A Comparison between Strand Spaces and Multiset Rewriting for Security Protocol Analysis*

Iliano Cervesato
ITT Industries, Inc.
iliano@itd.nrl.navy.mil

Nancy A. Durgin
Sandia National Laboratories
nadurgi@sandia.gov

Patrick D. Lincoln
SRI International
lincoln@csl.sri.com

John C. Mitchell
Stanford University
jcm@cs.stanford.edu

Andre Scedrov
University of Pennsylvania
scedrov@cis.upenn.edu

Abstract

Formal analysis of security protocols is largely based on a set of assumptions commonly referred to as the Dolev-Yao model. Two formalisms that state the basic assumptions of this model are related here: strand spaces and multiset rewriting with existential quantification. Strand spaces provide a simple and economical approach to analysis of completed protocol runs by emphasizing causal interactions among protocol participants. The multiset rewriting formalism provides a very precise way of specifying finite-length protocols with unboundedly many instances of each protocol role, such as client, server, initiator, or responder. A number of modifications to each system are required to produce a meaningful comparison. In particular, we extend the strand formalism with a way of incrementally growing bundles in order to emulate an execution of a protocol with parametric strands. The correspondence between the modified formalisms directly relates the intruder theory from the multiset rewriting formalism to the penetrator strands. The relationship we illustrate here between multiset rewriting specifications and strand spaces thus suggests refinements to both frameworks, and deepens our understanding of the Dolev-Yao model.

Keywords: Cryptographic Protocols, Specification Methodologies, Strand Spaces, Multiset Rewriting.

1 Introduction

The late 1990's saw a burst of research in security protocol analysis which yielded theoretical insight [16] and enhanced verification techniques [20, 29]. The cornerstone of these endeavors was a rising tide of formal notational frameworks for protocols, among them the spi-calculus [1], strand spaces [33] and multiset rewriting [10]. A first attempt to sort out the diverse languages, security assumptions, and execution models, appeared in an early version of this work [9], which focused on the relationship between strand

*Partial support for various authors by OSD/ONR CIP/SW URI "Software Quality and Infrastructure Protection for Diffuse Computing" through ONR Grant N00014-01-1-0795, by NRL under contract N00173-00-C-2086, by DoD MURI "Semantic Consistency in Information Exchange" as ONR grant N00014-97-1-0505 and by NSF grants CCR-9509931, CCR-9629754, CCR-9800785, CCR-0098096, and INT98-15731.

spaces [33, 34, 23] and a formalism based on multiset rewriting [10]. In the years that followed [9], researchers built several more bridges between languages for cryptographic protocol analysis, producing an almost complete map by now: strand spaces and multiset rewriting are tied to linear logic in [8]; mappings between process algebra, Petri nets (a close relative of multiset rewriting), strand spaces and inductive models are given in [11]; strand spaces and BAN logic are related in [32]; similarities between strand spaces and multi-agent systems are investigated in [21]; and the relation between multiset rewriting and process algebraic specifications is analyzed in [3, 27]. These theoretical investigations allow researchers to understand precisely how their results are related, often enabling a direct transfer of properties such as secrecy and many forms of authentication as most of these formalisms ultimately rely on a trace-based semantics. This observation is put into practice in the CAPSL Intermediate Language (CIL — another close relative of multiset rewriting) [14] and the numerous "connectors" translating CIL specifications to and from other languages and tools [5, 13, 25].

Protocol execution steps can be seen as inducing local changes to a global state consisting of messages in transit and the private data of each principal. This view was sharpened into a rigorous, formal language based on multiset rewriting with existential quantification [10, 16]: multiset rewrite rules represent protocol actions that can alter the portion of the global state visible to a principal. Messages and local data are represented symbolically in accordance with the Dolev-Yao abstraction [15, 28], while existential quantification, as commonly used in formal logic, provides a natural way of choosing new values, such as fresh keys or nonces. Protocol execution is carried out symbolically, with the behavior of the standard Dolev-Yao intruder explicitly implemented as rewrite rules. This model formed the core of CIL [14], which was designed as a neutral intermediate language for the exchange of specifications written for diverse verification tools [5, 13, 25]. This model also forms the core of the more recent MSR specification framework [7], an expressive and usable high-level language for describing cryptographic protocols.

Strand spaces [33, 34, 23] are a highly popular formalism for describing the result of a protocol execution as they visualize the causal interactions among individual steps. Roughly, the actions of each protocol participant are linearly ordered into strands, while a second spatial dimension connects complementary events of different principals, *e.g.* the transmission and reception of a given message. It should be observed that strand spaces were designed as a means to represent completed executions, both normal and malicious, and therefore offered only an indirect way of expressing protocols themselves. In this light, they provide a simple and succinct framework for state-based analysis of completed protocol runs. State space reduction techniques based on the strand space framework are utilized in an efficient automated checker, Athena [31]. It has recently been observed [11, 17] that strand spaces are closely related to process-based specification language such as the spi-calculus [1].

Since both multiset rewriting and strand spaces are used to specify cryptographic protocols and their (mis-)behaviors, one would expect them to be equivalent in some way. Producing a meaningful equivalence requires a heavy infrastructure and may be obtained only after a number of modifications are made in each setting. We shall trace these difficulties to two orthogonal aspects of these languages.

- First, they differ in their inherent focus: multiset rewriting provides a syntax to write protocols and a semantics to produce valid executions, while strand spaces offer static specifications of protocol executions, from which the protocols themselves can be glimpsed only in an indirect light.

  We choose to bring strand spaces on a par with multiset rewriting by endowing them with a syntax to express protocols as first-class objects, and a semantics that incrementally grows strand spaces (in the original sense) as an execution of a protocol unfolds. The resulting notion of parametric strand has since [9] been adopted under different names by numerous researchers, *e.g.* [11, 17].

  We also make minor changes to the multiset rewriting formalism, in particular with the elimination of the original "initialization phase" [10] which specifies rules to define principals and distribute shared,

public, or private keys. This information is now statically assumed as part of the initial state, in line with most protocol specification languages, including strand spaces, CIL and MSR.

- A second major difference derives from the underlying paradigm for specifying concurrent executions. As noted above, multiset rewriting is state-based, operating through local transformation on an explicit global state, while strand spaces are process-based with emphasis on communication between strands. This subtle distinction is a well-known problem in concurrency theory and has been repeatedly addressed, starting with [2]. It has recently been isolated relative to protocol specification in [3]. The material presented here specializes this account to strand spaces and multiset rewriting, and integrates it with the alterations outlined above.

The resolution of these two issues accounts for most of the machinery in this paper, with the rest taken up by minor syntactic differences. In the end, it provides effective procedures for faithfully translating the specification of security protocols, and in particular their execution, from the strand world into multiset rewriting, and vice versa. While comparing verification techniques is outside the scope of this paper, it is worth noting that the correspondence between executions in the two models allows swapping any trace-based property such as secrecy and most forms of authentication.

This paper is organized as follows: the multiset rewriting formalism is discussed in Section 2. In section 3, we introduce strand spaces and present our extensions. The translation from multiset rewriting to strand spaces is presented in Section 4. The reverse mapping, from strand spaces to multiset rewriting, is given in Section 5. Section 6 discusses related work, while Section 7 offers some concluding remarks.

2 Multiset Rewriting Theories

This section recalls basic notions pertaining to multiset rewriting (Section 2.1) and introduces the methodology by which they can be applied to specify cryptographic protocols (Section 2.2). We deviate from the original presentation [10] by eliminating the initialization phase in favor of persistent information given a priori. We also outline the transformation of generic specifications into *regular protocol theories* (Section 2.3), a step that will simplify the comparison with strand spaces. We conclude with a description of the intruder model tailored to simplify the comparison with strand spaces (Section 2.4).

2.1 First-Order Multiset Rewriting

A *multiset* $\boldsymbol{M}$ is an unordered collection of objects or *elements*, possibly with repetitions. The *empty multiset* does not contain any object and will be written "$\cdot$". We accumulate the elements of two multisets $\boldsymbol{M}$ and $\boldsymbol{N}$ by taking their *multiset union*, denoted "$\boldsymbol{M}, \boldsymbol{N}$". The elements we will consider here will be first-order atomic formulas $A(\vec{t})$ over some signature.

We will make use of the standard definitions pertaining to the variables of first-order logic. In particular, we write $\mathrm{Var}(A_0, \ldots, A_n)$ for the set of variables occurring in the multiset of atomic formulas $A_0, \ldots, A_n$. We say that a (multiset of) formula(s) is ground if no variable appear in it. Finally, *substitutions* (generally written δ, θ or ξ) are as usual mappings from variables to generic terms. We write $A[\delta]$ for the application of a substitution δ to a formula A, and use a similar notation for multisets of formulas.

In its simplest form, a *multiset rewrite rule* r is a pair of multisets $\boldsymbol{F}$ and $\boldsymbol{G}$, respectively called the *antecedent* and *consequent* of r. We will consider a slightly more elaborate notion in which $\boldsymbol{F}$ and $\boldsymbol{G}$ are multisets of first-order atomic formulas with variables among $\vec{x}$. We emphasize this aspect by writing them as $\boldsymbol{F}(\vec{x})$ and $\boldsymbol{G}(\vec{x})$. Furthermore, we shall be able to mark variables in the consequent so that they

are instantiated to *"fresh"* constants, that have not previously been encountered, even if the rule is used repeatedly. A rule assumes then the form

$$r: \ \boldsymbol{F}(\vec{x}) \rightarrow \exists \vec{n}.\, \boldsymbol{G}(\vec{x}, \vec{n})$$

where r is a label and $\exists \vec{n}$ indicates that the variables $\vec{n}$ are to be instantiated with constants that ought to be fresh. A *multiset rewriting system* $\mathcal{R}$ is a set of rewrite rules.

Rewrite rules allow transforming a multiset into another multiset by making localized changes to the elements that appear in it. Given a multiset of ground facts $\boldsymbol{M}$, a rule $r: \ \boldsymbol{F}(\vec{x}) \rightarrow \exists \vec{n}.\, \boldsymbol{G}(\vec{x}, \vec{n})$ is *applicable* if $\boldsymbol{M} = \boldsymbol{F}(\vec{t}), \boldsymbol{M}'$, for terms $\vec{t}$. Then, *applying* r to $\boldsymbol{M}$ yields the multiset $\boldsymbol{N} = \boldsymbol{G}(\vec{t}, \vec{c}), \boldsymbol{M}'$ where the constants $\vec{c}$ are fresh (in particular, they are distinct from any symbol appearing in $\boldsymbol{M}$ or r), $\vec{x}$ and $\vec{n}$ have been instantiated with $\vec{t}$ and $\vec{c}$ respectively, and the facts $\boldsymbol{F}(\vec{t})$ in $\boldsymbol{M}$ have been replaced with $\boldsymbol{G}(\vec{t}, \vec{c})$ to produce $\boldsymbol{N}$. Here, $\theta = [\vec{t}/\vec{x}]$ is the *matching substitution* of rule r with respect to $\boldsymbol{M}$, while $\xi = [\vec{c}/\vec{n}]$ is its *fresh constant substitution*. We write δ for the composite substitution (θ, ξ) and call it the *instantiating substitution* of rule r with respect to $\boldsymbol{M}$. We denote the application of a single rule and of zero or more rewrite rules from the rewriting system $\mathcal{R}$ by means of the *one-step* and *multistep transition* judgments:

$$\boldsymbol{M} \xrightarrow{(r,\delta)}_{\mathcal{R}} \boldsymbol{N} \qquad\qquad \boldsymbol{M} \xrightarrow{\vec{r},\vec{\delta}}{}^{*}_{\mathcal{R}} \boldsymbol{N}$$

respectively. The labels r and $\vec{r}$ identify which rule(s) have been applied together with its (their) instantiating substitution(s). Thus, $\vec{r}$ acts as a complete trace of the execution.

2.2 Protocol Theories

We model protocols by means of specifically tailored first-order multiset rewriting systems. We present here a simplified version of the model introduced in [10, 16]. We will further refine it in Section 2.3 to achieve a meaningful comparison with the strand formalism. We rely upon the following atomic formulas:

Persistent information: Data such as the identity of principals and their keys often constitute the stage on which the execution of a protocol takes place, and does not change as it unfolds. We will represent and access this *persistent information* through a fixed set of *persistent predicates* that we will indicate using a slanted font (*e.g.* KeyP, as opposed to N). A selection of persistent predicates is described in Appendix A.2.

In [10, 16], we described the choice of the persistent data by means of a set of multiset rewrite rules of a specific form, that we called the *initialization theory*. We showed that the application of these rules can be confined to an initialization phase that precedes the execution of any other rule. Let $\mathbf{\Pi}$ be the resulting set of ground facts.[1] Strand constructions assume instead that the persistent information is given up-front as a set. We reconcile the two approaches by dropping the explicit initialization phase of [10, 16] and assuming $\mathbf{\Pi}$ given. We will allow individual rules to query $\mathbf{\Pi}$ (but not to modify it). Therefore, for every rule, the persistent predicates appearing in its antecedent and consequent shall be identical. We will generically indicate them as $\boldsymbol{\pi}$, or $\boldsymbol{\pi}(\vec{x})$ when emphasizing the variables they may mention.

Network messages: Network messages are modeled by the predicate $\mathsf{N}(m)$, where m is the message being transmitted. Having a distinct network predicate for each message exchanged in a protocol specification, as done in [10, 16], is equivalent, but would obscure the translation in Section 5. Messages will consist of the class of terms freely generated from atomic messages (principal names, keys, nonces, etc.) by the operators of concatenation, denoted "$\_,\_$", and encryption, written "$\{\ \}_{\_}$". The detailed syntax of messages is presented in Appendix A.

