!|\gamma; H(N_2)|\!\}^{a}_{K_B^{-1}}; \\ & inp\ net_B\ (\exists N_1)\ N_1; \\ & chk(\delta \mid H(N_1)); \\ & out\ net_A\ N_2. \end{aligned}$$

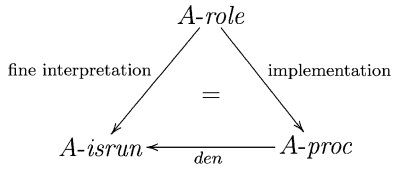


Fig. 13. Correctness of the implementation with respect to the fine interpretation.

Let us compare $A\text{-}proc = P_A(Data_A)$ with $A\text{-}isrun$. To begin with, $Data_A$ introduces precisely the minimal initial dataset D_A^0. Inspecting the spi-calculus process, we see that, by definition, $A\text{-}proc$ is a sequential composition of blocks of the kinds new-block, out-block, and a block kind that combines input and check as follows:

inp-chk-block: $inp\ net_A\ (\exists_\chi)\ M;\ chk(M'_1 \mid (\exists_{\chi^1})\ M_1); \ldots;\ chk(M'_t \mid (\exists_{\chi^t})\ M_t)$.

Recall that each of the messages M'_i in such an inp-chk-block is either a ghost symbol, or a pair of ghost symbols.

The blocks can be identified with **f**, **s**, and **r** actions, respectively.

Definition 30. The function *act* associates an action to each spi-calculus block of kind new-block, out-block, or inp-chk-block as follows:

$$act(Q) = \begin{cases} \mathbf{f}(N) & \text{if } Q = new\ N : Num, \\ \mathbf{s}(M, B) & \text{if } Q = out\ net_B\ M, \\ \mathbf{r}(M) & \text{if } Q = inp\ net_A\ (\exists_\chi)M;\ chk(M'_1 \mid (\exists_{\chi^1})M_1);\ \ldots;\ chk(M'_t \mid (\exists_{\chi^t})M_t). \end{cases}$$

Inp-chk-blocks, however, express much more than just the corresponding action. As we have seen before, while the action corresponding to each block is meant to extend the action execution sequence (with respect to the $\leqslant_\tau$ ordering), inp-chk-blocks also refine the execution sequence already obtained (with respect to the $\leqslant_\sigma$ ordering). When successful, the execution of each $chk(M' \mid (\exists_\chi)M)$ allows the principal executing the process to recognize that every previous occurrence of the symbols in M' match the corresponding submessages of M, i.e., the principal can refine the previous sequence by substituting every occurrence of each of the symbols with the corresponding submessage, thus building the sequence of substitutions previously defined.

Recall that a spi-calculus process $P = Q_1; \ldots; Q_n$ induces a sequence of substitutions $\sigma_0, \sigma_1, \ldots, \sigma_n$, as defined at the end of Section 4.1.

Definition 31. Let $P = Q_1; \ldots; Q_n$ be a spi-calculus process obtained by the sequential composition of blocks Q_i of kinds new-block, out-block, or inp-chk-block. The *symbolic run induced by P* is

$$den(P) = \langle \sigma_n(act(Q_1)), \ldots, \sigma_n(act(Q_n)) \rangle.$$

We can now formalize the *correctness* of the operational semantics we have presented with respect to the denotational semantics obtained by the fine interpretation. Recall that $A\text{-}run = \langle \mathbf{a}_1 \ldots \mathbf{a}_s \rangle$. By induction, exploiting items (2) and (3) of Proposition 27, we have that

Proposition 32. $den(spi_1^A(\mathbf{a}_1);\ \ldots;\ spi_i^A(\mathbf{a}_i)) = A\text{-}isrun^i$ *for every i such that* $1 \leqslant i \leqslant s$.

Hence, $A\text{-}isrun$ corresponds to

$$\begin{array}{l} den(spi_1^A(\mathbf{a}_1)) \\ den(spi_1^A(\mathbf{a}_1);\ spi_2^A(\mathbf{a}_2)) \\ \quad\vdots \\ den(spi_1^A(\mathbf{a}_1);\ spi_2^A(\mathbf{a}_2);\ \ldots;\ spi_s^A(\mathbf{a}_s)). \end{array}$$

Fig. 13 summarizes the resulting relations between $A\text{-}role$, $A\text{-}proc$, and $A\text{-}isrun$.

5. Related work, concluding remarks, and outlook

We have provided a detailed analysis of the meaning and limitations of a message sequence notation for security protocols, commonly known as Alice&Bob notation. We have given a formal characterization of the denotational semantic spectrum of Alice&Bob protocol specifications. Guided by our abstract denotational semantics based on incremental symbolic runs, we have also provided a fully executable operational semantics for Alice&Bob specifications in terms of the pattern-matching spi calculus, where all the details of the internal manipulation of messages by principals are made explicit. This semantic investigation is independent of the particular formalism chosen for protocol analysis and thus is relevant for researchers working on the design of security protocol specification languages and analysis tools, as well as for practitioners interested in precisely understanding what their specifications (should) mean.

Our results illustrate that, despite the fact that the Alice&Bob notation does not include explicit control-flow constructs, it is possible to make some of these aspects explicit when producing formal models and implementations for protocols that consist of a linear sequence of message-exchange steps. For non-linear protocols with explicit control flow, alternative notation must be used, based on richer specification languages.

As we remarked above, several researchers (e.g. [2,3,9,13,16,20,24,26,28,31,32]) have investigated ways of making explicit at least part of what is left implicit, or even unspecified, in Alice&Bob-style specifications. We will now discuss some of these alternatives in more detail and make comparisons.

The syntax of Casper [32] explicitly extends the standard Alice&Bob notation to include a "%"-notation for representing unreadable messages as variables. The user must insert this notation by hand during specification, i.e., he must perform an analysis of which messages are opaque and explicitly mark these as such. In contrast, by changing the interpretation of the terms used in Alice&Bob specifications, we automatically handle message opacity without resorting either to new notation or user involvement.

Our definition of the view that a principal has of a message with respect to his current knowledge is similar to the message patterns proposed by Abadi and Rogaway in [2]. Their patterns were introduced for a rather different purpose, namely reconciling cryptographic and formal methods approaches to security protocol analysis. The main technical difference is that we distinguish between distinct opaque elements by means of distinct variables, whereas Abadi and Rogaway use only one variable (their "box" symbol). This difference reflects the different objectives: Abadi and Rogaway aim to define a notion of equivalence between messages up to opaque submessages, while we are interested in describing precisely all the possible concrete instances of messages that match a pattern. In this sense, different variables can be given different values, which cannot be done if we use the same variable. Moreover, even if two messages are opaque, one can certainly still compare them and check if they are equal.

In [9], Bodei et al. present a protocol analysis approach in which standard Alice&Bob protocol specifications are extended with annotations that specify some of the checks that principals should perform on messages, as well as the authentication goals of the protocol under consideration. Their annotated protocol specifications are translated into terms in the process algebra LySa, where static analysis methods are used for protocol verification [9,13]. While their use of annotation also aims at disambiguating Alice&Bob protocol specifications, their annotations are quite different from our incremental symbolic runs. Their annotations allow one to specify details like the origin and destinations of messages, which data should be secret, as well as assumptions that are useful for proving authentication. In contrast, our incremental symbolic runs formalize how the data possessed by the principals grows during protocol execution. We believe that our equivalence results between the denotational and operational semantics could prove helpful for implementing the annotation process of [9], which is now carried out by hand.

CAPSL [24] and CASRUL [31] are two systems that, like Casper, automate the translation of protocol descriptions into low-level languages that can be handled by automated analysis tools. All these systems are based on high-level specification languages that allow one to write the messages exchanged in the protocols in an Alice&Bob-style notation and also to specify additional information, such as the types of the data manipulated, the initial knowledge of the principals, the capabilities of the intruder, and the protocol sessions to be considered during analysis. In particular, the CAPSL environment also features a compiler from the CAPSL high-level specification language to an intermediate formalism CIL, which may be converted to input for different automated protocol analysis tools such as Maude [6] or NRL [11]. Note, however, that CAPSL does not explicitly consider message opacity and, for instance, cannot handle protocols where a principal receives a message that he cannot decrypt immediately, which is a feature that is necessary for some protocols, such as non-repudiation protocols.

In [31], Jacquemard, Rusinowitch, and Vigneron present a compiler that automatically translates protocol descriptions written in a high-level protocol specification language based on Alice&Bob notation into low-level descriptions suitable for automatic analysis with CASRUL and the AVISS Tool [4]. [4] In [31], the authors present an operational semantics for Alice&Bob specifications by means of a translation into a low-level language based on multi-set rewriting. Our formalization of Alice&Bob specifications is in the same spirit as theirs, but our starting point and aim are different. Our work provides a formal characterization of the denotational semantic spectrum of Alice&Bob specifications, ranging from the direct interpretation to the abstract denotational semantics based on (incremental) symbolic runs. Our investigation is independent of the particular protocol analysis formalism chosen, and we have then formally related the denotational semantics to an operational one, thus providing a justification of the operational semantics in terms of the denotational one. For concreteness, as a "low-level" formalism, we have used the pattern-matching spi calculus, but our approach is general and other, similar formalisms could have also been chosen. In fact, thanks to their generality and expressiveness, the multi-set rewriting specifications of [31] can also be seen as providing an "implementation" of incremental symbolic runs, which constitutes an alternative to the operational semantics we have given here. Note, however, that our operational semantics provides a direct implementation of each of the protocol roles, while CASRUL does not consider the internal manipulation of messages by principals, thus yielding a symbolic model. Although this symbolic model does not provide an executable description of all the steps taken by principals executing their role automata, we should stress that it is a perfectly adequate model for protocol verification, since security properties depend only on the sequence of messages sent and received, but not on how these messages were synthesized or analyzed. This comment applies also to other approaches that consider only symbolic models.