[1] Constraints on the initialization theory prevent $\mathbf{\Pi}$ from containing duplicates [10, 16]

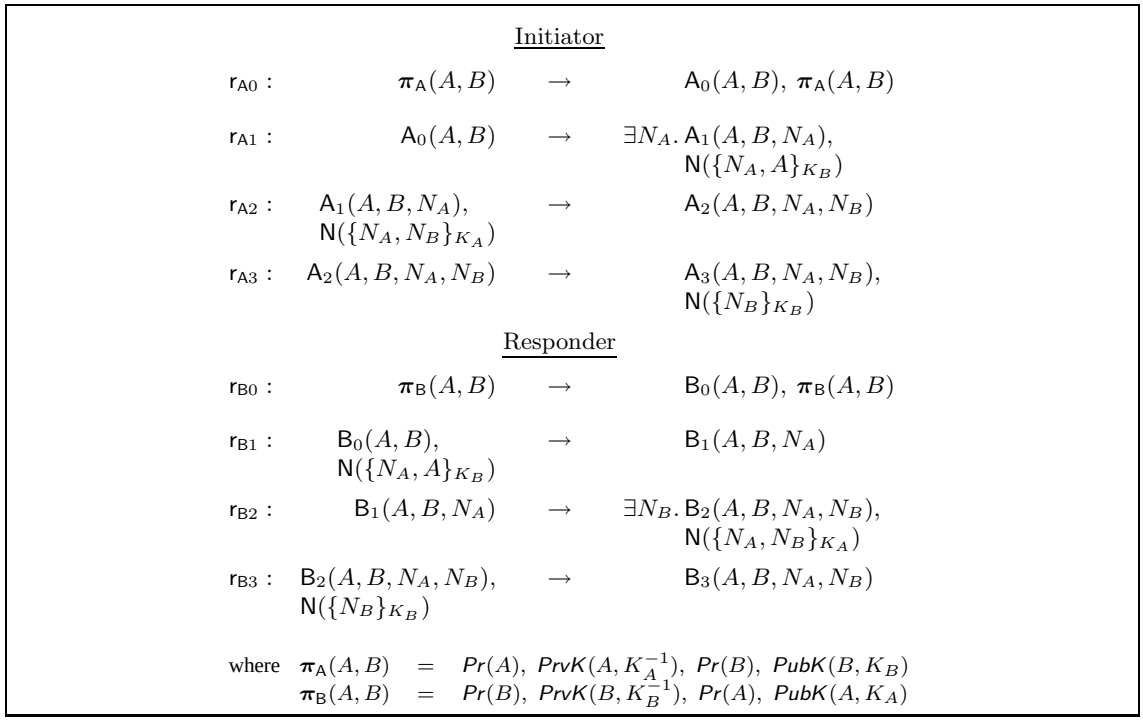


Figure 1. Multiset Rewriting Specification of the Needham-Schroeder Protocol

Role states: We first choose a set of *role identifiers* $\rho_1, \ldots, \rho_n$ for the different roles constituting the protocol. Then, for each role ρ, we have a finite family of *role state predicates* $\{\mathsf{A}_{\rho i}(\vec{m}) \mid i = 0, \ldots, l_\rho\}$. They are intended to hold the internal state of a principal in role ρ during the successive steps of the protocol.

This scheme can immediately be generalized to express roles that can take conditional or non-deterministic actions (*e.g.* toss a coin to choose among two messages to send — useful for zero-knowledge proofs for examples — or respond in two different ways depending on the contents of an incoming message — useful for intrusion detection). We simply need to alter our naming convention for role states and rules (below) to take alternatives into account.[2] This paper will consider only linearly ordered role states, as the layer of technicality required to treat the general case would obscure the comparison with strands.

Intruder knowledge: The unary predicate symbol I is needed to model the intruder's knowledge in a distributed fashion. It will be discussed at length in Section 2.4. This predicate shall not be accessible to honest principals.

We represent each role ρ in a protocol by means of a single *role generation rule* and a finite number of *protocol execution rules*. The purpose of the former is to prepare for the execution of an instance of role ρ. It has the form

$$r_{\rho 0} :\ \boldsymbol{\pi}(\vec{x}) \rightarrow \mathsf{A}_{\rho 0}(\vec{x}), \boldsymbol{\pi}(\vec{x}).$$

[2] Indeed, any partial ordering of the role state predicates will implement a *well-founded protocol theory*, as defined in [10, 16].

where, here and in the rest of the paper, $\boldsymbol{\pi}(\vec{x})$ denotes a multiset of persistent atomic formulas that may mention variables among $\vec{x}$. This portion of the antecedent will be matched against $\mathbf{\Pi}$, with the effect of instantiating $\vec{x}$ to actual persistent values such as principal names and keys. This implements a form of look-up. Notice how persistent information is preserved.

The execution rules describe the messages sent and expected by the principal acting in this role. For $i = 0, \ldots, l_\rho - 1$, we have a rule $r_{\rho i+1}$ of either of the following two forms:

Send: $$\begin{bmatrix}\mathsf{A}_{\rho i}(\vec{x}),\\ \boldsymbol{\pi}(\vec{x},\vec{z})\end{bmatrix} \quad\rightarrow\quad \exists\vec{n}.\quad \begin{bmatrix}\mathsf{A}_{\rho i+1}(\vec{x},\vec{z},\vec{n}),\\ \boldsymbol{\pi}(\vec{x},\vec{z}),\\ \mathsf{N}(m(\vec{x},\vec{z},\vec{n}))\end{bmatrix}$$

Receive: $$\begin{bmatrix}\mathsf{A}_{\rho i}(\vec{x}),\\ \boldsymbol{\pi}(\vec{x},\vec{y},\vec{z}),\\ \mathsf{N}(m(\vec{x},\vec{y}))\end{bmatrix} \quad\rightarrow\quad \begin{bmatrix}\mathsf{A}_{\rho i+1}(\vec{x},\vec{y},\vec{z}),\\ \boldsymbol{\pi}(\vec{x},\vec{y},\vec{z})\end{bmatrix}$$

where $m(\vec{v})$ stands for a message pattern with variables among $\vec{v}$. In the first type of rules, we rely on the existential operator $\exists\vec{n}$ to model the ability of a principal to create nonces when sending a message. This principal can also include some persistent data $\vec{z}$ (*e.g.* the name and public key of an interlocutor), possibly related to information it already possesses ($\vec{x}$). In the second rule template, the principal should be able to access persistent information $\vec{z}$ related to data $\vec{y}$ in the received message m (*e.g.* the sender's public key) or previously known information $\vec{x}$. Situations where a principal both sends and receives a message, or sends multiple messages, can easily be expressed by these rules.

A protocol is specified as a set $\mathcal{R}$ of such roles. Every $\mathcal{R}$ constructed in this way is trivially a well-founded protocol theory [10, 16]. As an example, Figure 1 shows the encoding of the familiar simplified Needham-Schroeder public key protocol in the multiset rewriting notation, according to the syntax defined in Appendix A. For the sake of readability, we omitted the keys in the persistent state predicates.

A *state* $\boldsymbol{S} = (\mathbf{\Pi}, \boldsymbol{A}, \boldsymbol{N}, \boldsymbol{I})$ is a multiset of ground facts, where $\mathbf{\Pi}$ is the persistent information, $\boldsymbol{A}$ is a multiset of role states $\mathsf{A}_{\rho i}(\vec{t})$, $\boldsymbol{N}$ is multiset of messages $\mathsf{N}(m)$ currently in transit, and $\boldsymbol{I}$ is a collection of predicates $\mathsf{I}(m)$ summarizing the intruder's knowledge. Notice in particular that the *initial state*, denoted $\boldsymbol{S}_0$, is just $(\mathbf{\Pi}, \boldsymbol{I}_0)$, where $\boldsymbol{I}_0$ contains the information (*e.g.* keys) initially known to the intruder.

2.3 Regular Protocol Theories

The notion of protocol theory introduced in the previous section, although weaker than our original definition [10, 16], is too liberal for a direct comparison with the strand formalism. We will instead rely on the more restrictive definition of *regular protocol theory*. The role generation rule of a regular role shall access all the persistent information that will be used in this role. It has therefore the following form:

$$r_{\rho 0}:\ \boldsymbol{\pi}(\vec{x}) \rightarrow \mathsf{A}_{\rho 0}(\vec{x}), \boldsymbol{\pi}(\vec{x}).$$

Consequently, protocol execution rules do not need to mention any persistent information:

Send: $\mathsf{A}_{\rho i}(\vec{x}) \quad\rightarrow\quad \exists\vec{n}.\quad \mathsf{A}_{\rho i+1}(\vec{x},\vec{n}),\quad \mathsf{N}(m(\vec{x},\vec{n}))$

Receive: $\mathsf{A}_{\rho i}(\vec{x}),\quad \mathsf{N}(m(\vec{x},\vec{y})) \quad\rightarrow\quad \mathsf{A}_{\rho i+1}(\vec{x},\vec{y})$

Regular protocol theories look up all the persistent information that is used in a role, including the identity and keys of other parties, before any message is exchanged. As we will see, this is closely related to the mode of executions of strands. The example in Figure 1, already discussed in the previous section, is indeed a regular protocol theory.

It should be observed that every protocol theory $\mathcal{R}$ can be transformed into a regular protocol theory $\widehat{\mathcal{R}}$ by simply moving all the persistent predicate occurring in a role to its role generation rule, and adding arguments to the role state predicates accordingly. In order to formalize this idea, we write $\hat{\boldsymbol{\pi}}(r)$ for the persistent predicates occurring in a rule r. We similarly write $\hat{\boldsymbol{\pi}}(\rho)$ for the persistent predicates appearing in role ρ; without loss of generality, we shall assume that variable names are used consistently in the different rules of a role: in particular every occurrence of state predicate symbol $\mathsf{A}_{\rho i}$ in ρ is always applied to the same string of variables. These notions are defined as follows:

$$
\begin{array}{lcll}
\hat{\boldsymbol{\pi}}(r_{\rho 0}) & = & \boldsymbol{\pi}(\vec{x}) & \text{where} \quad r_{\rho 0} = \boldsymbol{\pi}(\vec{x}) \rightarrow \mathsf{A}_{\rho 0}(\vec{x}), \boldsymbol{\pi}(\vec{x}) \\
\hat{\boldsymbol{\pi}}(r_{\rho i+1}) & = & \boldsymbol{\pi}(\vec{x}, \vec{z}) & \text{if} \quad r_{\rho i+1} = \\
 & & & \begin{bmatrix} \mathsf{A}_{\rho i}(\vec{x}), \\ \boldsymbol{\pi}(\vec{x}, \vec{z}) \end{bmatrix} \rightarrow \exists \vec{n}. \begin{bmatrix} \mathsf{A}_{\rho i+1}(\vec{x}, \vec{z}, \vec{n}), \\ \boldsymbol{\pi}(\vec{x}, \vec{z}), \\ \mathsf{N}(m(\vec{x}, \vec{z}, \vec{n})) \end{bmatrix} \\
\hat{\boldsymbol{\pi}}(r_{\rho i+1}) & = & \boldsymbol{\pi}(\vec{x}, \vec{y}, \vec{z}) & \text{if} \quad r_{\rho i+1} = \\
 & & & \begin{bmatrix} \mathsf{A}_{\rho i}(\vec{x}), \\ \boldsymbol{\pi}(\vec{x}, \vec{y}, \vec{z}), \\ \mathsf{N}(m(\vec{x}, \vec{y})) \end{bmatrix} \rightarrow \begin{bmatrix} \mathsf{A}_{\rho i+1}(\vec{x}, \vec{y}, \vec{z}), \\ \boldsymbol{\pi}(\vec{x}, \vec{y}, \vec{z}) \end{bmatrix} \\
\hat{\boldsymbol{\pi}}(\rho) & = & \hat{\boldsymbol{\pi}}(r_{\rho 0}), \ldots, \hat{\boldsymbol{\pi}}(r_{\rho n}) & \text{where} \quad \rho = r_{\rho 0}, \ldots, r_{\rho n}
\end{array}
$$

Then, the regular protocol theory $\hat{\mathcal{R}}$ (resp. role $\hat{\rho}$ and rule $\hat{r}$) corresponding to protocol theory $\mathcal{R}$ (resp. role ρ and rule r) is defined as follows:

$$
\begin{array}{lcll}
\hat{r}_{\rho 0} & = & \hat{\boldsymbol{\pi}}(\rho) \rightarrow \hat{\mathsf{A}}_{\rho 0}(\mathrm{Var}(\hat{\boldsymbol{\pi}}(\rho))), \hat{\boldsymbol{\pi}}(\rho) & \\
\hat{r}_{\rho i+1} & = & \hat{\mathsf{A}}_{\rho i}(\vec{x}\,') \rightarrow \exists \vec{n}. \hat{\mathsf{A}}_{\rho i+1}(\vec{x}\,', \vec{n}), \mathsf{N}(m(\vec{x}\,', \vec{n})) & (1) \\
\hat{r}_{\rho i+1} & = & \hat{\mathsf{A}}_{\rho i}(\vec{x}\,'), \mathsf{N}(m(\vec{x}\,', \vec{y})) \rightarrow \hat{\mathsf{A}}_{\rho i+1}(\vec{x}\,', \vec{y}) & (2) \\
\hat{\rho} & = & \hat{r}_{\rho 0}, \ldots, \hat{r}_{\rho n} & \\
\hat{\mathcal{R}} & = & \hat{\rho}_1, \ldots, \hat{\rho}_m &
\end{array}
$$

where

$$
\begin{array}{l}
r_{\rho 0} = \boldsymbol{\pi}(\vec{x}) \rightarrow \mathsf{A}_{\rho 0}(\vec{x}), \boldsymbol{\pi}(\vec{x}) \\
r_{\rho i+1} = \left\{ \begin{array}{ll} \mathsf{A}_{\rho i}(\vec{x}), \boldsymbol{\pi}(\ldots), \rightarrow \exists \vec{n}. \mathsf{A}_{\rho i+1}(\ldots), \boldsymbol{\pi}(\ldots), \mathsf{N}(m(\ldots)) & \text{in (1)} \\ \mathsf{A}_{\rho i}(\vec{x}), \boldsymbol{\pi}(\ldots), \mathsf{N}(m(\vec{x}, \vec{y})) \rightarrow \mathsf{A}_{\rho i+1}(\vec{x}, \vec{y}, \vec{z}), \boldsymbol{\pi}(\ldots) & \text{in (2)} \end{array} \right. \\
\rho = r_{\rho 0}, \ldots, r_{\rho n} \\
\mathcal{R} = \rho_1, \ldots, \rho_m
\end{array}
$$

Here, $\hat{\mathsf{A}}_{\rho i}$ is the role state predicate corresponding to $\mathsf{A}_{\rho i}$. The definition of rule translation describes how to compute its arguments: the first of these predicates, $\hat{\mathsf{A}}_{\rho 0}$ is equipped with all the variables occurring in $\hat{\boldsymbol{\pi}}(\rho)$, a set that we have written $\mathrm{Var}(\hat{\boldsymbol{\pi}}(\rho))$ above. The choice of argument of the remaining role state predicates, $\hat{\mathsf{A}}_{\rho i+1}$ is guided by the form of the rule in which the corresponding $\mathsf{A}_{\rho i+1}$ first occurs. Therefore, the variables $\vec{x}\,'$ in the definitions of $\hat{r}_{\rho i+1}$ are the arguments computed for $\hat{\mathsf{A}}_{\rho i+1}$ in the consequent of rule $\hat{r}_{\rho i}$.

Example: As a simple example, consider the three-rule role at the top of the following table, in which an initiator A sends a newly generated nonce N_A on the network, and expects it back encrypted with her own

private key K_A.

$r_{\rho 0} : \mathit{Pr}(A)$	$\rightarrow$	$\mathsf{A}_{\rho 0}(A), \mathit{Pr}(A)$
$r_{\rho 1} : \mathsf{A}_{\rho 0}(A)$	$\rightarrow$	$\exists N_A.\, \mathsf{A}_{\rho 1}(A, N_A), \mathsf{N}(N_A)$
$r_{\rho 2} : \begin{bmatrix} \mathsf{A}_{\rho 1}(A, N_A), \\ \mathit{PubK}(A, K_A), \\ \mathsf{N}(\{N_A\}_{K_A}) \end{bmatrix}$	$\rightarrow$	$\begin{bmatrix} \mathsf{A}_{\rho 2}(A, N_A, K_A), \\ \mathit{PubK}(A, K_A) \end{bmatrix}$
$\hat{r}_{\rho 0} : \begin{bmatrix} \mathit{Pr}(A), \\ \mathit{PubK}(A, K_A) \end{bmatrix}$	$\rightarrow$	$\begin{bmatrix} \hat{\mathsf{A}}_{\rho 0}(A, K_A), \mathit{Pr}(A), \\ \mathit{PubK}(A, K_A) \end{bmatrix}$
$\hat{r}_{\rho 1} : \hat{\mathsf{A}}_{\rho 0}(A, K_A)$	$\rightarrow$	$\exists N_A.\, \hat{\mathsf{A}}_{\rho 1}(A, K_A, N_A), \mathsf{N}(N_A)$
$\hat{r}_{\rho 2} : \begin{bmatrix} \hat{\mathsf{A}}_{\rho 1}(A, K_A, N_A), \\ \mathsf{N}(\{N_A\}_{K_A}) \end{bmatrix}$	$\rightarrow$	$\hat{\mathsf{A}}_{\rho 2}(A, K_A, N_A)$

The corresponding regular role is displayed at the bottom of the table. Observe that all the persistent information is gathered in the role generation rule $\hat{r}_{\rho 0}$ and no persistent predicates appears in any other rule. Observe also how the arguments of the role state predicates have been updated.

The transformation we have just outlined does not preserve execution. This can clearly be seen in the above example: assume that the specification contains a principals a who does not have a public key. Then, rule $r_{\rho 0}$ (and successively rule $r_{\rho 1}$) can fire in any state $\boldsymbol{S}$, resulting in the state $(\boldsymbol{S}, \mathsf{A}_{\rho 0}(a))$. Rule $r_{\rho 2}$ cannot execute since there is no $\mathit{PubK}(a, k)$ in $\mathbf{\Pi}$ for any k. However, rule $\hat{r}_{\rho 0}$ is not enabled in any reasonable translation $\hat{\boldsymbol{S}}$ of $\boldsymbol{S}$ since there is no k_a such that $\mathit{PubK}(a, k_a)$ holds.

However, any execution sequence in the transformed system can be mapped back to an application of the corresponding rules in the original system:

Property 2.1 *If $\boldsymbol{S}_0 \longrightarrow^*_{\hat{\mathcal{R}}} \boldsymbol{S}$, then $\boldsymbol{S}_0 \longrightarrow^*_{\mathcal{R}} \boldsymbol{S}$.*

Proof: The proof proceeds by induction on the length of the transformed execution sequence, say $\vec{\hat{r}}, \vec{\hat{\delta}}$, where $\vec{\hat{\delta}}$ denotes the instantiating substitutions used in it. □

Observe that this property does not hold, in general, if we start from a state that includes intermediate role state predicates as we cannot guarantee that their arguments are related to $\mathbf{\Pi}$. This is the reason we consider only the initial state $\boldsymbol{S}_0$.

Regular protocol theories upgrade our original definition of (unqualified) protocol theories [9] with the requirement that all the persistent information used during the execution of a role be accessed in its role generation rule. While the two definitions are equally acceptable in general, the regularity restriction brings us one step closer to the strand world, where all accessory values are chosen up-front. This is a slippery slope since, as we just saw, protocol theories cannot be regularized in general without losing transition sequences (see Section 4.1 for the consequences of starting from generic protocol theories). This is one more restriction that our multiset rewriting formalism shall abide by in order to set up a fair comparison with strand spaces.

From now on, all the protocol theories we will be considering shall be regular.

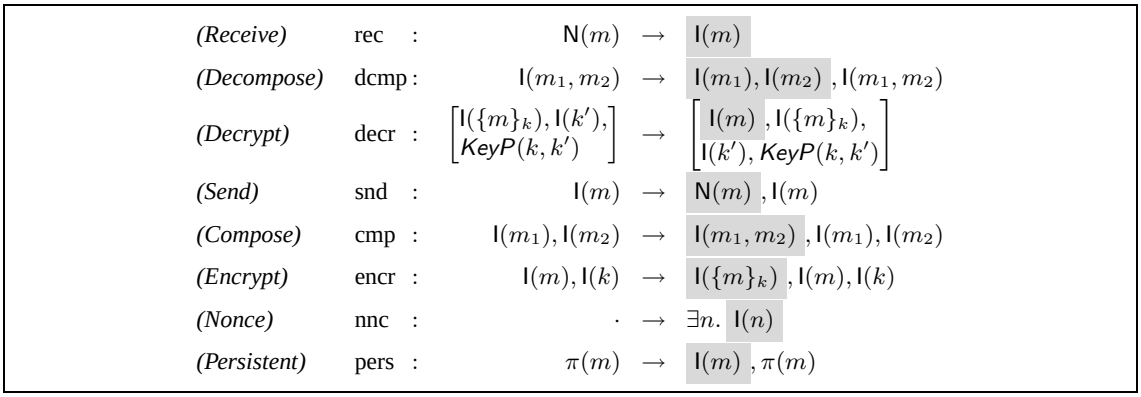


Figure 2. The Standard Intruder Theory $\mathcal{I}$

2.4 Intruder Theory

The knowledge available at any instant to the intruder consists of the persistent information in $\mathbf{\Pi}$, of the unused portion of its initial knowledge $\boldsymbol{I}_0$ (*e.g.* the keys of dishonest principals), and of intercepted or inferred messages. We use the state predicate $\mathsf{I}(\_)$ to hold each piece of information known to the intruder. In particular, we represent the fact that the intruder "knows" m (a message, a key, etc.) as $\mathsf{I}(m)$. The overall knowledge of the intruder at any particular instant is indicated with $\boldsymbol{I}$. As mentioned above, we write $\boldsymbol{I}_0$ for the intruder's initial knowledge.

The capabilities of the intruder are modeled by the *standard intruder theory* $\mathcal{I}$ displayed in Figure 2. These rules are taken from [10, 16]. The standard intruder theory $\mathcal{I}$ implements the Dolev-Yao model [15, 28] in our notation. For the sake of readability, we have grayed out the information produced by each rule. Observe that these rules display an overly conservative bookkeeping strategy for the known messages: knowledge is never discarded, but carried along as new messages are inferred.

The intruder capabilities formalized in the strand model relies on a slightly different strategy for managing captured knowledge: inferring new information has the effect of deleting the data it was constructed from. Moreover, it can discard information. However, explicit duplication is possible. We express this behavior by the set of rules $\mathcal{I}'$ in Figure 3.

Clearly, our original intruder model $\mathcal{I}$ can easily be simulated by a systematic use of the duplication rule of $\mathcal{I}'$. Going in the other direction is slightly more complicated as $\mathcal{I}$ never discards any information. The substantial equivalence of these two systems is summarized in the following result.

Property 2.2 *Let $\mathcal{R}$ be an arbitrary protocol theory, and $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$ two states.*

- *For every rule sequence $\vec{r}$ in $\mathcal{R}, \mathcal{I}$ such that $\boldsymbol{S}_1 \xrightarrow{\vec{r}}{}^*_{\mathcal{R},\mathcal{I}} \boldsymbol{S}_2$, there exists a rule sequence $\vec{r}\,'$ in $\mathcal{R}, \mathcal{I}'$ such that $\boldsymbol{S}_1 \xrightarrow{\vec{r}\,'}{}^*_{\mathcal{R},\mathcal{I}'} \boldsymbol{S}_2$.*
- *For every rule sequence $\vec{r}\,'$ in $\mathcal{R}, \mathcal{I}'$ such that $\boldsymbol{S}_1 \xrightarrow{\vec{r}\,'}{}^*_{\mathcal{R},\mathcal{I}'} \boldsymbol{S}_2$, there exist a rule sequence $\vec{r}$ in $\mathcal{R}, \mathcal{I}$ and an intruder state $\boldsymbol{I}'$ such that $\boldsymbol{S}_1 \xrightarrow{\vec{r}}{}^*_{\mathcal{R},\mathcal{I}} \boldsymbol{S}_2, \boldsymbol{I}'$.*

Proof: The idea underlying the proof of the first statement is that every rule in $\mathcal{I}$ can be emulated by the corresponding rule in $\mathcal{I}'$ preceded by one or more applications of dup. Rule del is never used. The transition sequence $\vec{r}\,'$ is derived from $\vec{r}$ according to this strategy. A formal proof proceeds by induction on $\vec{r}$.

The proof of the second half of this property is based on the observation that rule dup can be emulated in $\mathcal{I}$ by applying snd and rec in succession. Rules rec'–pers' are mapped to their unprimed name sakes in

(Receive)	rec'	:	$\mathsf{N}(m)$	$\rightarrow$	$\mathsf{I}(m)$
(Decompose)	dcmp'	:	$\mathsf{I}(m_1, m_2)$	$\rightarrow$	$\mathsf{I}(m_1), \mathsf{I}(m_2)$
(Decrypt)	decr'	:	$\mathsf{I}(\{m\}_k), \mathsf{I}(k'), \mathsf{KeyP}(k, k')$	$\rightarrow$	$\mathsf{I}(m), \mathsf{KeyP}(k, k')$
(Send)	snd'	:	$\mathsf{I}(m)$	$\rightarrow$	$\mathsf{N}(m)$
(Compose)	cmp'	:	$\mathsf{I}(m_1), \mathsf{I}(m_2)$	$\rightarrow$	$\mathsf{I}(m_1, m_2)$
(Encrypt)	encr'	:	$\mathsf{I}(m), \mathsf{I}(k)$	$\rightarrow$	$\mathsf{I}(\{m\}_k)$
(Nonce)	nnc'	:	$\cdot$	$\rightarrow$	$\exists n.\, \mathsf{I}(n)$
(Persistent)	pers'	:	$\pi(m)$	$\rightarrow$	$\mathsf{I}(m), \pi(m)$
(Duplicate)	dup	:	$\mathsf{I}(m)$	$\rightarrow$	$\mathsf{I}(m), \mathsf{I}(m)$
(Delete)	del	:	$\mathsf{I}(m)$	$\rightarrow$	$\cdot$

Figure 3. The Modified Intruder Theory $\mathcal{I}'$

$\mathcal{I}$, which has the effect of retaining copies of intermediate intruder information. Rule del is discarded. The extra intruder knowledge predicates resulting from these two situations are collected in the intruder state fragment $\boldsymbol{I}'$. Again, this is formally proved by induction on $\vec{r}\,'$. □

3 Strand Constructions

This section introduces strands spaces and the dynamic extension that will be considered throughout this paper. We start with some basic definitions from graph theory (Section 3.1). We then introduce the strands and related concepts as a graphical language to describe protocol executions (Section 3.2). It is then upgraded with a syntax for protocols as first-class objects and a dynamic semantics that emulates step-wise execution (Section 3.3). These extensions are of independent interest and some of their properties will be further analyzed in Appendix B. We conclude with a presentation of penetrator strands as the intruder model of strand spaces (Section 3.4).

3.1 Preliminary Definitions

A *directed graph* G is a pair $(S, \longrightarrow)$ where S is the set of *nodes* of G and $\longrightarrow \subseteq S \times S$ is the set of *edges* of G. We will generally write $\nu_1 \longrightarrow \nu_2$ for $(\nu_1, \nu_2) \in \longrightarrow$. A *directed labeled graph* G_L is a structure $(S, \longrightarrow, L, \Lambda)$ where $(S, \longrightarrow)$ is a directed graph, L is a set of *labels*, and $\Lambda : S \rightarrow L$ is a *labeling function* that associates a label to every node. In the sequel, all our graphs will be directed and labeled, but we will generally keep Λ implicit for simplicity. In particular, for $\nu \in S$ and $l \in L$, we will write "$\nu = l$" as an abbreviation of $\Lambda(\nu) = l$. However, for $\nu_1, \nu_2 \in S$, expressions of the form "$\nu_1 = \nu_2$" shall always refer to the nodes themselves, and not to their labels.

A graph $G = (S, \longrightarrow)$ is a *chain* if there is a total ordering $\nu_0, \nu_1, \ldots$ of the elements of S such that $\nu_i \longrightarrow \nu_j$ iff $j = i + 1$. A graph $G = (S, \longrightarrow)$ is a *disjoint union of chains* if $S = \bigcup_{i \in I} S_i$ and $\longrightarrow = \bigcup_{i \in I} \longrightarrow_i$ (for some set I) and $(S_i, \longrightarrow_i)$ are chains for each $i \in I$.

A *bipartite graph* is a structure $G = (S_1, S_2, \longrightarrow)$ such that S_1 and S_2 are disjoint, $(S_1 \cup S_2, \longrightarrow)$ is a graph, and if $\nu_1 \longrightarrow \nu_2$ then $\nu_1 \in S_1$ and $\nu_2 \in S_2$. Observe that all edges go from S_1 to S_2 (*i.e.* $\longrightarrow \subseteq S_1 \times S_2$).