In parallel with our work, Briais and Nestmann [12] have proposed an operational semantics for Alice&Bob protocol specifications that fixes a particular interpretation on how the protocol participants are supposed to execute, and which provides the basis for formally translating Alice&Bob specifications into the spi calculus. Although similar in spirit and aim, their work is quite different from ours. They provide an operational semantics for Alice&Bob specifications, without a denotational one. In contrast, we provide a denotational semantics which guided the formalization of our operational semantics. Another difference is that, for the sake of brevity, Briais and Nestmann consider a more restricted protocol language, which drastically simplifies the treatment of inverse keys. The definition of the knowledge of the honest agents and the intruder is also simplified. This simplification, together with a number of restrictions that they consider, restricts the kinds of checks that the agents perform in their approach in comparison with ours.

Explicit pattern matching as a primitive makes the spi calculus of Haack and Jeffrey [28] well-suited for our purposes (e.g., to check that the pattern of a ghost symbol matches a given message, and thus implement the symbolic checks for protocol abortion), and we have used a slight syntactic variant of a fragment of the calculus of [28]. Our notions of feasibility and executability are closely related to the notions of message and pattern implementability, as the underlying ideas are the same. However, as remarked above, implementability is a context-independent notion that depends only on the message or pattern at hand; in contrast, feasibility and executability depend on the context where the message (or pattern) appears.

We close with a few remarks on future work. To begin with, it is not difficult to see how our work could be integrated with the strand spaces approach, in particular with the extensions considered in [20,26] to model explicit checks. Denotational protocol models, for instance, could then be built such that the allowed behaviors for the participants correspond to all concrete instantiations of each of the incremental symbolic runs of the fine interpretation. For example, in strand spaces, each of these sequences should correspond to a strand. Capturing the resulting non-determinism would not require an enrichment of the strand spaces approach itself; rather, one could just enlarge the number of strands in the space that models the protocol under consideration. However, in the general case, i.e., for protocols with explicit control flow, the extension of strand spaces with a notion of *conflict* as suggested in [21,29] seems to be the only option.

As we have previously remarked, we have only focused in this paper on sending and receiving messages and generating fresh numbers. However, other internal actions, such as those that construct messages by applying cryptographic operations, can be modeled similarly and our results extended straightforwardly. Such an extension would allow us to fully formalize the process of compiling messages to sequences of actions and thus extend the ideas described above to

[4] This compiler has then been extended to similarly translate protocol descriptions written in the high-level protocol specification language HLPSL of the AVISPA Tool [3,7,16,41], which allows for the specification of control-flow constructs such as if-then-else branches, looping and other features that cannot be captured using Alice&Bob notation.

directly build analysis tools based on them. As shown by our operational semantics, incremental symbolic runs provide a good basis for generating protocol implementations from Alice&Bob-style specifications that explicitly carry out all necessary checks when parsing received messages.

Finally, note that we have not considered protocol goals here, i.e., the security properties that the protocols have been designed to achieve. This is deliberate and in contrast to most protocol analysis approaches, where the descriptions of protocols and properties are often given together (such as the different "annotations" considered in [3,9,16,24,31,32]). Our approach is based instead on a methodological decision to separate protocols and properties: this separation allows us to view a security protocol as a distributed program, for which we can give both a denotational and an operational semantics, independent of the particular properties that the protocol is supposed to satisfy. Analyzing whether a protocol satisfies a property can then be carried out based on the semantics. We have begun extending our approach to carry out such analysis and will report on that soon.

Appendix A

A.1. On datasets and views

In the first part of this appendix, we prove a number of technical lemmas related to datasets and views that are useful for proving the main results of the paper in Section A.2.

Recall that, given a dataset D, we say that a set of messages S is D-opaque, or D-transparent, provided that all the messages in S are. In case D is itself D-transparent, we will simply refer to D as transparent. By the definitions, D-transparency is related to the absence of D-opaque submessages.

Lemma 33. *A message M is D-transparent iff $sub(M)$ does not contain D-opaque elements.*

Proof. We proceed by induction on the structure of the message M.

- If M is atomic, then both sides of the "iff" are trivially true since $sub(M) = \{M\}$ and $v_D(M) = M$, which means that M is D-transparent and thus not D-opaque.
- If $M = M_1; M_2$, then M is D-transparent iff $v_D(M_1; M_2) = v_D(M_1); v_D(M_2) = M_1; M_2$ iff M_1 and M_2 are D-transparent iff $sub(M_1)$ and $sub(M_2)$ do not contain D-opaque elements (using the induction hypothesis) iff $sub(M_1; M_2)$ does not contain D-opaque elements (since $M_1; M_2$ is never D-opaque).
- If $M = \{\!|M_1|\!\}_K$, then M is D-transparent iff $v_D(\{\!|M_1|\!\}_K) = \{\!|v_D(M_1)|\!\}_{v_D(K)} = \{\!|M_1|\!\}_K$ iff M_1 and K are D-transparent and $\{\!|M_1|\!\}_K$ is not D-opaque iff $sub(M_1)$ and $sub(K)$ do not contain D-opaque elements (using the induction hypothesis) and $\{\!|M_1|\!\}_K$ is not D-opaque iff $sub(\{\!|M_1|\!\}_K)$ does not contain D-opaque elements.
- If $M = K^{-1}$, then M is D-transparent iff $v_D(K^{-1}) = v_D(K)^{-1} = K^{-1}$ iff K is D-transparent and K^{-1} is not D-opaque iff $sub(K)$ does not contain D-opaque elements (using the induction hypothesis) and K^{-1} is not D-opaque iff $sub(K^{-1})$ does not contain D-opaque elements.
- If $M = H(M_1)$, then M is D-transparent iff $v_D(H(M_1)) = H(v_D(M_1)) = H(M_1)$ iff M_1 is D-transparent and $H(M_1)$ is not D-opaque iff $sub(M_1)$ does not contain D-opaque elements (using the induction hypothesis) and $H(M_1)$ is not D-opaque iff $sub(H(M_1))$ does not contain D-opaque elements. □

To better understand how a principal's view of a message may evolve, we also prove:

Lemma 34. *Let $D \subseteq D'$, with D' a dataset. For a given message M', the following are equivalent*:

(1) $v_D(M') = v_{D'}(M')$.
(2) *If γ_M occurs in $v_D(M')$ then it also occurs in $v_{D'}(M')$.*
(3) *If γ_M occurs in $v_D(M')$ then M is D'-opaque, or else $M \notin D'$ occurs in M' only in non-analyzable positions.*

Proof. (1) implies (2) because if the views are the same, then they contain the same ghost symbols. It is also straightforward to show that (2) implies (3) because if γ_M occurs in $v_{D'}(M')$, then either it occurs in an analyzable position and M is D'-opaque, or it occurs only in non-analyzable positions and $M \notin D'$. Hence, by transitivity, (1) implies (3).

To show that (3) implies (1), we proceed by induction on the structure of M'.

- If M' is atomic, then it is immediate that $v_D(M') = M' = v_{D'}(M')$.
- If $M' = M_1; M_2$, then γ_M occurs in $v_D(M_1; M_2) = v_D(M_1); v_D(M_2)$ iff it occurs in $v_D(M_1)$ or $v_D(M_2)$ and, by assumption, M must be D'-opaque. Thus, by the induction hypothesis, $v_D(M_1) = v_{D'}(M_1)$ and $v_D(M_2) = v_{D'}(M_2)$, which implies that $v_{D'}(M_1; M_2) = v_{D'}(M_1); v_{D'}(M_2) = v_D(M_1; M_2)$.
- If $M' = \{\!|M_1|\!\}_K$, then we must consider two possibilities for $v_D(\{\!|M_1|\!\}_K)$.
  - If $\{\!|M_1|\!\}_K$ is not D-opaque then $v_D(\{\!|M_1|\!\}_K)$ is either $\{\!|v_D(M_1)|\!\}_{v_D(K)}$ or $\{\!|v_D(M_1)|\!\}_{v_D(K)}$. Hence, if γ_M occurs in $v_D(\{\!|M_1|\!\}_K)$ then M is D'-opaque or else it does not occur in analyzable positions of $v_D(M_1)$ and $M \notin D'$. Thus, by the induction hypothesis, $v_D(M_1) = v_{D'}(M_1)$. If $v_D(\{\!|M_1|\!\}_K) = \{\!|v_D(M_1)|\!\}_{v_D(K)}$ then $M_1, K \in D$. Now, when γ_M occurs in $v_D(K)$ and M is not D'-opaque then $M \notin D'$ and it can only occur in non-analyzable positions because $K \in D$. Thus, by the induction hypothesis, we have $v_D(K) = v_{D'}(K)$, and therefore $v_{D'}(\{\!|M_1|\!\}_K) = \{\!|v_{D'}(M_1)|\!\}_{v_{D'}(K)} = v_D(\{\!|M_1|\!\}_K)$ because $D \subseteq D'$.
  - Otherwise, $\{\!|M_1|\!\}_K$ is D-opaque. Then $v_D(\{\!|M_1|\!\}_K) = \gamma_{M'}$ occurs in an analyzable position and, by assumption, $\{\!|M_1|\!\}_K$ must be D'-opaque. Therefore, $v_{D'}(\{\!|M_1|\!\}_K) = \gamma_{M'} = v_D(\{\!|M_1|\!\}_K)$.
- If $M' = K^{-1}$, then we must consider two possibilities for $v_D(K^{-1})$.
  - If K^{-1} is not D-opaque, then $v_D(K^{-1})$ is either $v_D(K)^{-1}$ or γ_K^{-1}. In any case, if γ_M occurs in $v_D(K^{-1})$ then it must be in a non-analyzable position and, by assumption, M is D'-opaque or $M \notin D'$. If $v_D(K^{-1}) = v_D(K)^{-1}$ then, when γ_M occurs in $v_D(K)$ and M is not D'-opaque then $M \notin D'$ and it can only occur in non-analyzable positions because $K \in D$. Thus, by the induction hypothesis, $v_D(K) = v_{D'}(K)$ which, together with $D \subseteq D'$, implies that $v_{D'}(K^{-1}) = v_{D'}(K)^{-1} = v_D(K^{-1})$. If $v_D(K^{-1}) = \gamma_K^{-1}$ then K is D'-opaque or $K \notin D'$. If $K \notin D'$ then $v_{D'}(K^{-1}) = \gamma_K^{-1} = v_D(K^{-1})$. Otherwise, $v_{D'}(K^{-1}) = v_{D'}(K)^{-1} = \gamma_K^{-1} = v_D(K^{-1})$ because K must be D'-opaque.
  - Otherwise, $v_D(K^{-1}) = \gamma_{M'}$ occurs in an analyzable position and, by assumption, M' must be D'-opaque. Therefore, $v_{D'}(K^{-1}) = \gamma_{M'} = v_D(K^{-1})$.
- If $M' = H(M_1)$, then we must also consider two possibilities.
  - If $H(M_1)$ is not D-opaque, then γ_M occurs in $v_D(H(M_1)) = H(v_D(M_1))$ iff it occurs in $v_D(M_1)$ and, by assumption, M must be D'-opaque. Thus, by the induction hypothesis, $v_D(M_1) = v_{D'}(M_1)$ which, together with $D \subseteq D'$, implies that $v_{D'}(H(M_1)) = H(v_{D'}(M_1)) = v_D(H(M_1))$.
  - Otherwise, $v_D(H(M_1)) = \gamma_{M'}$ and, by assumption, M' must be D'-opaque. Therefore, $v_{D'}(H(M_1)) = \gamma_{M'} = v_D(H(M_1))$. □