We say that $G = (S_1, S_2, \longrightarrow)$ is

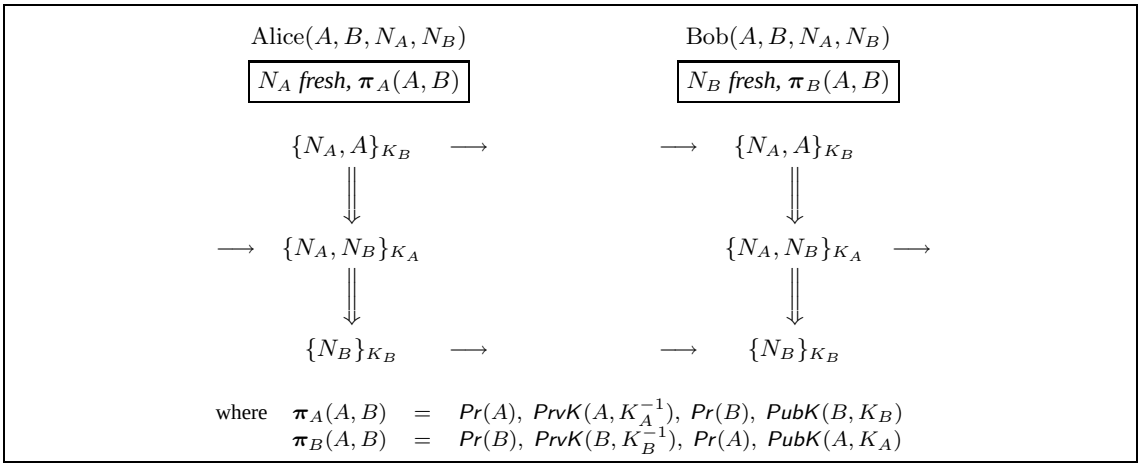


Figure 4. Parametric Strand Specification of the Needham-Schroeder Protocol

- *functional* if $\longrightarrow$ is a partial function (*i.e.* if $\nu \longrightarrow \nu'_1$ and $\nu \longrightarrow \nu'_2$ imply $\nu'_1 = \nu'_2$).
- *injective* if $\longrightarrow$ is injective (*i.e.* if $\nu_1 \longrightarrow \nu'$ and $\nu_2 \longrightarrow \nu'$ imply $\nu_1 = \nu_2$).
- *surjective* if $\longrightarrow$ is surjective onto S_2 (*i.e.* for each $\nu' \in S_2$ there is $\nu \in S_1$ such that $\nu \longrightarrow \nu'$).

A *bi-graph* G is a structure $(S, \Longrightarrow, \longrightarrow)$ where both $(S, \Longrightarrow)$ and $(S, \longrightarrow)$ are graphs.

In the sequel, we will often rely on the natural adaptation of standard graph-theoretic notions (*e.g.* isomorphism) to labeled graphs and bi-graphs.

3.2 Strands and Bundles

An *event* is a pair consisting of a message m and an indication of whether it has been sent ($+m$) or received ($-m$) [33]. The set of all events will be denoted $\pm\mathcal{M}$.

A *strand* is a finite sequence of events, *i.e.* an element of $(\pm\mathcal{M})^*$. We indicate strands with the letter s, the length of a strand as $|s|$, and its i-th event as s_i (for $i = 1, \ldots, |s|$). Observe that a strand s can be thought of as a chain graph $(S, \Longrightarrow)$ with labels over $\pm\mathcal{M}$, where $S = \{s_i : i = 1, \ldots, |s|\}$ and $s_i \Longrightarrow s_j$ iff $j = i + 1$.

Slightly simplifying from [33], a *strand space* is a set of strands with an additional relation ($\longrightarrow$) on the nodes. The only condition is that if $\nu_1 \longrightarrow \nu_2$, then $\nu_1 = +m$ and $\nu_2 = -m$ (for the same message m). Therefore, $\longrightarrow$ represents the transmission of the message m from the sender ν_1 to the receiver ν_2. Alternatively, a strand space can be viewed as a labeled bi-graph $\sigma = (S, \Longrightarrow, \longrightarrow)$ with labels over $\pm\mathcal{M}$, $\Longrightarrow \subseteq S \times S$, and $\longrightarrow \subseteq S^+ \times S^-$ where S^+ and S^- indicate the set of positively- and negatively-labeled nodes in S respectively, and the constraints discussed above: $(S, \Longrightarrow)$ is a disjoint union of chains, and if $\nu_1 \longrightarrow \nu_2$, then $\nu_1 = +m$ and $\nu_2 = -m$ for some message m.

A *bundle* is a strand space $\sigma = (S, \Longrightarrow, \longrightarrow)$ such that the bipartite graph $(S^+, S^-, \longrightarrow)$ is injective, and surjective, and the graph obtained by dropping the distinction between $\Longrightarrow$ and $\longrightarrow$ is acyclic. In terms of protocols, the first two constraints imply that no message is received from more than one sender, and every received message has been sent, respectively. Dangling positive nodes correspond to messages in transit. Slightly departing from [33, 31], it will be convenient to restrict our attention to functional bundles, which corresponds to adding the further constraint that a message is sent to at most one recipient at a time.

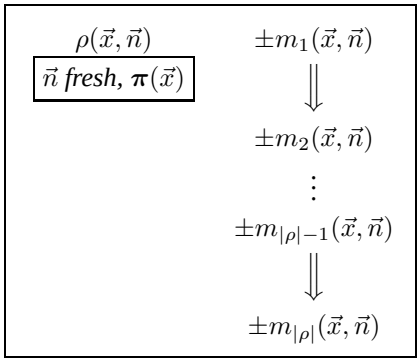


Figure 5. A Parametric Strand

An arbitrary bundle can be easily made functional by inserting T strands from the standard intruder toolkit, developed in Section 3.4.

If we think in terms of protocols, a bundle represents a snapshot of the execution of a protocol. As we will see in Sections 3.3 and also in Appendix B, this comprises a current global state (what each principal and the intruder are up to, and the messages in transit), as well as a precise account of how this situation has been reached. Each role is expressed as a strand in the current bundle. The intruder capabilities are themselves modeled as a fixed set of *penetrator strands*, which can be woven in a bundle. We postpone the exact definition until Section 3.4 as the construction we propose in the next sections will generalize the presentation in [33, 31].

3.3 Extensions

We now refine these concepts with a language to describe protocols as first-class objects and a semantics to grow bundles dynamically. Following [9], similar extensions have become popular in the strand literature, *e.g.* [11, 17, 19, 26].

The notion of role is kept implicit in [33] and introduced as the concept of *trace-type* in [31]. A *role* is nothing but a parametric strand: a strand where the messages may contain variables. An actual strand is obtained by instantiating all the variables in a parametric strand (or an initial segment of one) with persistent information and actual message pieces. For simplicity, we will not define nor consider constructions corresponding to arbitrary well-founded protocol theories (see Section 2 and [10, 16]).

A *parametric strand* for the role ρ may look as in Figure 5. The freshness of $\vec{n}$, *i.e.* the fact that the variables $\vec{n}$ should be instantiated with "new" constants that have not been used before, is expressed as a side condition. Using the terminology in [33, 31], the values $\vec{n}$ are *uniquely originated*. This descriptive notion is sufficient to characterize fresh information in a stopped execution. In a parametric strand, '$\vec{n}\ fresh$' (like $\exists\vec{n}$ in the previous section) has instead prescriptive strength as it shall enforce freshness as the execution unfolds rather than just acknowledge it. Therefore, '$fresh$' is an operator in our specification calculus while unique origination only needed to be a meta-level property in [33, 31]. The relationships between variables are expressed in [31] using intuitive notation, *e.g.* k^{-1} for the inverse key of k, or k_A for the key of A. We formalize these relations by equipping ρ with the constraints $\boldsymbol{\pi}(\vec{x})$, that, without loss of generality, will be a set of persistent atomic formulas from Section 2, parameterized over $\vec{x}$.

As in the case of transition systems, a *protocol* is given as a set of roles. The model of the intruder in the style of Dolev and Yao [15, 28] is also specified as a set of parametric strands $\mathcal{P}(P_0)$ called *penetrator strands*, where P_0 is the intruder's initial knowledge (see Section 3.4 or [31] for a definition). As an

example, Figure 4 shows how the Needham-Schroeder public key protocol is modeled using parametric strands, where we have used incoming and outgoing arrows instead of the tags $+$ and $-$ for readability.

As in Section 2.1, a *substitution* is a finite tuple $\delta = (t_1/x_1, \ldots, t_n/x_n)$ of term-variable pairs t_i/x_i. The domain of δ is $\mathrm{dom}(\delta) = (x_1, \ldots, x_n)$, with each x_i distinct. All our substitution will be *ground*, by which we mean that none of the t_i's will contain any variable. We will rely on two types of substitutions: substitutions that replace variables with distinct fresh constants that have not been previously encountered, and substitutions that map variables to previously used ground terms (not necessarily constants). We will use the letters ξ and θ, possibly subscripted, to denote them respectively. We will use δ for substitutions that mix these two components. Given a parametric message m with variables in $\mathrm{dom}(\delta)$, we denote the application of δ to m as $m[\delta]$. Given substitutions $\delta_1, \ldots, \delta_n$, we write $m[\delta_1 \cdots \delta_n]$ for $(\ldots(m[\delta_1])\ldots)[\delta_n]$. We extend this notation to nodes, writing $\nu[\delta]$ and to (possibly partially instantiated) parametric strands, with the notation $\rho[\delta]$.

These definitions allow us to specialize the bundles we will be looking at: given a set of parametric strands $\mathcal{S}$, every strand in a bundle σ should be a fully instantiated initial prefix of a protocol (or penetrator) strand. We are interested in initial prefixes since a bundle is a snapshot of the execution of a protocol, and a particular role instance may be halfway through its execution. We then say that σ is a *bundle over* $\mathcal{S}$. We need to generalize strands constructions to admit strand spaces containing partially instantiated parametric strands. We call them *parametric strand spaces*. The bundles we will consider will however always be ground.

We will now give a few definitions needed to emulate the execution of a protocol with parametric strands. No such definitions can be found in the original description of strand constructions [33, 31], which focuses on analyzing protocol traces, not on specifying how to generate them.

First, observe that the network traffic in a bundle is expressed in terms of events and of the $\longrightarrow$ relation. The edges of $\longrightarrow$ represent past traffic: messages that have been sent and successfully received. The dangling positive nodes correspond to current traffic: messages in transit that have been sent, but not yet received. We will call these nodes the *fringe* of the bundle (or strand space). More formally, given a strand space $\sigma = (S, \Longrightarrow, \longrightarrow)$, its fringe is the set

$$\mathrm{Fr}(\sigma) = \{\nu : \nu \in S,\ \nu = +m,\ \text{and } \nexists \nu'.\, \nu \longrightarrow \nu'\}$$

Another component of the execution state of a protocol is a description of the actions that can legally take places in order to continue the execution. First, some technicalities. Let σ be a bundle over a set of parametric strands $\mathcal{S}$, a *completion* of σ is any strand space $\tilde{\sigma}$ that embeds σ as a subgraph, and that extends each incomplete strand in it with the omitted nodes and the relative $\Longrightarrow$-edges. A completion of σ may contain additional strands, possibly only partially instantiated. If s is a strand in σ and $\tilde{s}$ is its extension in $\tilde{\sigma}$, the sequence obtained by removing every event in s from $\tilde{s}$ is itself a (possibly empty) strand. We call it a *residual strand* and indicate it as $\tilde{s} \setminus s$. We then write $\tilde{\sigma} \setminus \sigma$ for the set of all residual strands of $\tilde{\sigma}$ with respect to σ, plus any strands that $\tilde{\sigma}$ may contain in addition to those in σ.

Figure 6 illustrates these concepts on a standard run of the Needham-Schroeder protocol. Role names and variable instantiations are given in the header. Ignoring for a moment the lower horizontal $\longrightarrow$-edge, the grayed-out portion of this figure shows a bundle σ representing an initial segment of the execution of this protocol. The strand space $\tilde{\sigma}$ in the overall figure is a possible completion of σ with respect to the parametric strands given in Figure 4. The set of residual strands $\tilde{\sigma} \setminus \sigma$ is given by the white portion of this figure.

Given these preliminary definitions, a *configuration* over $\mathcal{S}$ is a pair of strand spaces $(\sigma, \sigma^\sharp)$ where σ is a bundle over $\mathcal{S}$, and $\sigma^\sharp$ is a completion of σ whose only additional $\longrightarrow$-edges originate in $\mathrm{Fr}(\sigma)$, cover all of $\mathrm{Fr}(\sigma)$, and point to $\sigma^\sharp \setminus \sigma$. Clearly, if $\sigma = (S, \Longrightarrow, \longrightarrow)$ and $\sigma^\sharp = (S^\sharp, \Longrightarrow^\sharp, \longrightarrow^\sharp)$, we have that

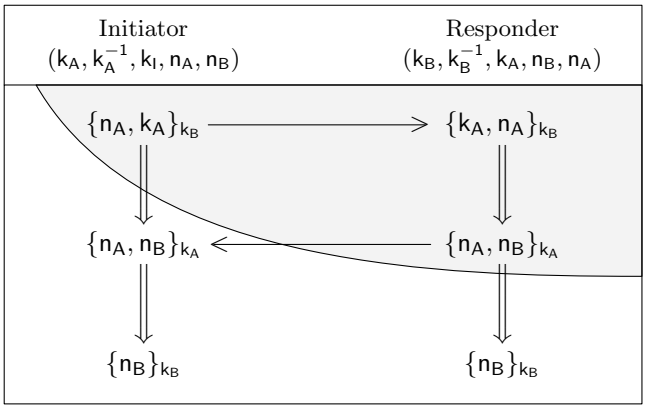


Figure 6. A Configuration for the Needham-Schroeder Protocol

$S \subseteq S^\sharp$, and $\Longrightarrow \subseteq \Longrightarrow^\sharp$, and finally $\longrightarrow \subseteq \longrightarrow^\sharp$. The above figure represents a configuration for the Needham-Schroeder protocol. We will rely on this intuitive format as a diagrammatic abstraction in the remaining of this paper.

A *one-step transition* is what it takes to go from one bundle to the "next". Since a bundle only keeps track of events that have taken place on each strand, but does not have any record of the remaining events on that strand, we shall define this relation over configurations. There are two ways to make progress in the bundle world: extend an existing strand, or add a new one. Let us analyze them:

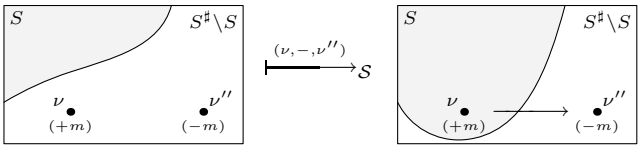
- *Extending a strand*: If the configuration at hand embeds a strand that is not fully contained in its bundle part, then we add the first missing node of the latter and the incoming $\Longrightarrow$-edge. If this node is positive, we add an $\longrightarrow^\sharp$-arrow to a matching negative node devoid of any incoming $\longrightarrow$-arrow. If it is negative, we must make sure that it has an incoming $\longrightarrow^\sharp$-edge.
- *Creating a strand*: Alternatively, we can select a parametric strand and instantiate first its "fresh" variables and then its other parameters. The first operation replaces the "fresh" variables with new values prescriptively enforcing freshness, while the second relies solely on existing values. Combining these two instantiations into a single operation would either imply abandoning the prescriptive power of "fresh" (which would not serve our purposes), or would lead to a logical deadlock, in general. Take for example the above configuration for the Needham-Schroeder protocol: the initiator cannot instantiate the parameter n_B until the responder has created a nonce $\mathsf{n_B}$ for it and dually for n_A. The execution is possible only if the instantiation of the fresh variable of each role precedes the instantiation of the other variables.

We will now formalize this notion. Let $(\sigma_1, \sigma_1^\sharp)$ and $(\sigma_2, \sigma_2^\sharp)$ be configurations over a set of parametric strands $\mathcal{S}$, with $\sigma_i = (S_i, \Longrightarrow_i, \longrightarrow_i)$ and $\sigma_i^\sharp = (S_i^\sharp, \Longrightarrow_i^\sharp, \longrightarrow_i^\sharp)$, for $i = 1, 2$. We say that $(\sigma_2, \sigma_2^\sharp)$ *immediately follows* $(\sigma_1, \sigma_1^\sharp)$ by means of move o, written $(\sigma_1, \sigma_1^\sharp) \stackrel{o}{\longmapsto}_{\mathcal{S}} (\sigma_2, \sigma_2^\sharp)$, if any of the following situations apply. An intuitive sense of what each case formalizes can be gained by looking at the pictorial abstraction to the right of each possibility. Here, ν, ν' and ν'' stand for nodes on fully instantiated strands, while ν_0 will generally be only partially instantiated.

$\mathbf{S_0}$ — Initial Send: There are nodes $\nu, \nu'' \in S_1^\sharp \setminus S_1$ such that $\nu = +m$, $\nu'' = -m$, no $\longrightarrow$-edge enters

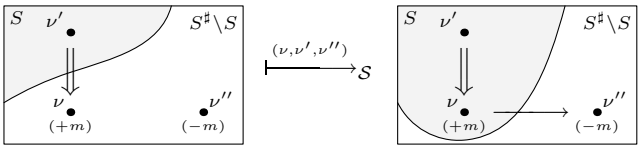
ν'', and no $\Longrightarrow$-arrow enters ν. Then,

$$S_2 = S_1 \cup \{\nu\}, \quad \Longrightarrow_2 = \Longrightarrow_1, \quad \longrightarrow_2 = \longrightarrow_1;$$
$$S_2^\sharp = S_1^\sharp, \quad \Longrightarrow_2^\sharp = \Longrightarrow_1^\sharp, \quad \longrightarrow_2^\sharp = \longrightarrow_1^\sharp \cup \{(\nu, \nu'')\}.$$



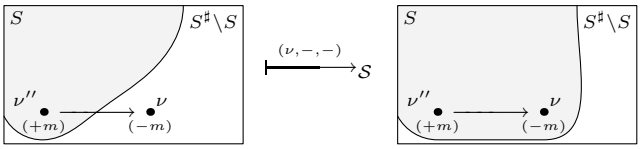
S — Successive Send: There are nodes $\nu, \nu'' \in S_1^\sharp \setminus S_1$ and $\nu' \in S_1$ such that $\nu = +m$, $\nu'' = -m$, no $\longrightarrow$-edge enters ν'', and $\nu' \Longrightarrow_1^\sharp \nu$. Then,

$$S_2 = S_1 \cup \{\nu\}, \quad \Longrightarrow_2 = \Longrightarrow_1 \cup \{(\nu', \nu)\}, \quad \longrightarrow_2 = \longrightarrow_1;$$
$$S_2^\sharp = S_1^\sharp, \quad \Longrightarrow_2^\sharp = \Longrightarrow_1^\sharp, \quad \longrightarrow_2^\sharp = \longrightarrow_1^\sharp \cup \{(\nu, \nu'')\}.$$



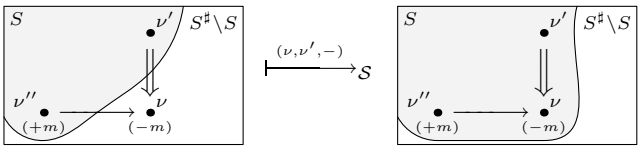
$\mathbf{R_0}$ — Initial Receive: There are nodes $\nu \in S_1^\sharp \setminus S_1$ and $\nu'' \in S_1$ such that $\nu = -m$, $\nu'' = +m$, $\nu'' \longrightarrow_1^\sharp \nu$, and no $\Longrightarrow$ enters ν. Then,

$$S_2 = S_1 \cup \{\nu\}, \quad \Longrightarrow_2 = \Longrightarrow_1, \quad \longrightarrow_2 = \longrightarrow_1 \cup \{(\nu'', \nu)\};$$
$$\sigma_2^\sharp = \sigma_1^\sharp.$$



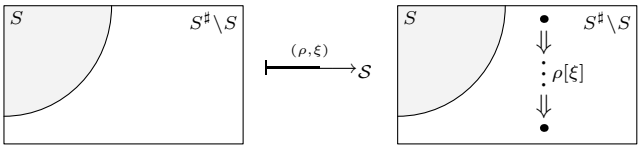
R — Successive Receive: There are nodes $\nu \in S_1^\sharp \setminus S_1$ and $\nu', \nu'' \in S_1$ such that $\nu = -m$, $\nu'' = +m$, $\nu'' \longrightarrow_1^\sharp \nu$, and $\nu' \Longrightarrow_1^\sharp \nu$. Then,

$$S_2 = S_1 \cup \{\nu\}, \quad \Longrightarrow_2 = \Longrightarrow_1 \cup \{(\nu', \nu)\}, \quad \longrightarrow_2 = \longrightarrow_1 \cup \{(\nu'', \nu)\};$$
$$\sigma_2^\sharp = \sigma_1^\sharp.$$



$\mathbf{C_f}$ — Fresh Variable Instantiation: ρ is a parametric strand in $\mathcal{S}$ and ξ is a substitution for all its variables marked "fresh" with constants that appear nowhere in $(\sigma_1, \sigma_1^\sharp)$.

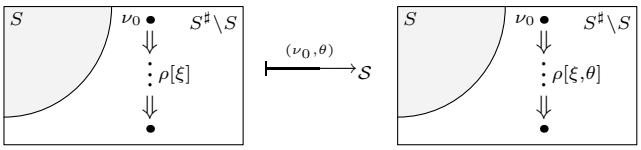
$$\sigma_2 = \sigma_1; \qquad \sigma_2^\sharp = \sigma_1^\sharp \cup \rho[\xi].$$



where, $\sigma \cup s$ is obtained by taking the union of the nodes and $\Longrightarrow$-edges of σ and s,

$\mathbf{C_i}$ — Other Variables Instantiation: $\rho[\xi]$ is a partially instantiated parametric strand in $\sigma_1^\sharp$ and θ is a ground substitution for the remaining variables. In particular, if $\rho[\xi]$ mentions constraints $\boldsymbol{\pi}$, then their instantiation should be compatible with the know persistent data, *i.e.* $\boldsymbol{\pi}[\theta] \subseteq \Pi$. Then,

$$\sigma_2 = \sigma_1; \qquad \sigma_2^\sharp = (\sigma_1^\sharp - \rho[\xi]) \cup \rho[\xi, \theta].$$



where, $\sigma - s$ is the subgraph of σ obtained by removing all nodes of s and their incident edges.

The *move* o that labels the transition arrow $\longmapsto_{\mathcal{S}}$ records the necessary information to reconstruct the transition uniquely. Given a configuration $(\sigma, \sigma^\sharp)$, a *move* for transitions of type $\mathbf{S_0}$, $\mathbf{S}$, $\mathbf{R_0}$, and $\mathbf{R}$ is a triple $o = (\nu, \bar{\nu}^p, \bar{\nu}^s)$ where ν is a node, $\bar{\nu}^p$ is the parent node ν^p of ν according to the $\Longrightarrow$ relation (or "–" if ν is the first node of a chain — cases $\mathbf{S_0}$ and $\mathbf{R_0}$), and $\bar{\nu}^s$ is the recipient ν^s of the message that labels ν along the $\longrightarrow$ relation (if ν is positive, or "–" otherwise). For transitions of type $\mathbf{C_f}$ and $\mathbf{C_i}$, moves have the form (ρ, ξ) and (ν_0, θ) respectively, where ρ is the name of the chosen parametric strand, ν_0 is the first node of the partially instantiated strand $\rho[\xi]$, and ξ and θ are the instantiating substitutions.

A *multistep transition* amounts to chaining zero or more one-step transitions. This relation is obtained by taking the reflexive and transitive closure $\stackrel{\vec{o}}{\longmapsto}{}^*_{\mathcal{S}}$ of $\stackrel{o}{\longmapsto}_{\mathcal{S}}$, where $\vec{o}$ is the sequence of the component moves ("·" if empty). $\vec{o}$ is a trace of the computation.

Observe that our definition of transition preserves configurations, *i.e.* if $(\sigma_1, \sigma_1^\sharp)$ is a configuration and $(\sigma_1, \sigma_1^\sharp) \stackrel{o}{\longmapsto}_{\mathcal{S}} (\sigma_2, \sigma_2^\sharp)$, then $(\sigma_2, \sigma_2^\sharp)$ is also a configuration. This property clearly extends to multistep transitions.

Property 3.1 *Let $(\sigma_1, \sigma_1^\sharp)$ be a configuration.*

1. *If $(\sigma_1, \sigma_1^\sharp) \stackrel{o}{\longmapsto}_{\mathcal{S}} (\sigma_2, \sigma_2^\sharp)$, then $(\sigma_2, \sigma_2^\sharp)$ is a configuration.*
2. *If $(\sigma_1, \sigma_1^\sharp) \stackrel{\vec{o}}{\longmapsto}{}^*_{\mathcal{S}} (\sigma_2, \sigma_2^\sharp)$, then $(\sigma_2, \sigma_2^\sharp)$ is a configuration.*

Proof: By inspection, it is easy to ascertain that each form of transition produces a configuration when applied to a configuration. The second part of this lemma is proved by induction on the length of $\vec{o}$. □

An analysis of the notions just defined can be found in Appendix B.

3.4 Penetrator Strands

We now formalize the intruder model of [33, 31], which consists of patterns called *penetrator strands*, and of a set of messages P_0 expressing the intruder's initial knowledge. The corresponding parametric

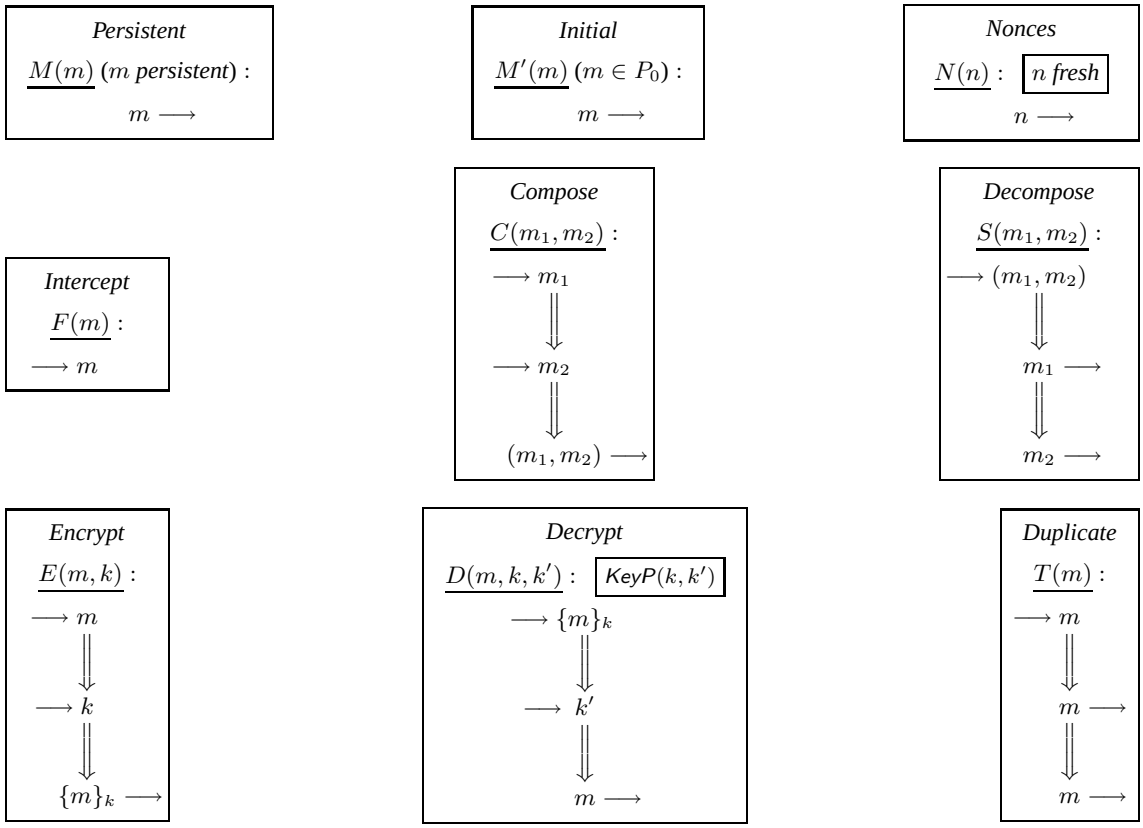


Figure 7. The Penetrator Strands $\mathcal{P}$

strands are shown in Figure 7, which includes a case to handle intruder-generated nonces. Since freshness is a descriptive property in [33, 31], this possibility is handled by the M penetrator strand. For convenience of comparison with the multiset model, we also distinguished cases $M(m)$ and $M'(m)$, which are identified in [33, 31]. We refer to the collection of (parametric) penetrator strands in Figure 7 as $\mathcal{P}(P_0)$.

Several observations need to be made. First, the intruder specification underlying penetrator strands follows the Dolev-Yao model [15, 28]. The parametric strands in Figure 7 are indeed closely related to the multiset rewriting intruder model $\mathcal{I}'$ above. A translation can be found in Sections 4.2.2 and 5.2.2 below.

As a final remark, notice that the transition system specification distinguishes between messages transmitted on the network (identified by the predicate symbol N) and messages intercepted and manipulated by the intruder. Indeed, the predicate I implements a private database, a workshop for the fabrication of unauthorized messages, hidden from the honest principals of the system. No such distinction exists in the strand world. Therefore, it seems that the intruder dismantles and puts together messages in the open, under the eyes of the other principals in the system. This is not a problem as honest agents expect messages of a very specific format. Moreover, when such a principal accepts the result of penetrator manipulations, this can be viewed as a final product of message forgery rather than an intermediate step.