Note that if $D \subseteq D'$ then it immediately follows that D'-opaque messages are also D-opaque. Hence, as a corollary of Lemma 34, we have that if D'' is also a dataset and $D \subseteq D' \subseteq D''$, then $v_D(M') = v_{D''}(M')$ implies that $v_D(M') = v_{D'}(M') = v_{D''}(M')$.

The following lemma tells us that if an opaque message does not appear as a submessage of any message in a set S, then it cannot appear as a submessage of any message in $close(S)$.

Lemma 35. *Let M be a D-opaque message and let $S \subseteq D$. Then $M \in sub(close(S))$ iff $M \in sub(S)$.*

Proof. Note that $S \subseteq close(S)$ and thus $sub(S) \subseteq sub(close(S))$. Therefore, the right-to-left implication is immediate. We prove the left-to-right implication by induction on the construction rules of the messages $M' \in close(S)$, assuming that $M \in sub(M')$.

- If $M' \in S$, then $M \in sub(M') \subseteq sub(S)$.
- If $M' = M_1; M_2$ with $M_1, M_2 \in close(S)$, then $M \in sub(M_1)$ or $M \in sub(M_2)$ because $M_1; M_2$ cannot be D-opaque. In either case, by the induction hypothesis, $M \in sub(S)$.
- If $M' = H(M_1)$ with $M_1 \in close(S)$, then $M \in sub(M_1)$ because $M_1 \in D$, and hence $H(M_1)$ is not D-opaque. So, by the induction hypothesis, $M \in sub(S)$.
- If $M' = \{\!|M_1|\!\}_K$ with $M_1, K \in close(S)$, then $M \in sub(M_1)$ or $M \in sub(K)$ because $M_1, K \in D$ and hence $\{\!|M_1|\!\}_K$ is not D-opaque. In either case, by the induction hypothesis, $M \in sub(S)$.
- If $M'; M'' \in close(S)$ (or, analogously, if $M''; M' \in close(S)$), then $M \in sub(M'; M'')$ and by the induction hypothesis, $M \in sub(S)$.

- If $\{\!|M'|\!\}^{\mathrm{a}}_{K}$, $K^{-1} \in \mathit{close}(S)$, then $M \in \mathit{sub}(\{\!|M'|\!\}^{\mathrm{a}}_{K})$ and by the induction hypothesis, $M \in \mathit{sub}(S)$.
- If $\{\!|M'|\!\}^{\mathrm{a}}_{K^{-1}}$, $K \in \mathit{close}(S)$, then $M \in \mathit{sub}(\{\!|M'|\!\}^{\mathrm{a}}_{K^{-1}})$ and by the induction hypothesis, $M \in \mathit{sub}(S)$.
- Finally, if $\{\!|M'|\!\}^{\mathrm{s}}_{K}$, $K \in \mathit{close}(S)$, then $M \in \mathit{sub}(\{\!|M'|\!\}^{\mathrm{s}}_{K})$ and by the induction hypothesis, $M \in \mathit{sub}(S)$. □

Lemma 35 has the following two immediate corollaries.

Corollary 36. *If S is a set of atomic messages, then $\mathit{close}(S)$ is transparent.*

Note that, by definition, atomic messages are always transparent. Hence, if S is a set of atomic messages, then $\mathit{sub}(S) = S$ does not contain opaque elements. Therefore, Lemma 35 implies that also $\mathit{sub}(\mathit{close}(S))$ does not contain opaque elements. Finally, Lemma 33 guarantees that $\mathit{close}(S)$ is transparent.

Corollary 37. *Let M' be a message and $D' = \mathit{close}(D \cup \{M'\})$. If D is transparent, then D' is transparent iff M' is D'-transparent.*

By induction on the structure of messages, we can obtain a result similar to Lemma 35, which explains how ghost symbols appear during protocol execution:

Lemma 38. *Let S be a set of messages and $D' = \mathit{close}(D \cup S)$. If $M' \in D'$, $M \notin \mathit{sub}(D)$, and γ_M occurs in $v_{D'}(M')$, then γ_M also occurs in $v_{D'}(S)$.*

Proof. We proceed by induction on the construction rules of $M' \in D'$.
- If $M' \in D$, then γ_M does not occur in $v_{D'}(M')$ or else we would have $M \in \mathit{sub}(M') \subseteq \mathit{sub}(D)$.
- If $M' \in S$, then the result follows trivially because $v_{D'}(M') \in v_{D'}(S)$.
- If $M' = M_1; M_2$ with $M_1, M_2 \in D'$, then when γ_M occurs in $v_{D'}(M_1; M_2) = v_{D'}(M_1); v_{D'}(M_2)$ it must occur in $v_{D'}(M_1)$ or in $v_{D'}(M_2)$. In either case, by the induction hypothesis, γ_M also occurs in $v_{D'}(S)$.
- If $M' = H(M_1)$ with $M_1 \in D'$, then when γ_M occurs in $v_{D'}(H(M_1)) = H(v_{D'}(M_1))$ it must occur in $v_{D'}(M_1)$. So, by the induction hypothesis, γ_M also occurs in $v_{D'}(S)$.
- If $M' = \{\!|M_1|\!\}_K$ with $M_1, K \in D'$, then when γ_M occurs in $v_{D'}(\{\!|M_1|\!\}_K) = \{\!|v_{D'}(M_1)|\!\}_{v_{D'}(K)}$ it must occur in $v_{D'}(M_1)$ or in $v_{D'}(K)$. In either case, by the induction hypothesis, γ_M also occurs in $v_{D'}(S)$.
- If $M'; M'' \in D'$, then when γ_M occurs in $v_{D'}(M')$ it must also occur in $v_{D'}(M'; M'') = v_{D'}(M'); v_{D'}(M'')$. So, by the induction hypothesis, γ_M also occurs in $v_{D'}(S)$.
- Analogously, if $M''; M' \in D'$, then when γ_M occurs in $v_{D'}(M')$ it must also occur in $v_{D'}(M''; M') = v_{D'}(M''); v_{D'}(M')$. So, by the induction hypothesis, γ_M also occurs in $v_{D'}(S)$.
- If $\{\!|M'|\!\}^{\mathrm{a}}_{K}$, $K^{-1} \in D'$, then when γ_M occurs in $v_{D'}(M')$ it must also occur in $v_{D'}(\{\!|M'|\!\}^{\mathrm{a}}_{K})$, no matter whether it is $\{\!|v_{D'}(M')|\!\}^{\mathrm{a}}_{v_{D'}(K)}$ or $\{\!|v_{D'}(M')|\!\}^{\mathrm{a}}_{\gamma_K}$. So, by the induction hypothesis, γ_M also occurs in $v_{D'}(S)$.
- Analogously, if $\{\!|M'|\!\}^{\mathrm{a}}_{K^{-1}}$, $K \in D'$, then when γ_M occurs in $v_{D'}(M')$ it must also occur in $v_{D'}(\{\!|M'|\!\}^{\mathrm{a}}_{K^{-1}}) = \{\!|v_{D'}(M')|\!\}^{\mathrm{a}}_{v_{D'}(K^{-1})}$. So, by the induction hypothesis, γ_M also occurs in $v_{D'}(S)$.
- Finally, if $\{\!|M'|\!\}^{\mathrm{s}}_{K}$, $K \in D'$, then when γ_M occurs in $v_{D'}(M')$ it must also occur in $v_{D'}(\{\!|M'|\!\}^{\mathrm{s}}_{K}) = \{\!|v_{D'}(M')|\!\}^{\mathrm{s}}_{v_{D'}(K)}$. So, by the induction hypothesis, γ_M also occurs in $v_{D'}(S)$. □

The way that views are updated as the datasets evolve can also be explained in terms of substitutions of ghost symbols.

Lemma 39. *Let S be a set of messages and $D' = \mathit{close}(D \cup S)$. For every message $M \in D$ (without ghost symbols), we have that $v_{D'}(M) = \sigma(v_D(M))$, where σ is the substitution that replaces each ghost symbol $\gamma_{M'}$ such that $M' \in D'$ occurring in $v_D(M)$ by $v_{D'}(M')$.*

Proof. We proceed by induction on the structure of the message M.
- If M is atomic, then $v_{D'}(M) = v_D(M) = M$, and $\sigma(M) = M$ because M cannot be a ghost symbol.
- If $M = M_1; M_2$, then $v_{D'}(M_1; M_2) = v_{D'}(M_1); v_{D'}(M_2)$, which equals, using the induction hypotheses, $\sigma(v_D(M_1)); \sigma(v_D(M_2)) = \sigma(v_D(M_1); v_D(M_2)) = \sigma(v_D(M_1; M_2))$.