The concepts and extensions we have just introduced set the basis for the translations between the multiset rewriting approach to security protocol specification and strand constructions. We describe the two

directions of this translations in Sections 4 and 5, respectively.

4 From Multisets to Strands

We observed that multiset rewriting is a state-based specification language for concurrent systems, while the strand space formalism is process-based. In general, a translation from this first paradigm to the second needs to be rather elaborate to be faithful [2] as their atomic steps have different granularity. Multiset rewriting specifications for security protocols have however a particularly streamlined form, with one action per rule and a control structure highly regulated by the role state predicates. This will considerably simplify the translation in this direction. The basic idea will be to map a set of multiset rewrite rules specifying a role to a parametric strand. In particular, rules will correspond to nodes, and the role state predicates will be replaced by the backbone ($\Longrightarrow$) of the strand. The technique is described in a more general form in [3].

It will be convenient to stage this translation into two steps: we first operate within the multiset rewriting formalism and transform a regular protocol theory into an equivalent but more manageable *normal form* (Section 4.1). Normal protocols theories are then rather directly mapped to strands (Section 4.2), which permits a very simple proof of correctness.

4.1 Normal Protocol Theories

A normal protocol theory collapses the generation step of each role and its first action of the execution in a single rule. It also requires that all the nonces used in a role be chosen up-front. We will now formalize this intuition and show how to normalize a regular protocol theory. For simplicity, we will describe the two parts of this transformation as if they ere two separate steps. Note that these transformations are only used for mathematical convenience as we devise a mapping from multiset rewriting specifications to the strand model: non-normal, and even non-regular, protocol theories are often more perspicuous than their normalized counterparts.

Role generation rule: We subsume the role generation rule of every role ρ, *i.e.* the rule $r_{\rho 0} : \boldsymbol{\pi}(\vec{x}) \longrightarrow \mathsf{A}_{\rho 0}(\vec{x}), \boldsymbol{\pi}(\vec{x})$, into the first rule of ρ. For each of its two schematic forms:

$$\begin{array}{ll} r_{\rho 1} : & \mathsf{A}_{\rho 0}(\vec{x}) \longrightarrow \exists \vec{n}.\, \mathsf{A}_{\rho 1}(\vec{x}, \vec{n}), \mathsf{N}(m(\vec{x}, \vec{n})) \\ r_{\rho 1} : & \mathsf{A}_{\rho 0}(\vec{x}), \mathsf{N}(m(\vec{x}, \vec{y})) \longrightarrow \mathsf{A}_{\rho 1}(\vec{x}, \vec{y}) \end{array}$$

we obtain the following rules:

$$\begin{array}{ll} \ddot{r}_{\rho 1} : & \boldsymbol{\pi}(\vec{x}) \longrightarrow \exists \vec{n}.\, \mathsf{A}_{\rho 1}(\vec{x}, \vec{n}), \mathsf{N}(m(\vec{x}, \vec{n})), \boldsymbol{\pi}(\vec{x}) \\ \ddot{r}_{\rho 1} : & \boldsymbol{\pi}(\vec{x}), \mathsf{N}(m(\vec{x}, \vec{y})) \longrightarrow \mathsf{A}_{\rho 1}(\vec{x}, \vec{y}), \boldsymbol{\pi}(\vec{x}) \end{array}$$

respectively. Observe that, by definition of role state predicate, the parameters $\vec{x}$ include the arguments of the elided $\mathsf{A}_{\rho 0}$ (as usual, $m(\vec{x})$ does not need to mention each variable in $\vec{x}$). This amounts to setting initial values in the first step of a role, rather than prior to any message exchange.

If $\mathcal{R}$ is a regular protocol theory, we will denote the effect of this transformation as $\ddot{\mathcal{R}}$. If $\boldsymbol{S}$ is a state, the transformed state $\ddot{\boldsymbol{S}}$ is obtained by dropping every mention of an initial role state $\mathsf{A}_{\rho 0}$ from $\boldsymbol{S}$. Clearly, $\ddot{\boldsymbol{S}}_0 = \boldsymbol{S}_0$ for any initial state $\boldsymbol{S}_0$. Similarly, a transition sequence $\vec{r}$ is mapped to a sequence $\ddot{\vec{r}}$ from which all the instances of rules for the form $r_{\rho 0}$ have been dropped, and the uses of $r_{\rho 1}$ have been replaced with $\ddot{r}_{\rho 1}$.

The above transformation is sound and complete as witnessed by the following result:

Lemma 4.1

Let $\mathcal{R}$ be a regular protocol theory with initial state $\boldsymbol{S}_0$ and $\boldsymbol{S}$ a state. Then,

1. *If $\boldsymbol{S}_0 \xrightarrow{\vec{r}}{}^*_{\mathcal{R}} \boldsymbol{S}$, then $\boldsymbol{S}_0 \xrightarrow{\vec{\ddot{r}}}{}^*_{\ddot{\mathcal{R}}} \ddot{\boldsymbol{S}}$.*
2. *If $\boldsymbol{S}_0 \xrightarrow{\vec{\ddot{r}}}{}^*_{\ddot{\mathcal{R}}} \ddot{\boldsymbol{S}}$, then $\boldsymbol{S}_0 \xrightarrow{\vec{r}}{}^*_{\mathcal{R}} \boldsymbol{S}$.*

Proof: In both cases, the proof proceeds by induction on the length of the given transition sequences.

1. If $\vec{r} = \cdot$, then the result follows immediately since $\vec{\ddot{r}} = \cdot$.
   Assume then that the transition sequence at hands has the form $(\vec{r}, r)$, with $\boldsymbol{S}_0 \xrightarrow{\vec{r}}{}^*_{\mathcal{R}} \boldsymbol{S}'$ and $\boldsymbol{S}' \xrightarrow{r}_{\mathcal{R}} \boldsymbol{S}$ for some state $\boldsymbol{S}'$. By induction hypothesis, we know that $\boldsymbol{S}_0 \xrightarrow{\vec{\ddot{r}}}{}^*_{\ddot{\mathcal{R}}} \ddot{\boldsymbol{S}}'$. We then show by cases on r that this property can be extended to $(\vec{r}, r)$.
   - If r is any rule beside $r_{\rho 0}$ or $r_{\rho 1}$, then its applicability does not change when going from $\boldsymbol{S}'$ to $\ddot{\boldsymbol{S}}'$ since this transformation only affects initial role states. For the same reason, its application clearly produces $\ddot{\boldsymbol{S}}$.
   - If r has the form $r_{\rho 1}$ for some role ρ, then $\ddot{r}$ differs from r only by the addition of the persistent predicated $\boldsymbol{\pi}(\vec{x})$ of $r_{\rho 0}$. It is therefore applicable only if the proper instance of these predicates, $\boldsymbol{\pi}(\vec{t})$ say, holds in the current state $\boldsymbol{S}'$. By definition of state, this reduces to requiring that $\boldsymbol{\pi}(\vec{t}) \in \Pi$. However, by definition of persistent information, Π is contained in every state. Thus it is sufficient to show that there exists one state $\boldsymbol{S}_{\boldsymbol{\pi}}$ such that $\boldsymbol{\pi}(\vec{t}) \subseteq \boldsymbol{S}_{\boldsymbol{\pi}}$. Now, since $r_{\rho 1}$ is enabled in $\boldsymbol{S}'$, it must be the case that an identical instantiation of $r_{\rho 0}$ appears in $\vec{r}$. The state to which $r_{\rho 0}$ applies is precisely $\boldsymbol{S}_{\boldsymbol{\pi}}$.
   - If r has the form $r_{\rho 0}$, then $\ddot{\boldsymbol{S}}' = \ddot{\boldsymbol{S}}$. By induction hypothesis, $\boldsymbol{S}_0 \xrightarrow{\vec{\ddot{r}}}{}^*_{\ddot{\mathcal{R}}} \ddot{\boldsymbol{S}}'$, which is what we want since the transformation cancels r.
2. If $\vec{\ddot{r}} = \cdot$, the result is immediate.
   Otherwise, the transformation sequence will have the form $(\vec{\ddot{r}}, \ddot{r})$. Again, we assume the result holds for $\vec{\ddot{r}}$ and show by cases that it must also hold for the extended sequence.
   - If $\ddot{r} = \ddot{r}_{\rho 1}$ for some role ρ, then we restore it as $r_{\rho 0}$ immediately followed by $r_{\rho 1}$.
   - In all other cases, we leave $\ddot{r}$ unchanged. □

Observe that applying this transformation and then "undoing" it as specified in the above lemma is not equivalent to the identical transformation: going in the reverse direction, we group occurrences of $r_{\rho 0}$ and $r_{\rho 1}$ together, and moreover we eliminate every isolated instance of $r_{\rho 0}$.

Stating this lemma relative to general rather than regular protocol theories would be incorrect: assume that $\boldsymbol{S}_0 \xrightarrow{\vec{r}}{}^*_{\mathcal{R}} \boldsymbol{S}_1$ thanks to the initialization rule $r_{\rho 0}$ of some role ρ. As in Section 2.3, assume also that the first message exchange rule $r_{\rho 1}$ of this role contains a persistent predicate which does not have any instantiation in $\boldsymbol{\Pi}$. The normal form of $r_{\rho 0}$ would then contain this constraint, making it inapplicable to any state $\boldsymbol{S}_1$ would be mapped to. This scenario can clearly not occur when starting from a *regular* protocol theory since, by definition, all the accesses to persistent predicate are confined in the role instantiation rule.

An alternative way to go about it is to statically bind persistent information to any point in a protocol description where it is used. A realization of this idea by means of a strong typing infrastructure is at the basis of *MSR* [7], a thorough redesign of the multiset rewriting formalism discussed in this

paper. Although the language discussed here has been superceded by *MSR* for any practical purposes, the results shown here should not be dismissed as obsolete. Indeed, *MSR* builds on the language considered here, and any mapping between *MSR* and strand spaces is likely to use that language as a meeting point, or risk a considerably more complex translation.

Nonces: We further transform protocol theories so that all nonces generated by a role are preemptively chosen in the first rule of that role. We accomplish this by adding extra arguments to role state predicates, and pass the nonces generated in the first rule to subsequent uses through fresh variables in these predicates. Since roles are bounded, there are only a small finite number of nonces that need to be generated in an entire role. This transformation intuitively means that a participant should roll all her dice immediately, and look at them as needed later.

More formally, let ρ be the multiset rewriting specification of a role as from the previous transformation, and let $e_{\rho i}$ be the number of existentially quantified variables in rule $r_{\rho i}$, for $i = 1..|\rho|$. We map each role state predicate $\mathsf{A}_{\rho i}(\vec{x})$ in ρ to a predicate of the form

$$\bar{\mathsf{A}}_{\rho i}(\vec{x}, \vec{n}_{i+1}, \ldots, \vec{n}_{|\rho|})$$

where, for $j = i + 1..|\rho|$, there are exactly $e_{\rho j}$ elements in $\vec{n}_j$, and each of the added arguments is a distinct new variable.

We transform rules by replacing each state predicate $\mathsf{A}_{\rho i}$ with $\bar{\mathsf{A}}_{\rho i}$, and moving existential quantifiers to the first rule of the role. As a result, we are left with the following *normalized rules*:

Role generation rules:

$$\begin{array}{lrcll} \bar{r}_{\rho 1}: & \boldsymbol{\pi}(\vec{x}) & \longrightarrow & \exists \vec{n}.\, \bar{\mathsf{A}}_{\rho 1}(\vec{x}, \vec{n}), & \mathsf{N}(m(\vec{x}, \vec{n})), \boldsymbol{\pi}(\vec{x}) \\ \bar{r}_{\rho 1}: & \boldsymbol{\pi}(\vec{x}), \mathsf{N}(m(\vec{x}, \vec{y})) & \longrightarrow & \exists \vec{n}.\, \bar{\mathsf{A}}_{\rho 1}(\vec{x}, \vec{y}, \vec{n}), & \boldsymbol{\pi}(\vec{x}) \end{array}$$

Other rules:

$$\begin{array}{lrcll} \bar{r}_{\rho i+1}: & \bar{\mathsf{A}}_{\rho i}(\vec{x}) & \longrightarrow & \bar{\mathsf{A}}_{\rho i+1}(\vec{x}), & \mathsf{N}(m(\vec{x})) \\ \bar{r}_{\rho i+1}: & \bar{\mathsf{A}}_{\rho i}(\vec{x}), \mathsf{N}(m(\vec{x}, \vec{y})) & \longrightarrow & \bar{\mathsf{A}}_{\rho i+1}(\vec{x}, \vec{y}) & \end{array}$$

where all the newly introduced variables in rule $\bar{r}_{\rho 1}$ are existentially quantified. Given a role ρ, we denote the normalized specification as $\bar{\rho}$. We write $\overline{\mathcal{R}}$ for the application of this transformation to a protocol theory $\mathcal{R}$.

In order to formally relate a regular protocol theory with its normalized form, we need to assess the effect of normalization on states. Given a ground predicate P in a state $\boldsymbol{S}$, we construct the open term $\overline{P}$ corresponding to the possible normalizations of P as follows:

$$\begin{cases} \overline{\mathsf{A}_{\rho i}(\vec{t})} = \bar{\mathsf{A}}_{\rho i}(\vec{t}, \vec{n}_i, \ldots, \vec{n}_{|\rho|}) & \text{where } \vec{n}_i, \ldots, \vec{n}_{|\rho|} \text{ consist of distinct variables} \\ \overline{P} = P & \text{if } P \text{ is not a role state predicate} \end{cases}$$

It is easy to extend this definition to *open states*: if $\boldsymbol{S}$ is a state, we construct the open multiset $\overline{\boldsymbol{S}}$ representing all normalized states it is mapped to. $\overline{\boldsymbol{S}}$ is defined as follows:

$$\overline{\boldsymbol{S}} = \{\!|\overline{P} : P \leftarrow \boldsymbol{S}|\!\}$$

where $\{\!|\ldots|\!\}$ is the multiset equivalent of the usual set notation $\{\ldots\}$, and $x \leftarrow M$ denotes multiplicity-conscious multiset membership. We shall choose different variables for each $\overline{P}$ in $\overline{\boldsymbol{S}}$. Observe that since the initial state $\boldsymbol{S}_0$ does not contain role state predicates, we have that $\overline{\boldsymbol{S}_0} = \boldsymbol{S}_0$.

The mapping between an open state $\overline{\boldsymbol{S}}$ and states that can be processed by transitions is done by means of substitutions ξ that map each variable in $\overline{\boldsymbol{S}}$ to a distinct constant that does not appear in $\boldsymbol{S}$. Observe that $\overline{\boldsymbol{S}}[\xi]$ is a (ground) state.

The definition of transition does not change, but we will denote a transition sequence that uses normalized rules as $\vec{\bar{r}}$ with the usual subscripts. We will shortly see how to normalize a transition sequence $\vec{r}$.

Given these various definitions, we are now in a position to prove that normalization preserves transitions. We have the following result.

Lemma 4.2

Let $\ddot{\mathcal{R}}$ be a regular protocol theory that has been subjected to the role generation transformation in the first part of this section, $\boldsymbol{S}_0$ the initial state, and $\boldsymbol{S}$ a state. Let moreover ξ be an arbitrary substitution from the variables in $\overline{\boldsymbol{S}}$ to distinct unused constants. Then,

1. *If $\boldsymbol{S}_0 \xrightarrow{\vec{r}}{}^*_{\ddot{\mathcal{R}}} \boldsymbol{S}$, then $\boldsymbol{S}_0 \xrightarrow{\vec{\bar{r}}}{}^*_{\bar{\mathcal{R}}} \overline{\boldsymbol{S}}[\xi]$.*
2. *If $\boldsymbol{S}_0 \xrightarrow{\vec{\bar{r}}}{}^*_{\bar{\mathcal{R}}} \overline{\boldsymbol{S}}[\xi]$, then $\boldsymbol{S}_0 \xrightarrow{\vec{r}}{}^*_{\ddot{\mathcal{R}}} \boldsymbol{S}$.*

Proof: In both cases, the proof proceeds by induction on the length of the given transition sequences. We will examine them in turn.

1. If $\vec{r} = \cdot$, then the result holds trivially since $\overline{\boldsymbol{S}_0} = \boldsymbol{S}_0$. Clearly, $\vec{\bar{r}} = \cdot$.
   Assume then that the given derivation sequence has the form $(\vec{r}, r)$ so that there is a state $\boldsymbol{S}'$ such that $\boldsymbol{S}_0 \xrightarrow{\vec{r}}{}^*_{\ddot{\mathcal{R}}} \boldsymbol{S}'$ and $\boldsymbol{S}' \xrightarrow{r}_{\ddot{\mathcal{R}}} \boldsymbol{S}$. By induction hypothesis, for any substitution ξ' of the required form, $\boldsymbol{S}_0 \xrightarrow{\vec{\bar{r}}}{}^*_{\bar{\mathcal{R}}} \overline{\boldsymbol{S}'}[\xi']$. We show by cases on the structure of r that this property extends to $(\vec{\bar{r}}, \bar{r})$. For the sake of conciseness, we will only develop the (more challenging) cases where r is a sending rule. The other situations follow the same pattern.

   Sending, initial: $\quad r : \ \boldsymbol{\pi}(\vec{x}) \ \longrightarrow \ \exists \vec{n}.\, \mathsf{A}_{\rho 1}(\vec{x}, \vec{n}), \mathsf{N}(m), \boldsymbol{\pi}(\vec{x}).$

   $\bar{r}$ has then the form

   $$\bar{r} : \ \boldsymbol{\pi}(\vec{x}) \ \longrightarrow \ \exists \vec{n}.\, \exists \vec{n}_2, \ldots, \vec{n}_{|\rho|}.\, \begin{bmatrix} \bar{\mathsf{A}}_{\rho 1}(\vec{x}, \vec{n}, \vec{n}_2, \ldots, \vec{n}_{|\rho|}), \\ \mathsf{N}(m), \boldsymbol{\pi}(\vec{x}) \end{bmatrix}$$

   The normalized rule $\bar{r}$ is enabled in $\overline{\boldsymbol{S}'}[\xi']$ since r is enabled in $\boldsymbol{S}'$ and their preconditions consist only of persistent information. Its application yields the state $\overline{\boldsymbol{S}'}[\xi'], \bar{\mathsf{A}}_{\rho 1}(\vec{t}, \vec{c}, \vec{c}_2, \ldots, \vec{c}_{|\rho|}), \mathsf{N}(m)$ for constants $\vec{c}, \vec{c}_2, \ldots, \vec{c}_{|\rho|}$ that can be arbitrary as long as they are distinct from each other and from constants in $\overline{\boldsymbol{S}'}[\xi']$.

   Now, $\boldsymbol{S} = \boldsymbol{S}', \mathsf{A}_{\rho 1}(\vec{t}, \vec{c}), \mathsf{N}(m)$, and therefore, we have that $\overline{\boldsymbol{S}} = \overline{\boldsymbol{S}'}, \bar{\mathsf{A}}_{\rho 1}(\vec{t}, \vec{c}, \vec{n}_2, \ldots, \vec{n}_{|\rho|}), \mathsf{N}(m)$ for distinct variables $\vec{n}_2, \ldots, \vec{n}_{|\rho|}$. It then suffices to define ξ as the extension of ξ' with the mapping $(\vec{c}_2, \ldots, \vec{c}_{|\rho|})/(\vec{n}_2, \ldots, \vec{n}_{|\rho|})$ to obtain the desired result.

   Sending, non-initial: $\quad r : \ \mathsf{A}_{\rho i}(\vec{x}) \ \longrightarrow \ \exists \vec{n}.\, \mathsf{A}_{\rho i+1}(\vec{x}, \vec{n}), \mathsf{N}(m).$

   $\bar{r}$ has then the form

   $$\bar{r} : \ \bar{\mathsf{A}}_{\rho i}(\vec{x}, \vec{n}, \vec{n}_{i+1}, \ldots, \vec{n}_{|\rho|}) \ \longrightarrow \ \bar{\mathsf{A}}_{\rho 1}(\vec{x}, \vec{n}, \vec{n}_{i+1}, \ldots, \vec{n}_{|\rho|}), \mathsf{N})$$

   By definition, $\boldsymbol{S}' = \boldsymbol{S}'', \mathsf{A}_{\rho i}(\vec{t})$ and $S = S'', \mathsf{A}_{\rho i+1}(\vec{t}, \vec{c}), \mathsf{N}(m)$, for distinct new constants $\vec{c}$. We then have that $\overline{\boldsymbol{S}'} = \overline{\boldsymbol{S}''}, \bar{\mathsf{A}}_{\rho i}(\vec{t}, \vec{n}, \vec{n}_{i+1}, \ldots, \vec{n}_{|\rho|})$. Therefore $\bar{r}$ is applicable to $\overline{\boldsymbol{S}'}[\xi']$ for

any ξ' satisfying the above conditions. In particular, this holds for all such substitutions $\xi'_{\vec{c}/\vec{n}}$ that map $\vec{n}$ to $\vec{c}$. The application of $\bar{r}$ to $\overline{\boldsymbol{S}'}[\xi'_{\vec{c}/\vec{n}}]$ produces the state $(\overline{\boldsymbol{S}''}, \bar{\mathsf{A}}_{\rho i+1}(\vec{t}, \vec{n}, \vec{n}_{i+1}, \ldots, \vec{n}_{|\rho|}), \mathsf{N}(m))[\xi'_{\vec{c}/\vec{n}}]$, which we can rewrite as $(\overline{\boldsymbol{S}''}, \bar{\mathsf{A}}_{\rho i+1}(\vec{t}, \vec{c}, \vec{n}_{i+1}, \ldots, \vec{n}_{|\rho|}), \mathsf{N}(m))[\xi'_{\vec{c}/\vec{n}}]$. The desired result follows then immediately once we observe that $\overline{\boldsymbol{S}} = \overline{\boldsymbol{S}''}, \bar{\mathsf{A}}_{\rho i+1}(\vec{t}, \vec{c}, \vec{n}_{i+1}, \ldots, \vec{n}_{|\rho|}), \mathsf{N}(m)$.

2. The proof of the reverse direction of this lemma uses the techniques we just deployed: in the inductive step, we proceed by case distinction on the form of the last transition applied, assuming that the result holds up to this last step. The reverse of the substitution manipulations that we witnessed above are used to drop the added arguments of the role state predicates, which allows us to do without the substitution. □

In the following we will start from a regular protocol theory $\mathcal{R}$ and apply these two transformations in sequence. For clarity reasons, we will generally write $\bar{\mathcal{R}}$ when $\ddot{\mathcal{R}}$ would be appropriate. We extend this convention to roles and states.

The following corollary chains the above results together. It also considers protocols augmented with the standard intruder theory $\mathcal{I}$. It must be observed that the above transformations do not have any effect on $\mathcal{I}$.

Corollary 4.3
Let $\mathcal{R}$ be a regular protocol theory, $\boldsymbol{S}_0$ the initial state, and $\boldsymbol{S}$ a state. Let moreover ξ be an arbitrary substitution from the variables in $\overline{\boldsymbol{S}}$ to distinct unused constants. Then,

1. *If $\boldsymbol{S}_0 \xrightarrow{\vec{r}}{}^*_{\mathcal{R},\mathcal{I}} \boldsymbol{S}$, then $\boldsymbol{S}_0 \xrightarrow{\vec{\bar{r}}}{}^*_{\bar{\mathcal{R}},\mathcal{I}} \overline{\boldsymbol{S}}[\xi]$.*
2. *If $\boldsymbol{S}_0 \xrightarrow{\vec{r}}{}^*_{\bar{\mathcal{R}},\mathcal{I}} \overline{\boldsymbol{S}}[\xi]$, then $\boldsymbol{S}_0 \xrightarrow{\vec{r}}{}^*_{\mathcal{R},\mathcal{I}} \boldsymbol{S}$.*

Proof: This is a direct consequence of Lemmas 4.1 and 4.2 once we observe that the intruder rules never access the role state predicates of a principal. Therefore, the elision of the state predicate A_0 is invisible to the intruder. Similarly, the intruder cannot see nor take advantage of the fact that all existentials in a normal role have been instantiated up-front since they are safely stored in $\mathsf{A}_{\rho i}(\vec{x}, \vec{n}_i, \ldots, \vec{n}_{|\rho|})$ until they are made visible in a message. □

4.2 Translation

We are now in a position to translate protocol representations expressed in the multiset rewriting formalisms into strands. We first show in Section 4.2.1 how to map a general protocol theory into a set of parametric strands, and then relate the intruder theory directly to the penetrators strands in Section 4.2.2. In Section 4.2.4, we prove that this translation preserve transitions after discussing how states are handled in Section 4.2.3.

4.2.1 From Protocol Theories to Parametric Strands

To each normalized role specification $\bar{\rho}$, we associate a parametric strand $\ulcorner \bar{\rho} \urcorner$ of the following form

$$\underline{\rho(\vec{x}, \vec{y}, \vec{n})} \quad \boxed{\vec{n} \text{ fresh}, \boldsymbol{\pi}(\vec{x})}$$

where $\vec{n}$ are the existential variables mentioned in the first rule $\bar{r}_{\rho 1}$ of this role, $\boldsymbol{\pi}(\vec{x})$ are the persistent predicates accessed in this rule, and $\vec{y}$ are the other variables appearing in the role ($\vec{x}, \vec{y}, \vec{n}$ appear therefore in its last role state predicate).

Next, we associate a parametric node $\nu_{\bar{r}_{\rho i}}$ with each rule $\bar{r}_{\rho i}$. The embedded message is the message appearing in the antecedent or the consequent of the rule, the distinction being accounted for by the associated action. More precisely, we have the following translation (where we have omitted the argument of the state predicates, the indication of the variables occurring in the message, persistent information, and the existential quantifiers appearing in the role generation rule):

$$\begin{aligned} \ulcorner \bar{\mathsf{A}}_{\rho i} \longrightarrow \bar{\mathsf{A}}_{\rho i+1}, \mathsf{N}(m) \urcorner &= +m \\ \ulcorner \bar{\mathsf{A}}_{\rho i}, \mathsf{N}(m) \longrightarrow \bar{\mathsf{A}}_{\rho i+1} \urcorner &= -m \end{aligned}$$

where $\ulcorner \_ \urcorner$ is our translation function.

Finally, we set the backbone of this parametric strand according to the order of the indices of the nodes (and rules):

$$\nu_{\bar{r}_{\rho i}} \Longrightarrow \nu_{\bar{r}_{\rho j}} \qquad \text{iff} \qquad j = i + 1.$$

In this way, we are identifying the role state predicates of the transition system specification with the $\Longrightarrow$-edges constituting the backbone of the corresponding parametric strand. Notice that the well-founded ordering over role state predicates is mapped onto the acyclicity of the $\Longrightarrow$-arrows of the strand constructions.

This completes our translation as far as roles, and therefore protocols, are concerned. Applying it to the Needham-Schroeder protocol yields exactly the parametric strand specification of Figure 4 presented in Section 3. Given a set of roles $\mathcal{R}$ in the transition system notation, we indicate the corresponding set of parametric strands as $\ulcorner \overline{\mathcal{R}} \urcorner$. We will give correctness results at the end of this section after showing how to translate global states.

4.2.2 From Intruder Theory to Penetrator Strands

The introduction of the alternate intruder theory $\mathcal{I}'$ in Section 2.4 enables a trivial mapping to penetrator strands: we simply map every intruder rule to the corresponding penetrator strand, with the exception of rec' and snd', which do not have any correspondent. In symbols:

$$\begin{aligned} \ulcorner \mathrm{rec}'(m) \urcorner &= \textit{none} & \ulcorner \mathrm{snd}'(m) \urcorner &= \textit{none} \\ \ulcorner \mathrm{dcmp}'(m_1, m_2) \urcorner &= S(m_1, m_2) & \ulcorner \mathrm{cmp}'(m_1, m_2) \urcorner &= C(m_1, m_2) \\ \ulcorner \mathrm{decr}'(m, k) \urcorner &= D(m, k) & \ulcorner \mathrm{encr}'(m, k) \urcorner &= E(m, k) \\ \ulcorner \mathrm{nnc}'(n) \urcorner &= N(n) & \ulcorner \mathrm{pers}'(m) \urcorner &= M(m) \\ \ulcorner \mathrm{dup}(m) \urcorner &= T(m) & \ulcorner \mathrm{del}(m) \urcorner &= F(m) \end{aligned}$$

where we have equipped the intruder rules with arguments in the obvious way. We also need to map the initial intruder knowledge $\boldsymbol{I}_0$ to a set P_0 of messages initially known to the intruder, to be processed by the penetrator strand M': $\ulcorner \boldsymbol{I}_0 \urcorner = \{m : \mathsf{I}(m) \in \boldsymbol{I}_0\}$. Every access to a message $\mathsf{I}(m)$ in $\boldsymbol{I}_0$ will be translated to an application of the penetrator strand $M'(m)$.