- If $M = \{\!|M_1|\!\}_K$, then we must consider the following two situations.
  - When $\{\!|M_1|\!\}_K$ is not D-opaque, then $v_D(\{\!|M_1|\!\}_K)$ is either $\{\!|v_D(M_1)|\!\}_{v_D(K)}$ or $\{\!|v_D(M_1)|\!\}_{\gamma_K}$. In the former case, the result follows by just applying the induction hypothesis. In the latter case, using also the induction hypothesis for M_1, if $K \in D'$ then $\{\!|\sigma(v_D(M_1))|\!\}_{\sigma(\gamma_K)} = \{\!|v_{D'}(M_1)|\!\}_{v_{D'}(K)} = v_{D'}(\{\!|M_1|\!\}_K)$; if, on the contrary, $K \notin D'$ then we have $\{\!|\sigma(v_D(M_1))|\!\}_{\sigma(\gamma_K)} = \{\!|v_{D'}(M_1)|\!\}_{\gamma_K} = v_{D'}(\{\!|M_1|\!\}_K)$.
  - When $\{\!|M_1|\!\}_K$ is D-opaque, we have $v_D(\{\!|M_1|\!\}_K) = \gamma_M$, $M \in D$, and so $\sigma(\gamma_M) = v_{D'}(\{\!|M_1|\!\}_K)$.
- If $M = K^{-1}$, then we must also consider two situations.
  - When K^{-1} is not D-opaque, then $v_D(K^{-1})$ is either $v_D(K)^{-1}$ or γ_K^{-1}. In the former case, the result follows immediately by using the induction hypothesis. In the latter case, if $K \in D'$ then $\sigma(\gamma_K)^{-1} = v_{D'}(K)^{-1} = v_{D'}(K^{-1})$; if, on the contrary, $K \notin D'$ then $\sigma(\gamma_K)^{-1} = \gamma_K^{-1} = v_{D'}(K^{-1})$.
  - When K^{-1} is D-opaque, we have $v_D(K^{-1}) = \gamma_M$, $M \in D$, and $\sigma(\gamma_M) = v_{D'}(K^{-1})$.
- Finally, if $M = H(M_1)$, then we must similarly consider two situations.
  - When $H(M_1)$ is not D-opaque, and hence also not D'-opaque, we have $v_{D'}(H(M_1)) = H(v_{D'}(M_1))$, which equals, using the induction hypothesis, $H(\sigma(v_D(M_1))) = \sigma(H(v_D(M_1))) = \sigma(v_D(H(M_1)))$.
  - When $H(M_1)$ is D-opaque, we have $v_D(H(M_1)) = \gamma_M$, and $\sigma(\gamma_M) = v_{D'}(H(M_1))$. □

To continue this sequence of technical results, we consider what happens to the notion of opacity when D is augmented by a fresh number instead of an arbitrary message. Recall that freshness means that the value occurs neither in any previous message nor in the inverse of a submessage of any previous message.

Lemma 40. *Let N be a number symbol such that $N \notin sub(D)$, and $D' = close(D \cup \{N\})$. If $M \in sub(D)$ and M is D-opaque, then M is also D'-opaque.*

Proof. We prove this in several steps.

(1) We begin by observing that $D \cup \{N\}$ is closed for analysis. If $M_1; M_2 \in D \cup \{N\}$, then $M_1; M_2 \in D$ and therefore $M_1, M_2 \in D$ because D is closed. If $\{\!|M_1|\!\}^{\mathrm{a}}_K, K^{-1} \in D \cup \{N\}$, then $\{\!|M_1|\!\}^{\mathrm{a}}_K \in D$. Thus $K \in sub(D)$ and $N \notin sub(K)$. Since $K^{-1} \neq N$, it must be the case that $K^{-1} \in D$ and $M_1 \in D$ because D is closed. If $\{\!|M_1|\!\}^{\mathrm{a}}_{K^{-1}}, K \in D \cup \{N\}$, then $\{\!|M_1|\!\}^{\mathrm{a}}_{K^{-1}} \in D$. Thus $K \in sub(D)$ and $N \notin sub(K)$. Hence, $K \neq N$ and it must be the case that $K \in D$ and $M_1 \in D$ because D is closed. Similarly, if $\{\!|M_1|\!\}^{\mathrm{s}}_K, K \in D \cup \{N\}$, then $\{\!|M_1|\!\}^{\mathrm{s}}_K \in D$. Thus $K \in sub(D)$ and $N \notin sub(K)$. Hence, $K \neq N$ and it must be the case that $K \in D$ and $M_1 \in D$ because D is closed.

(2) Next we can show, by induction on the synthesis rules, that if $M' \in synth(D \cup \{N\})$ and $N \notin sub(M')$ then $M' \in D$.

- If $M' \in D \cup \{N\}$ and $M' \neq N$, then $M' \in D$.
- If $M' = M_1; M_2$ and $M_1, M_2 \in synth(D \cup \{N\})$, then $N \notin sub(M_1)$ and $N \notin sub(M_2)$. So, by the induction hypothesis $M_1, M_2 \in D$ and $M_1; M_2 \in D$ because D is closed.
- If $M' = H(M_1)$ and $M_1 \in synth(D \cup \{N\})$, then $N \notin sub(M_1)$. So, by the induction hypothesis $M_1 \in D$ and $H(M_1) \in D$ because D is closed.
- If $M' = \{\!|M_1|\!\}_K$ and $M_1, K \in synth(D \cup \{N\})$, then $N \notin sub(M_1)$ and $N \notin sub(K)$. So, by the induction hypothesis $M_1, K \in D$ and $\{\!|M_1|\!\}_K \in D$ because D is closed.

(3) Now we can prove that $D' = synth(D \cup \{N\})$. Indeed, since $D \cup \{N\}$ is closed for analysis and it is always the case that $synth(analyz(S)) \subseteq close(S)$, it suffices to show that $synth(D \cup \{N\})$ is still closed for analysis.

- If $M_1; M_2 \in synth(D \cup \{N\})$, then either $M_1; M_2 \in D \cup \{N\}$ or $M_1; M_2$ was synthesized from $M_1, M_2 \in synth(D \cup \{N\})$. The latter case is trivial. In the former case, since $D \cup \{N\}$ is closed for analysis $M_1, M_2 \in D \cup \{N\} \subseteq synth(D \cup \{N\})$.
- If $\{\!|M_1|\!\}^{\mathrm{a}}_K, K^{-1} \in synth(D \cup \{N\})$, then either $\{\!|M_1|\!\}^{\mathrm{a}}_K \in D \cup \{N\}$ or $\{\!|M_1|\!\}^{\mathrm{a}}_K$ was synthesized from $M_1, K \in synth(D \cup \{N\})$. The latter case is trivial. In the former case, since $N \notin sub(K) \subseteq sub(D)$ and $N \neq K^{-1}$, the result in (2) above guarantees that $K^{-1} \in D$. But $\{\!|M_1|\!\}^{\mathrm{a}}_K \in D \cup \{N\}$ implies that $\{\!|M_1|\!\}^{\mathrm{a}}_K \in D$ and so $M_1 \in D \subseteq synth(D \cup \{N\})$ because D is closed.
- If $\{\!|M_1|\!\}^{\mathrm{a}}_{K^{-1}}, K \in synth(D \cup \{N\})$, then either $\{\!|M_1|\!\}^{\mathrm{a}}_{K^{-1}} \in D \cup \{N\}$ or $\{\!|M_1|\!\}^{\mathrm{a}}_{K^{-1}}$ was synthesized from $M_1, K^{-1} \in synth(D \cup \{N\})$. The latter case is trivial. In the former case, since $N \notin sub(K) \subseteq sub(K^{-1}) \subseteq sub(D)$, the result in (2) guarantees that $K \in D$. But $\{\!|M_1|\!\}^{\mathrm{a}}_{K^{-1}} \in D \cup \{N\}$ implies that $\{\!|M_1|\!\}^{\mathrm{a}}_{K^{-1}} \in D$ and so $M_1 \in D \subseteq synth(D \cup \{N\})$ because D is closed.

- If $\{\!|M_1|\!\}^{\mathrm{s}}_K$, $K \in \mathit{synth}(D \cup \{N\})$, then either $\{\!|M_1|\!\}^{\mathrm{s}}_K \in D \cup \{N\}$ or $\{\!|M_1|\!\}^{\mathrm{s}}_K$ was synthesized from $M_1, K \in \mathit{synth}(D \cup \{N\})$. The latter case is trivial. In the former case, since $N \notin \mathit{sub}(K) \subseteq \mathit{sub}(D)$, the result in (2) guarantees that $K \in D$. But $\{\!|M_1|\!\}^{\mathrm{s}}_K \in D \cup \{N\}$ implies that $\{\!|M_1|\!\}^{\mathrm{s}}_K \in D$ and so $M_1 \in D \subseteq \mathit{synth}(D \cup \{N\})$ because D is closed.

Let us now consider the D-opaque message $M \in \mathit{sub}(D)$.

- If $M = \{\!|M_1|\!\}^{\mathrm{a}}_K$, then $K^{-1} \notin D$ and $\{M_1, K\} \not\subseteq D$. Moreover, $N \notin \mathit{sub}(M_1) \subseteq \mathit{sub}(D)$, $N \notin \mathit{sub}(K) \subseteq \mathit{sub}(D)$, and so $N \notin \mathit{sub}(K^{-1})$. Hence, by (2) and (3), $K^{-1} \notin D'$, $\{M_1, K\} \not\subseteq D'$ and so $M = \{\!|M_1|\!\}^{\mathrm{a}}_K$ is D'-opaque.
- If $M = \{\!|M_1|\!\}^{\mathrm{a}}_{K^{-1}}$, then $K \notin D$ and $\{M_1, K^{-1}\} \not\subseteq D$. Moreover, $N \notin \mathit{sub}(M_1) \subseteq \mathit{sub}(D)$ and $N \notin \mathit{sub}(K) \subseteq \mathit{sub}(K^{-1}) \subseteq \mathit{sub}(D)$. Hence, by (2) and (3), $K \notin D'$, $\{M_1, K^{-1}\} \not\subseteq D'$ and so $M = \{\!|M_1|\!\}^{\mathrm{a}}_{K^{-1}}$ is D'-opaque.
- If $M = \{\!|M_1|\!\}^{\mathrm{s}}_K$ then $K \notin D$. Moreover, $N \notin \mathit{sub}(K) \subseteq \mathit{sub}(D)$. Hence, by (2) and (3), $K \notin D'$ and so $M = \{\!|M_1|\!\}^{\mathrm{s}}_K$ is D'-opaque.
- If $M = K^{-1}$, then $\{\!|M'|\!\}^{\mathrm{a}}_K \notin D$ for any M'. In particular, $K \notin D$. Moreover, $\{\!|M'|\!\}^{\mathrm{a}}_K \neq N$, hence, by (2) and (3), $\{\!|M'|\!\}^{\mathrm{a}}_K \notin D'$, for any M', and K^{-1} is D'-opaque.
- If $M = H(M_1)$, then $M_1 \notin D$. Moreover, $N \notin \mathit{sub}(M_1) \subseteq \mathit{sub}(D)$, hence, by (2) and (3), $M_1 \notin D'$ and $M = H(M_1)$ is D'-opaque. □

A.2. Proofs of the main results

We now prove the main results given in the body of the paper. Recall that we assume fixed an executable protocol and a participant A, with $A\text{-}run = \langle \mathbf{a}_1 \dots \mathbf{a}_s \rangle$.

Proposition 20. *The sequence A-run is representative iff every received message is transparent when it is received, i.e., if* $\mathbf{a}_i = \mathbf{r}(M)$, *then M is* D^i_A*-transparent.*

Proof. Note that *A-run* is representative if, for every $i \leqslant s$, we have that $A\text{-}isrun^i = A\text{-}run|_i$, that is, $\langle v_{D^i_A}(\mathbf{a}_1), \dots, v_{D^i_A}(\mathbf{a}_i)\rangle = \langle \mathbf{a}_1, \dots, \mathbf{a}_i\rangle$. This amounts to having $v_{D^i_A}(\mathbf{a}_j) = \mathbf{a}_j$, for each $1 \leqslant j \leqslant i \leqslant s$.

The left-to-right implication is immediate since, for each i, $v_{D^i_A}(\mathbf{a}_i) = \mathbf{a}_i$ implies that $v_{D^i_A}(\mathbf{r}(M)) = \mathbf{r}(M)$ if $\mathbf{a}_i = \mathbf{r}(M)$, and therefore $v_{D^i_A}(M) = M$.

For the right-to-left implication, we first prove, by induction, that for every $i \leqslant s$, the data set D^i_A is transparent and $v_{D^i_A}(\mathbf{a}_i) = \mathbf{a}_i$ if $i > 0$. Clearly, D^0_A is transparent according to Corollary 36. For $i > 0$, assume that D^{i-1}_A is transparent. If $\mathbf{a}_i = \mathbf{s}(M, B)$, then it follows from the executability of the protocol that $M \in D^{i-1}_A$ and therefore $v_{D^i_A}(M) = M$. Of course, $D^i_A = D^{i-1}_A$ is transparent. If $\mathbf{a}_i = \mathbf{r}(M)$ then, by assumption, $v_{D^i_A}(M) = M$. Hence, by Corollary 37, we also have that D^i_A is transparent. Finally, if $\mathbf{a}_i = \mathbf{f}(N)$ then obviously $v_{D^i_A}(N) = N$ and Corollary 37 again implies that D^i_A is transparent. The result follows by observing that $D^j_A \subseteq D^i_A$ for $0 \leqslant j < i$, and thus $v_{D^j_A}(M) = M$ implies $v_{D^i_A}(M) = M$. □

Proposition 22. *The sequence A-srun is representative iff every received message preserves the ghost symbols that occur in the views of previously received messages, i.e., if* $1 \leqslant j < i \leqslant s$, $\mathbf{a}_j$ *and* $\mathbf{a}_i$ *are receiving actions, and* γ_M *occurs in* $v_{D^{i-1}_A}(\mathbf{a}_j)$, *then* γ_M *also occurs in* $v_{D^i_A}(\mathbf{a}_j)$.

Proof. Note that *A-srun* is representative if, for every $i \leqslant s$, we have that $A\text{-}isrun^i = A\text{-}srun|_i$, that is, $\langle v_{D^i_A}(\mathbf{a}_1), \dots, v_{D^i_A}(\mathbf{a}_i)\rangle = \langle v_{D^1_A}(\mathbf{a}_1), \dots, v_{D^i_A}(\mathbf{a}_i)\rangle$. This amounts to having $v_{D^i_A}(\mathbf{a}_j) = v_{D^j_A}(\mathbf{a}_j)$, for each $1 \leqslant j \leqslant i \leqslant s$.

The left-to-right implication is straightforward. Assume that *A-srun* is representative, $1 \leqslant j < i \leqslant s$, $\mathbf{a}_i$ and $\mathbf{a}_j$ are receiving actions, and γ_M occurs in $v_{D^{i-1}_A}(\mathbf{a}_j)$. Then γ_M also occurs in $v_{D^i_A}(\mathbf{a}_j) = v_{D^j_A}(\mathbf{a}_j) = v_{D^{i-1}_A}(\mathbf{a}_j)$.

For the right-to-left implication, we will prove that $v_{D^{i-1}_A}(\mathbf{a}_j) = v_{D^i_A}(\mathbf{a}_j)$ for every $1 \leqslant j < i \leqslant s$. We have to consider the three possibilities for $\mathbf{a}_i$.

- If $\mathbf{a}_i$ is a sending action, then $D^i_A = D^{i-1}_A$ and thus the views are equal.

- If $\mathbf{a}_i$ is a fresh generation action, then $D_A^i = close(D_A^{i-1} \cup \{N\})$, where N is the fresh number symbol. If γ_M occurs in $v_{D_A^{i-1}}(\mathbf{a}_j)$ then M is D_A^{i-1}-opaque or else it occurs only in non-analyzable positions and $M \notin D_A^{i-1}$. In the former case, $M \in sub(D_A^{i-1})$ because M is a submessage of the message present in $\mathbf{a}_j$, which is certainly an element of $D_A^j \subseteq D_A^{i-1}$. Therefore, Lemma 40 guarantees that M is D_A^i-opaque. In the latter case, property (2) of Lemma 40, guarantees that $M \notin D_A^i$. Thus, it follows from Lemma 34 that $v_{D_A^{i-1}}(\mathbf{a}_j) = v_{D_A^i}(\mathbf{a}_j)$.
- Finally, let us assume that $\mathbf{a}_i$ is a receiving action.
  - If $\mathbf{a}_j$ is a fresh generation action, then obviously $v_{D_A^{i-1}}(\mathbf{a}_j) = v_{D_A^i}(\mathbf{a}_j)$.
  - If $\mathbf{a}_j$ is also a receiving action, then, by assumption, all the ghost symbols in $v_{D_A^{i-1}}(\mathbf{a}_j)$ also occur in $v_{D_A^i}(\mathbf{a}_j)$. Thus the equality of the views follows from Lemma 34.
  - If $\mathbf{a}_j$ is a sending action, then, as the protocol is executable, the message M_j being sent must be in $D_A^j \subseteq D_A^{i-1}$. Clearly for each ghost symbol γ_M that occurs in $v_{D_A^{i-1}}(M_j)$ it must be the case that M is D_A^{i-1}-opaque or it occurs only in non-analyzable positions and $M \notin D_A^{i-1}$. In the former case, since $D_A^{i-1} = close(D_A^0 \cup S)$, where S collects all the messages sent or freshly generated up to the $(i-1)$th action, Lemma 35 ensures that in this case M must appear as a submessage of one of the received messages in S, and thus Lemma 38 guarantees that γ_M also occurs in the view at $i-1$ of some of these received messages. But, by assumption, these are preserved up to the view at step i, so M must also be D_A^i-opaque. Using a similar argument, if $M \notin D_A^{i-1}$ one can also conclude that $M \notin D_A^i$. Finally, it follows from Lemma 34 that $v_{D_A^{i-1}}(M_j) = v_{D_A^i}(M_j)$.

  A simple inductive argument now shows that $v_{D_A^j}(\mathbf{a}_j) = v_{D_A^{j+1}}(\mathbf{a}_j) = \cdots = v_{D_A^{i-1}}(\mathbf{a}_j) = v_{D_A^i}(\mathbf{a}_j)$, and the representativity of *A-srun* follows. □

Note that in this case only message forwarding may be necessary. The precise meaning of *forwarding* can also be clarified with the help of Lemmas 34, 35, and 38, as explained in the proof of Proposition 22: if a sent message contains an opaque submessage M then M must also occur, and be opaque, in some previously received message.

We now turn to the correctness of the operational semantics with respect to the denotational semantics obtained by fine interpretation.

Proposition 27. *Let $D_A^0, \ldots, D_A^s$ be the evolving datasets corresponding to A's run of the protocol. For every i such that $0 \leqslant i \leqslant s$, if $M \in D_A^i$ then*

(1) $cf_{\chi_i}(M) \in synth(\chi_i)$,

(2) $\sigma_k(cf_{\chi_i}(M)) = v_{D_A^k}(M)$, *for each* $i \leqslant k \leqslant s$, *and*

(3) *if* $face(M) \in synth(\chi_i)$ *then* $\sigma_k(face(M)) = v_{D_A^k}(M)$, *for each* $i \leqslant k \leqslant s$.

Proof. (1) The proof proceeds by induction on i. Clearly, we have that $D_A^0 = close(Data_A) = synth(Data_A) = synth(\chi_0)$. Moreover, $cf_{\chi_0}(M) = M$ for every $M \in Data_A$. Hence if $M \in D_A^0$ then $cf_{\chi_0}(M) \in synth(\chi_0)$.

Assume now, by the induction hypothesis, that if $M \in D_A^{i-1}$ then $cf_{\chi_{i-1}}(M) \in synth(\chi_{i-1})$. We have to consider the three possibilities for $\mathbf{a}_i$.

- If $\mathbf{a}_i = \mathbf{s}(M', B)$ then $D_A^i = D_A^{i-1}$, $\chi_i = \chi_{i-1}$, and the result follows by the induction hypothesis.
- If $\mathbf{a}_i = \mathbf{f}(N)$ then $D_A^i = synth(D_A^{i-1} \cup \{N\})$, using property (1) of Lemma 40, and $\chi_i = \chi_{i-1} \cup \{N\}$. By the induction hypothesis, if $M \in D_A^{i-1}$ then it is constructible. By the addition of N to χ_i, N is also constructible. Moreover, we can use the synthesis rules, so if $M \in D_A^i$ then $cf_{\chi_i}(M) \in synth(\chi_i)$.
- If $\mathbf{a}_i = \mathbf{r}(M')$ then $D_A^i = synth(D_A^{i-1} \cup \{M'\})$ and χ_i extends χ_{i-1}. We prove that if $M \in D_A^i$ then $cf_{\chi_i}(M) \in synth(\chi_i)$ by induction on the closure rules.
  - If $M \in D_A^{i-1}$ then the result follows by the induction hypothesis.
  - If $M = M'$, the received message, then we do $inp\ net_A\ (\exists_{\chi_{i-1}})face(M)$. Now, $M = M_1; \ldots; M_k$ where each M_j for $1 \leqslant j \leqslant k$ is an atomic, encrypted, hashed, or inverse message, and the definition of $face(M_1)$ will force all the atoms and ghost symbols of the complex messages M_j to be in χ_i. By the definition of cf, it is clear that each $cf_{\chi_i}(M_j) \in synth(\chi_i)$. Take, for instance, an atomic M_j, then $cf_{\chi_i}(M_j) = M_j \in \chi_i$. Take now a complex

M_j for which $cf_{\chi_i}(M_j) = \gamma_{M_j}$. Clearly, $\gamma_{M_j} \in \chi_i$. Finally, consider a complex M_j such that $cf_{\chi_i}(M_j) \neq \gamma_{M_j}$ and note that all the possibilities in the definition of $cf_{\chi_i}(M_j)$ guarantee that it belongs to $synth(\chi_i)$.

- If M is synthesized from messages in D_A^i then, by the induction hypothesis, the construction of these messages is in $synth(\chi_i)$, and so is the construction of the whole message as well.
- If M is analyzed from M; $M_1 \in D_A^i$ (and analogously for M_1; $M \in D_A^i$) then, by the induction hypothesis, $cf_{\chi_i}(M; M_1) = cf_{\chi_i}(M)$; $cf_{\chi_i}(M_1) \in synth(\chi_i)$. Since basic sets like $synth(\chi_i)$ do not contain paired messages, $cf_{\chi_i}(M)$; $cf_{\chi_i}(M_1)$ must have been synthesized from $cf_{\chi_i}(M)$, $cf_{\chi_i}(M_1) \in synth(\chi_i)$.
- If M is analyzed from $\{\!|M|\!\}_K^{\mathrm{a}}$, $K^{-1} \in D_A^i$ then, by the induction hypothesis, $cf_{\chi_i}(\{\!|M|\!\}_K^{\mathrm{a}})$, $cf_{\chi_i}(K^{-1}) \in synth(\chi_i)$. If $cf_{\chi_i}(\{\!|M|\!\}_K^{\mathrm{a}}) = \{\!|cf_{\chi_i}(M)|\!\}_{cf_{\chi_i}(K)}^{\mathrm{a}}$ then $cf_{\chi_i}(M) \in synth(\chi_i)$, as we wanted. Otherwise, $cf_{\chi_i}(\{\!|M|\!\}_K^{\mathrm{a}}) = \gamma_{\{\!|M|\!\}_K^{\mathrm{a}}}$ and there are three possibilities for $cf_{\chi_i}(K^{-1})$.

  If $cf_{\chi_i}(K^{-1}) = face(K)^{-1} \in \chi_i$ then the process includes the check $chk(\gamma_{\{\!|M|\!\}_K^{\mathrm{a}}} \mid (\exists \ldots)\{\!|face(M)|\!\}_{face(K)}^{\mathrm{a}})$.

  If $cf_{\chi_i}(K^{-1}) = \gamma_K^{-1}$ with $\gamma_K^{-1} \in \chi_i$, then the process includes the check $chk(\gamma_{\{\!|M|\!\}_K^{\mathrm{a}}} \mid (\exists \ldots)\{\!|face(M)|\!\}_{\gamma_k}^{\mathrm{a}})$ and $cf_{\chi_i}(M) \in synth(\chi_i)$ follows, as above in the case of the received message.

  If $cf_{\chi_i}(K^{-1}) = \gamma_{K^{-1}} \in \chi_i$, then the process includes the joint check $chk(\gamma_{K^{-1}}; \gamma_{\{\!|M|\!\}_K^{\mathrm{a}}} \mid (\exists \ldots)\gamma_K^{-1}; \{\!|face(M)|\!\}_{\gamma_k}^{\mathrm{a}})$. In any case, from $face(M)$ it then follows that $cf_{\chi_i}(M) \in synth(\chi_i)$, using the same argument as above in the case of the received message.
- If M is analyzed from $\{\!|M|\!\}_{K^{-1}}^{\mathrm{a}}$, $K \in D_A^i$, then, by the induction hypothesis, $cf_{\chi_i}(\{\!|M|\!\}_{K^{-1}}^{\mathrm{a}})$, $cf_{\chi_i}(K) \in synth(\chi_i)$. If we have $cf_{\chi_i}(\{\!|M|\!\}_{K^{-1}}^{\mathrm{a}}) = \{\!|cf_{\chi_i}(M)|\!\}_{cf_{\chi_i}(K^{-1})}^{\mathrm{a}}$, then $cf_{\chi_i}(M) \in synth(\chi_i)$, as we wanted. Otherwise, $cf_{\chi_i}(\{\!|M|\!\}_{K^{-1}}^{\mathrm{a}}) = \gamma_{\{\!|M|\!\}_{K^{-1}}^{\mathrm{a}}}$ and the process includes the check $chk(\gamma_{\{\!|M|\!\}_{K^{-1}}^{\mathrm{a}}} \mid (\exists \ldots)\{\!|face(M)|\!\}_{cf_{\chi_i}(K)^{-1}}^{\mathrm{a}})$. From $face(M)$, it then follows that $cf_{\chi_i}(M) \in synth(\chi_i)$, using the same argument as above.
- If M is analyzed from $\{\!|M|\!\}_K^{\mathrm{s}}$, $K \in D_A^i$ then, by the induction hypothesis, $cf_{\chi_i}(\{\!|M|\!\}_K^{\mathrm{s}})$, $cf_{\chi_i}(K) \in synth(\chi_i)$. If $cf_{\chi_i}(\{\!|M|\!\}_K^{\mathrm{s}}) = \{\!|cf_{\chi_i}(M)|\!\}_{cf_{\chi_i}(K)}^{\mathrm{s}}$ then $cf_{\chi_i}(M) \in synth(\chi_i)$, as we wanted. Otherwise, $cf_{\chi_i}(\{\!|M|\!\}_K^{\mathrm{s}}) = \gamma_{\{\!|M|\!\}_K^{\mathrm{s}}}$ and the process includes $chk(\gamma_{\{\!|M|\!\}_K^{\mathrm{s}}} \mid (\exists \ldots)\{\!|face(M)|\!\}_{cf_{\chi_i}(K)}^{\mathrm{s}})$. From $face(M)$, it follows that $cf_{\chi_i}(M) \in synth(\chi_i)$, using the same argument as above.

(2) First note that all the elements of χ_i have a counterpart in D_A^i. Namely, if $\gamma_M \in \chi_i$ them $M \in D_A^i$, if $\gamma_K^{-1} \in \chi_i$ or $face(K)^{-1} \in \chi_i$ then $K^{-1} \in D_A^i$, and every atom in χ_i is also in D_A^i. This fact can be proved by detailed inspection of all possible checks, but it should be clear that all the elements of χ_i appear in analyzable positions of received messages. Therefore, if $face(M) \in synth(\chi_i)$ or $cf_{\chi_i}(M) \in synth(\chi_i)$, then we have that $M \in D_A^i$.

Now, we prove that $\sigma_i(\gamma_M) = v_{D_A^i}(M)$ whenever $\gamma_M \in \chi_i$, or $face(M) \in synth(\chi_i)$ and $\gamma_M^{-1} \in \chi_i$, and $\sigma_i(\gamma_M) = \gamma_M$ otherwise. The proof proceeds by induction on i.

- Obviously, no γ_M occurs in $\chi_0 = Data_A$ and $\sigma_0 = []$. Therefore, $\sigma_0(\gamma_M) = \gamma_M$, as it should.
- We now consider the three possible forms for the ith action.
  - If $\mathbf{a}_i = \mathbf{s}(M', B)$ then $\chi_i = \chi_{i-1}$, $\sigma_i = \sigma_{i-1}$, $D_A^i = D_A^{i-1}$, and the result follows from the induction hypothesis.
  - If $\mathbf{a}_i = \mathbf{f}(N)$ then $\chi_i = \chi_{i-1} \cup \{N\}$, $\sigma_i = \sigma_{i-1}$, $D_A^i = close(D_A^{i-1} \cup \{N\}) = synth(D_A^{i-1} \cup \{N\})$, by using the proof of Lemma 40. If $\gamma_M \in \chi_i$, or $face(M) \in synth(\chi_i)$ and $\gamma_M^{-1} \in \chi_i$, then that must have been the case already for χ_{i-1}. Thus, using the induction hypothesis, $\sigma_i(\gamma_M) = \sigma_{i-1}(\gamma_M) = v_{D_A^{i-1}}(M) = v_{D_A^i}(M)$, again using Lemma 40.
  - If $\mathbf{a}_i = \mathbf{r}(M)$ then $\sigma_i = \rho_i \circ \sigma_{i-1}$, where ρ_i is obtained by composing the substitutions obtained in each of checks. Hence, given Lemma 39, the desired property for σ_i boils down to showing the same property for ρ_i. If $\gamma_M \in \chi_i$ is not checked then it means that M is D_A^i-opaque and so $\rho_i(\gamma_M) = \gamma_M = v_{D_A^i}(M)$. If $\gamma_M^{-1} \in \chi_i$ and γ_M is not checked then it means that $cf_{\chi_i}(M) \notin synth(\chi_i)$, which implies that $M \notin D_A^i$, in which case also $\rho_i(\gamma_M) = \gamma_M$, as it should.

    Thus, we now need to show that for the remaining γ_M occurring in χ_i, the checks done are enough to guarantee that $\rho_i(\gamma_M) = v_{D_A^i}(M)$. Since all possible checks will be done, by definition of spi_i^A, it is enough to verify that each check performed, namely on γ_M, indeed "opens" the message and allows a principal to compute its view. Let us analyze each of the three possible kinds of checks.

In the first case, the check is $chk(\gamma_M \mid (\exists \ldots) inface_{\chi_{i-1}^k}(M))$ with $\gamma_M \in \chi_i$ and M parsable according to $inface_{\chi_{i-1}^k}(M)$. An inspection of its definition guarantees that everything works, given the properties of *cf* and *face* proved in this proposition. An exhaustive analysis of all possible cases is straightforward and is thus omitted.

In the second case, the check that is performed is $chk(\gamma_{K^{-1}}; \gamma_{\{\!|M|\!\}_K^a} \mid (\exists \ldots)\gamma_K^{-1}; \{\!|face(M)|\!\}_{\gamma_K}^a)$ with $\gamma_{K^{-1}}, \gamma_{\{\!|M|\!\}_K^a} \in \chi_i$. Clearly, a subsequent check for γ_K happens iff $K \in D_A^i$. Thus, if $K \notin D_A^i$ then $\rho_i(\gamma_{K^{-1}}) = \gamma_K^{-1} = v_{D_A^i}(K^{-1})$ and $\rho_i(\gamma_{\{\!|M|\!\}_K^a}) = \{\!|\rho_i(face(M))|\!\}_{\rho_i(\gamma_K)}^a) = \{\!|v_{D_A^i}(M)|\!\}_{\gamma_K}^a = v_{D_A^i}(\{\!|M|\!\}_K^a)$. Otherwise, $\rho_i(\gamma_{K^{-1}}) = \rho_i(\gamma_K^{-1}) = \rho_i(\gamma_K)^{-1} = v_{D_A^i}(K)^{-1} = v_{D_A^i}(K^{-1})$ and $\rho_i(\gamma_{\{\!|M|\!\}_K^a}) = \{\!|\rho_i(face(M))|\!\}_{\rho_i(\gamma_K)}^a = \{\!|v_{D_A^i}(M)|\!\}_{v_{D_A^i}(K)}^a = v_{D_A^i}(\{\!|M|\!\}_K^a)$.

In the third and final case, the check that is performed is $chk(\gamma_M \mid (\exists \ldots) cf_{\chi_{i-1}^k}(M))$ with $\gamma_M \notin \chi_{i-1}^k$, $\gamma_M^{-1} \in \chi_i$ and $cf_{\chi_{i-1}^k}(M) \in synth(\chi_i)$. Hence, $\rho_i(\gamma_M) = \rho_i(cf_{\chi_{i-1}^k}(M)) = v_{D_A^i}(M)$.

Using this fact, together with the definition of *cf*, we can now prove by induction on M that $\sigma_k(cf_{\chi_i}(M)) = v_{D_A^k}(M)$.

- If M is atomic then $cf_{\chi_i}(M) = M$, $\sigma_k(M) = M$, and $v_{D_A^k}(M) = M$.
- If $M = M_1; M_2$ then $\sigma_k(cf_{\chi_i}(M_1; M_2)) = \sigma_k(cf_{\chi_i}(M_1); cf_{\chi_i}(M_2)) = \sigma_k(cf_{\chi_i}(M_1)); \sigma_k(cf_{\chi_i}(M_2))$, which equals $v_{D_A^k}(M_1); v_{D_A^k}(M_2) = v_{D_A^k}(M_1; M_2)$, by the induction hypothesis.
- If $M = \{\!|M_1|\!\}_K$ then there are two possibilities. First, if $cf_{\chi_i}(\{\!|M_1|\!\}_K) = \{\!|cf_{\chi_i}(M_1)|\!\}_{cf_{\chi_i}(K)}$ then $\sigma_k(\{\!|cf_{\chi_i}(M_1)|\!\}_{cf_{\chi_i}(K)}) = \{\!|\sigma_k(cf_{\chi_i}(M_1))|\!\}_{\sigma_k(cf_{\chi_i}(K))}$, which equals, by the induction hypothesis, $\{\!|v_{D_A^k}(M_1)|\!\}_{v_{D_A^k}(K)} = v_{D_A^k}(\{\!|M_1|\!\}_K)$, because $cf_{\chi_i}(M_1), cf_{\chi_i}(K) \in synth(\chi_i)$ imply that $M_1, K \in D_A^i \subseteq D_A^k$. If $cf_{\chi_i}(\{\!|M_1|\!\}_K) = \gamma_M \in \chi_i \subseteq \chi_k$ then we know that $\sigma_k(\gamma_M) = v_{D_A^k}(M)$.
- If $M = H(M_1)$ then there are two possibilities. If we have $cf_{\chi_i}(H(M_1)) = H(cf_{\chi_i}(M_1))$ then $\sigma_k(H(cf_{\chi_i}(M_1))) = H(\sigma_k(cf_{\chi_i}(M_1)))$, which equals, by the induction hypothesis, $H(v_{D_A^k}(M_1)) = v_{D_A^k}(H(M_1))$, because $cf_{\chi_i}(M_1) \in synth(\chi_i)$ implies that $M_1 \in D_A^i \subseteq D_A^k$. If we have $cf_{\chi_i}(H(M_1)) = \gamma_M \in \chi_i \subseteq \chi_k$ then we know that $\sigma_k(\gamma_M) = v_{D_A^k}(M)$.
- If $M = K^{-1}$ then there are three possibilities. If $cf_{\chi_i}(K^{-1}) = cf_{\chi_i}(K)^{-1}$ then we must have checked $chk(\gamma_K \mid (\exists \ldots) face(K))$, and thus $\gamma_K^{-1} \in \chi_i$ and $face(K) \in synth(\chi_i)$. So, by the induction hypothesis, $\sigma_k(cf_{\chi_i}(K)a^{-1}) = \sigma_k(cf_{\chi_i}(K))^{-1} = v_{D_A^k}(K)^{-1} = v_{D_A^k}(K^{-1})$ as $\gamma_K^{-1} \in \chi_i$ implies that $K^{-1} \in D_A^i \subseteq D_A^k$. Otherwise, we can have the second possibility, i.e., $cf_{\chi_i}(K^{-1}) = \gamma_K^{-1}$, provided that some $\gamma_{\{\!|M'|\!\}_K^a} \in \chi_i$. If that is the case, then we have checked $chk(\gamma_{K^{-1}}; \gamma_{\{\!|M'|\!\}_K^a} \mid (\exists \ldots)\gamma_K^{-1}; \{\!|face(M_1)|\!\}_{\gamma_K}^a)$. Therefore, $\sigma_k(\gamma_K^{-1}) = \sigma_k(\gamma_K)^{-1}$. But now, if $K \in D_A^k$ then $\gamma_K \in \chi_k$ and $\sigma_k(\gamma_K)^{-1} = v_{D_A^k}(K)^{-1} = v_{D_A^k}(K^{-1})$. If not, then $\sigma_k(\gamma_K)^{-1} = \gamma_K^{-1} = v_{D_A^k}(K^{-1})$. Finally, if $cf_{\chi_i}(K^{-1}) = \gamma_M \in \chi_i \subseteq \chi_k$ then we know that $\sigma(\gamma_M) = v_{D_A^k}(M)$.

(3) Clearly, $face(M) \in synth(\chi_i)$ means that all outermost atoms of M are in χ_i, and all outermost encrypted, hashed, or inverse submessages of M have their corresponding ghost symbol in χ_i. The fact that $\sigma_k(face(M)) = v_{D_A^k}(M)$ follows immediately employing the result used in the proof of property (2) above. □

Proposition 28. *For every participant A of an executable Alice&Bob protocol specification, the process A-proc is implementable.*

Proof. The fact that all the patterns in the process are transparent can be shown by a simple inspection of each of the possible cases, according to Definitions 23, 25, and 26. The constructibility of every sent message follows immediately from property (1) of Proposition 27. □

Proposition 32. $den(spi_1^A(\mathbf{a}_1); \ldots; spi_i^A(\mathbf{a}_i)) = A\text{-}isrun^i$, *for every i such that* $1 \leqslant i \leqslant s$.

Proof. Clearly it suffices to show that $\sigma_i(act(spi_k^A(\mathbf{a}))) = v_{D_A^i}(\mathbf{a})$, for every action $\mathbf{a}$, and $1 \leqslant k \leqslant i \leqslant s$. We consider each of the three possibilities for action $\mathbf{a}$.

- If $\mathbf{a} = \mathbf{f}(N)$, then $\sigma_i(act(spi_k^A(\mathbf{f}(N)))) = \sigma_i(act(new\ N : Num)) = \sigma_i(\mathbf{f}(N)) = \mathbf{f}(\sigma_i(N)) = \mathbf{f}(N) = v_{D_A^i}(\mathbf{f}(N))$, since N is atomic and $v_{D_A^i}(N) = N$.
- If $\mathbf{a} = \mathbf{s}(M, B)$, then $\sigma_i(act(spi_k^A(\mathbf{s}(M, B)))) = \sigma_i(act(out\ net_B\ cf_{\chi_k}(M))) = \sigma_i(\mathbf{s}(cf_{\chi_k}(M), B) = \mathbf{s}(\sigma_i(cf_{\chi_k}(M)), \sigma_i(B)) = \mathbf{s}(v_{D_A^i}(M), B) = v_{D_A^i}(\mathbf{s}(M, B))$, since B is atomic and $v_{D_A^i}(B) = B$, and using property (2) of Proposition 27.
- If $\mathbf{a} = \mathbf{r}(M)$, then we have that $\sigma_i(act(spi_k^A(\mathbf{r}(M)))) = \sigma_i(act(inp\ net_A\ (\exists_{\chi_k})\, face(M); \ldots)) = \sigma_i(\mathbf{r}(face(M))) = \mathbf{r}(\sigma_i(face(M))) = \mathbf{r}(v_{D_A^i}(M)) = v_{D_A^i}(\mathbf{r}(M))$, using property (3) of Proposition 27.

Then

$$
\begin{array}{lcl}
den(spi_1^A(\mathbf{a}_1);\ \ldots;\ spi_i^A(\mathbf{a}_i)) & = & \text{(by def.)} \\
\langle \sigma_i(act(spi_1^A(\mathbf{a}_1));\ \ldots;\ \sigma_i(act(spi_i^A(\mathbf{a}_i))\rangle & = & \text{(by the above)} \\
\langle v_{D_A^i}(\mathbf{a}_1);\ \ldots;\ v_{D_A^i}(\mathbf{a}_i)\rangle & = & \text{(by def.)} \\
A\text{-}isrun^i. \quad \square & &
\end{array}
$$

References

[1] M. Abadi, A.D. Gordon, A calculus for cryptographic protocols: the spi calculus, Inform. Comput. 148 (1999) 1–70.

[2] M. Abadi, P. Rogaway, Reconciling two views of cryptography (the computational soundness of formal encryption), J. Cryptology 15 (2) (2002) 103–127.

[3] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P.-C. Heám, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, L. Vigneron, The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications, in: Proc. CAV'05, Lecture Notes in Computer Science, Vol. 3576, Springer, Berlin, 2005, pp. 281–285.

[4] A. Armando, D. Basin, M. Bouallagui, Y. Chevalier, L. Compagna, S. Mödersheim, M. Rusinowitch, M. Turuani, L. Viganò, L. Vigneron, The AVISS security protocol analysis tool, in: Proc. CAV'02, Lecture Notes in Computer Science, Vol. 2404, Springer, Berlin, 2002, pp. 349–354.

[5] N. Asokan, V. Shoup, M. Waidner, Asynchronous protocols for optimistic fair exchange, in: Proc. IEEE Symp. on Research in Security and Privacy, IEEE Computer Society Press, Silver Spring, MD, 1998, pp. 86–99.

[6] D. Basin, G. Denker, Maude versus Haskell: an experimental comparison in security protocol analysis, in: Proc. WRLA'00, ENTCS, Vol. 36, 2001, pp. 235–256.

[7] D. Basin, S. Mödersheim, L. Viganò, OFMC: a symbolic model checker for security protocols, Internat. J. Inform. Security 4 (3) (2005) 181–208.

[8] B. Blanchet, An efficient cryptographic protocol verifier based on prolog rules, in: Proc. CSFW'01, IEEE Computer Society Press, Silver Spring, MD, 2001, pp. 82–96.

[9] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, H. Riis Nielson, Static validation of security protocols, J. Comput. Security 13 (3) (2005) 347–390.

[10] M. Boreale, M.G. Buscemi, A framework for the analysis of security protocols, in: Proc. CONCUR 2002, Lecture Notes in Computer Science, Vol. 2421, Springer, Berlin, 2002, pp. 483–498.

[11] S. Brackin, C. Meadows, J. Millen, CAPSL interface for the NRL protocol analyzer, in: Proc. ASSET'99, IEEE Computer Society Press, Silver Spring, MD, 1999.

[12] S. Briais, U. Nestmann, A formal semantics for protocol narrations, in: Proc. TGC'05, Lecture Notes in Computer Science, Vol. 3705, Springer, Berlin, 2005, pp. 163–181.

[13] M. Buchholtz, H. Riis Nielson, F. Nielson, A calculus for control flow analysis of security protocols, Internat. J. Inform. Security 2 (3–4) (2004) 145–167.

[14] C. Caleiro, L. Viganò, D. Basin, Deconstructing Alice and Bob, in: Proc. ARSPA'05, ENTCS, Vol. 135(1), 2005, pp. 3–22.

[15] C. Caleiro, L. Viganò, D. Basin, Metareasoning about security protocols using distributed temporal logic, in: Proc. ARSPA'04, ENTCS, Vol. 125(1), 2005, pp. 67–89.

[16] Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, J. Mantovani, S. Mödersheim, L. Vigneron, A high level protocol specification language for industrial security-sensitive protocols, in: Proc. SAPS'04, Austrian Computer Society, 2004, pp. 193–205.

[17] Y. Chevalier, R. Küsters, M. Rusinowitch, M. Turuani, An NP decision procedure for protocol insecurity with XOR, Theor. Comput. Sci. 338 (1–3) (2005) 247–274.

[18] J. Clark, J. Jacob, A survey of authentication protocol literature: version 1.0, 17. November 1997, URL: ⟨www.cs.york.ac.uk/~jac/papers/www.drareview.ps.gz⟩.

[19] H. Comon-Lundh, V. Shmatikov, Intruder deductions, constraint solving and insecurity decision in presence of exclusive or, in: Proc. LICS'03, IEEE Computer Society Press, Silver Spring, MD, 2003.

[20] R. Corin, S. Etalle, An improved constraint-based system for the verification of security protocols, in: Proc. SAS'02, Lecture Notes in Computer Science, Vol. 2477, Springer, Berlin, 2002.

[21] F. Crazzolara, G. Winskel, Composing strand spaces, in: Proc. FST TCS 2002, Lecture Notes in Computer Science, Vol. 2556, Springer, Berlin, 2002, pp. 97–108.

[22] C. Cremers, Scyther documentation, ⟨http://www.win.tue.nl/~ccremers/scyther⟩.

[23] W. Damm, D. Harel, LSCs: breathing life into message sequence charts, Formal Methods in System Design 19 (1) (2001) 45–80.

[24] G. Denker, J. Millen, H. Rueß, The CAPSL integrated protocol environment, Technical Report SRI-CSL-2000-02, SRI International, Menlo Park, CA, 2000.
[25] D. Dolev, A. Yao, On the security of public key protocols, IEEE Trans. Inform. Theory 29 (2) (1983) 198–208.
[26] N. Durgin, J.C. Mitchell, D. Pavlovic, A compositional logic for proving security properties of protocols, J. Comput. Security 11 (2003) 677–721.
[27] A.D. Gordon, A. Jeffrey, Authenticity by typing for security protocols, in: Proc. CSFW'01, IEEE Computer Society Press, Silver Spring, MD, 2001, pp. 145–159.
[28] C. Haack, A. Jeffrey, Pattern-matching spi-calculus, in: Proc. FAST'04, IFIP series 173, Kluwer Academic Press, Dordrecht, 2004, pp. 193–205.
[29] J.Y. Halpern, R. Pucella, On the relationship between strand spaces and multi-agent systems, ACM Trans. Inform. System Security 6 (1) (2003) 43–70.
[30] ITU-T Recommendation Z.120: Message Sequence Chart (MSC), 2000.
[31] F. Jacquemard, M. Rusinowitch, L. Vigneron, Compiling and verifying security protocols, in: Proc. LPAR 2000, Lecture Notes in Computer Science, Vol. 1955, Springer, Berlin, 2000, pp. 131–160.
[32] G. Lowe, Casper: a compiler for the analysis of security protocols, J. Comput. Security 6 (1) (1998) 53–84.
[33] S. Mauw, M. Reniers, T. Willemse, Message sequence charts in the software engineering process, in: Handbook of Software Engineering and Knowledge Engineering, World Scientific Publishing Co., Singapore, 2001, pp. 437–463.
[34] C. Meadows, Formal methods for cryptographic protocol analysis: emerging issues and trends, IEEE J. Sel. Areas Comm. 21 (1) (2003) 44–54.
[35] L. Paulson, The inductive approach to verifying cryptographic protocols, J. Comput. Security 6 (1998) 85–128.
[36] R.L. Rivest, A. Shamir, L.M. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Comm. ACM 21 (2) (1978) 120–126.
[37] P. Rumbaugh, I. Jacobson, G. Booch, The Unified Modeling Language Reference Manual, Addison-Wesley, Reading, MA, 1999.
[38] P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, B. Roscoe, Modelling and Analysis of Security Protocols, Addison-Wesley, Reading, MA, 2000.
[39] D. Song, S. Berezin, A. Perrig, Athena: a novel approach to efficient automatic security protocol analysis, J. Comput. Security 9 (2001) 47–74.
[40] F.J. Thayer Fábrega, J.C. Herzog, J.D. Guttman, Strand spaces: proving security protocols correct, J. Comput. Security 7 (1999) 191–230.
[41] L. Viganò, Automated security protocol analysis with the AVISPA tool, in: Proc. MFPS'05, ENTCS, Vol. 155, 2006, pp. 61–86.
[42] Wikipedia: Alice and Bob, URL: ⟨www.wikipedia.org/wiki/Alice_and_Bob⟩.