4.2.3 Relating States and Configurations

In order to show that a transition system specification and its strand translation behave in the same way, we need to relate states and configurations. We do not need to give an exact mapping, since a configuration embeds a bundle expressing the execution up to the current point in fine detail. A state is instead a much

simpler construction that does not contain any information about how it has been reached. Therefore, we will consider some properties that a configuration should have to be related to a state.

We say that a state $\boldsymbol{S} = (\boldsymbol{\Pi}, \boldsymbol{A}, \boldsymbol{N}(\vec{m}), \boldsymbol{I}(\vec{m}'))$ is *compatible* with a strand configuration $(\sigma, \sigma^\sharp)$ relative to a (regular, but not necessarily normal) protocol theory $\mathcal{R}$, written $\boldsymbol{S} \sim_{\mathcal{R}} (\sigma, \sigma^\sharp)$, if the following conditions hold:

- $\mathrm{Fr}(\sigma) = \{\vec{m}, \vec{m}'\}$.
- Let $\mathsf{A}_{\rho i}(\vec{t_\rho}, \vec{c_\rho})$ in $\boldsymbol{A}$ be the instantiation of the i-th role state predicate of a role ρ in $\mathcal{R}$ with terms $\vec{t}_\rho$ and fresh nonces $\vec{c}_\rho$. Then,
  - $\sigma^\sharp$ contains a strand $s^\rho(\vec{c_\rho}, \vec{t_\rho})$, obtained by instantiating the strand $s^\rho = \ulcorner \rho \urcorner$ with terms $\vec{t}_\rho$ and new constants $\vec{c}_\rho$.
  - σ contains an initial prefix of $s^\rho(\vec{t})$ whose last node has index i.

  Moreover every non-penetrator strand in $(\sigma, \sigma^\sharp)$ is obtained in this way.
- Every instance of a penetrator strand in $(\sigma, \sigma^\sharp)$ is completely contained in σ.

Intuitively, we want the state and the configuration to mention the same nonces, to have the same messages in transit (including the data currently processed by the intruder), to be executing corresponding role instances and have them be stopped at the same point.

4.2.4 Transition to Move Sequences

Given these definitions, we can state the correctness result for our translation of transition systems into strand constructions. We shall start by limiting our attention to normal protocol theories together with the modified intruder theory introduced in Section 2.4.

Lemma 4.4
Let $\overline{\mathcal{R}}$ be a normal protocol theory, $\boldsymbol{I}_0$ some initial intruder knowledge, and $\ulcorner \boldsymbol{I}_0 \urcorner$ its strand translation. If $\boldsymbol{\Pi}, \boldsymbol{I}_0 \xrightarrow{\vec{r}}{}^*_{\mathcal{I}', \overline{\mathcal{R}}} \boldsymbol{S}$ is a normal multiset rewriting transition sequence over $\mathcal{I}', \overline{\mathcal{R}}$ from the empty state to state $\boldsymbol{S}$, then there is a configuration $(\sigma, \sigma^\sharp)$ and a sequence of moves $\vec{o}$ such that

$$(\cdot, \cdot) \xmapsto{\vec{o}}{}^*_{\mathcal{P}(\ulcorner \boldsymbol{I}_0 \urcorner), \ulcorner \overline{\mathcal{R}} \urcorner} (\sigma, \sigma^\sharp)$$

is a strand transition sequence from the empty configuration $(\cdot, \cdot)$ to $(\sigma, \sigma^\sharp)$, and $\boldsymbol{S} \sim_{\overline{\mathcal{R}}} (\sigma, \sigma^\sharp)$, i.e. $\boldsymbol{S}$ is compatible with $(\sigma, \sigma^\sharp)$.

Proof: The proof proceeds by induction on $\vec{r}$. The base case is trivial. The inductive step does a case analysis on the last rule applied in $\vec{r}$. Intruder rules from $\mathcal{I}'$ are directly emulated by the corresponding penetrator strands, as defined in Section 4.2.2. The use of protocol rule $r_{\rho i}$ is emulated by a move involving the corresponding node in $\ulcorner \bar{\rho} \urcorner$. For each of these possibilities, we show that the corresponding move in the strand world is possible, and that it preserves the compatibility relation.

We omit formalizing this proof as it relies on exactly the same techniques as the proofs of previous results. □

We can now extend this result to any regular (not necessarily normal) theory together with the standard intruder model. We have the following theorem:

Theorem 4.5

Let $\mathcal{R}$ a regular protocol theory and $\boldsymbol{I}_0$ be some initial intruder knowledge. For every regular multiset rewriting transition sequence $\mathbf{\Pi}, \boldsymbol{I}_0 \xrightarrow{\vec{r}}{}^*_{\mathcal{I},\mathcal{R}} \boldsymbol{S}$ there is a configuration $(\sigma, \sigma^\sharp)$ and a sequence of moves $\vec{o}$ such that

$$(\cdot,\cdot) \overset{\vec{o}}{\longmapsto}{}^*_{\mathcal{P}(\ulcorner I_0 \urcorner),\ulcorner \overline{\mathcal{R}} \urcorner} (\sigma, \sigma^\sharp)$$

is a strand transition sequence from the empty configuration $(\cdot,\cdot)$ to $(\sigma, \sigma^\sharp)$, and $\boldsymbol{S} \sim_{\mathcal{R}} (\sigma, \sigma^\sharp)$.

Proof: This is a simple corollary of the above lemma mediated by an application of Lemma 4.1 to move between regular and normal protocol theories, and Property 2.2 reconcile using the standard vs. the modified intruder theory. □

Observe that we cannot further relax the statement of this theorem to consider arbitrary (*i.e.* non-regular) protocol theories as regularization does not preserve transition sequences.

5 From Strands to Multisets

We will now show how to translate a set of parametric strands into a set of multiset rewrite rules that preserve multistep transitions. To this end, we rely on relatively standard techniques to map process-based representations of security protocols to state-based descriptions [3]. However, we shall first address a slight mismatch between the two formalisms (Section 5.1). This technical adjustment of our definition of strands will produce precisely the regular role transition rules we originally defined in Section 2. The translation itself and its proof of correctness are then rather straightforward (Section 5.2).

5.1 Decorated Strands

In the previous section, we have observed and taken advantage of the fact that there is a close affinity between the rules in the transition system specification of a role and the nodes in a parametric strand. More precisely, a node together with the outgoing or incoming $\longrightarrow$-edge and an indication of what to do next corresponds to a transition. In transition systems, "what to do next"is specified through the role state predicates $\mathsf{A}_{\rho i}$; in strand constructions, by means of the $\Longrightarrow$-edges. Therefore, using the same intuition as in Section 4, we will translate $\Longrightarrow$-edges to state predicates. We need to equip these predicates with the appropriate arguments (while we were able to simply drop them in the inverse translation). This method is relatively standard when mapping process-based representations of security protocols to state-based descriptions [3].

Before describing how to do so, we will address two other minor syntactic discrepancies: the absence of an (explicit) strand equivalent of the role generation rule $\boldsymbol{\pi}(\vec{x}) \longrightarrow \mathsf{A}_{\rho 0}(\vec{x}), \boldsymbol{\pi}(\vec{x})$, and the fact that, in the transition system specification of a role, there is a final state predicate that lingers in the global state no matter what other transitions take place.

Role Generation transition: We add a dummy initial node, say $\top$, to every strand, with no incoming or outgoing $\longrightarrow$-edges, and one outgoing $\Longrightarrow$-edge to the original first node of the strand.

Final state: Dually, we alter the definition of strands to contain a final node, say $\bot$, again without any incoming or outgoing $\longrightarrow$-edge, and with one incoming $\Longrightarrow$-arrow from the original last node of the strand.

This corresponds to redefining strands as strings drawn from the language $\top(\pm\mathcal{M})^*\bot$, rather than just $(\pm\mathcal{M})^*$. Notice that now every (proper) event has both a predecessor and a successor $\Longrightarrow$-edge.

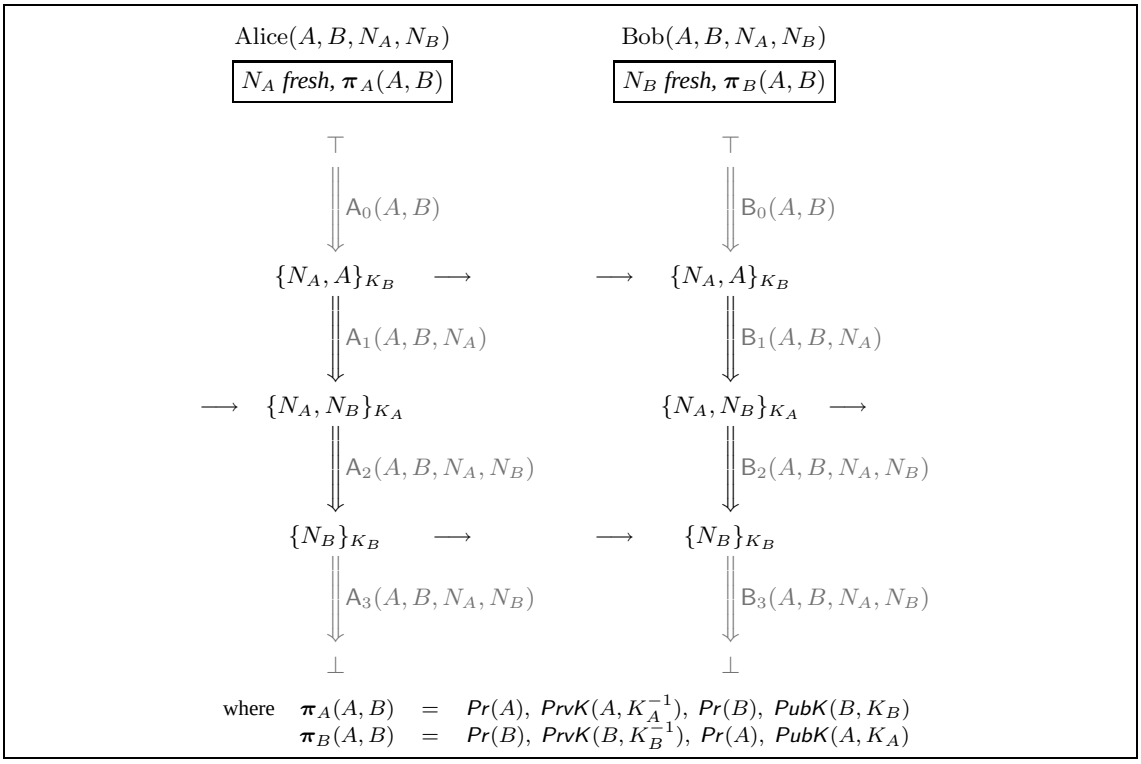


Figure 8. Extended Strand Specification of the Needham-Schroeder Protocol

With the addition of these auxiliary nodes, we can label each $\Longrightarrow$-arrow in a strand s with parameters $\vec{x}_s, \vec{n}_s$ ($\vec{n}_s$ marked *fresh*), in symbols,

$$\underline{\rho(\vec{x}_s, \vec{n}_s)}: \quad \boxed{\vec{n}_s \textit{ fresh, } \boldsymbol{\pi}(\vec{x}_s)}$$

and a predicate constant A_{si} with progressive indices i. In the case of parametric strands, we equip these labels with arguments drawn from its set of parameters as follows:

Initial arrow: $\top \Longrightarrow \nu$

This is the predicate A_{s0} labeling the $\Longrightarrow$-edge that links the added initial node $\top$ to the first node of the original strand. The arguments of A_{s0} will be $\vec{x}_s$, the persistent information used by the strand.

Successor arrow to a positive node:

$$\ldots \overset{\mathsf{A}_{si}(\vec{x})}{\Longrightarrow} +m(\vec{x}, \vec{n}) \Longrightarrow \ldots$$

Let $\mathsf{A}_{si}(\vec{x})$ be the label of the incoming $\Longrightarrow$-edge of a positive node $\nu = +m(\vec{x}, \vec{n})$, where m mentions known variables among $\vec{x}$ and unused nonces $\vec{n}$ among $\vec{n}_s$. Then the outgoing $\Longrightarrow$-arrow of ν will have label $\mathsf{A}_{si+1}(\vec{x}, \vec{n})$.

Successor arrow to a negative node:

$$\ldots \overset{\mathsf{A}_{si}(\vec{x})}{\Longrightarrow} -m(\vec{x}, \vec{y}) \Longrightarrow \ldots$$

Let $\mathsf{A}_{si}(\vec{x})$ be the label of the incoming $\Longrightarrow$-edge of a positive node $\nu = -m(\vec{x},\vec{y})$, where m mentions known variables among $\vec{x}$, and unseen data $\vec{y}$ (for examples nonces created by another party). Then, the outgoing $\Longrightarrow$-arrow of n will have label $\mathsf{A}_{si+1}(\vec{x},\vec{y})$.

Given a parametric strand s, we denote the result of applying these transformations as $\bar{s}$. If $\mathcal{S}$ is a set of parametric strands specifying a protocol, we write $\overline{\mathcal{S}}$ for the transformed set. Applying this transformation to the Needham-Schroeder protocol yields the enhanced strand specification in Figure 8, where the additions have been grayed out.

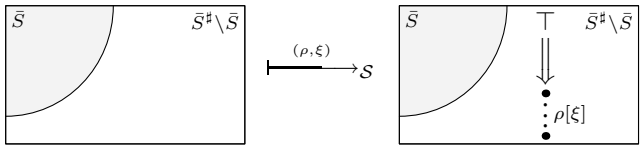
Since we have changed the syntax of a parametric strand, we need to upgrade its dynamics, originally presented in Section 2. First, an obvious alteration to the instantiation of a parametric strand: we apply the substitution to the labels of the $\Longrightarrow$-edges as well as to the messages embedded in the nodes. We carry on this change to the resulting bundles and configurations: every $\Longrightarrow$-edge between two nodes ν_1 and ν_2 now carries a label $\mathsf{A}_{si}(\vec{t})$. We indicate this as $\nu_1 \overset{\mathsf{A}_{si}(\vec{t})}{\Longrightarrow} \nu_2$ (or with its vertical equivalent). Notice that we erased this information in the reverse translation. Given a bundle σ and a configuration $(\sigma, \sigma^\sharp)$ relative to a set of parametric strands $\mathcal{S}$, we write $\bar{\sigma}$ and $(\bar{\sigma}, \bar{\sigma}^\sharp)$ for the corresponding entities relative to $\overline{\mathcal{S}}$.

The definition of one-step transition, in symbols $(\bar{\sigma}_1, \bar{\sigma}_1^\sharp) \overset{o}{\longmapsto}_{\overline{\mathcal{S}}} (\bar{\sigma}_2, \bar{\sigma}_2^\sharp)$, changes as follows:

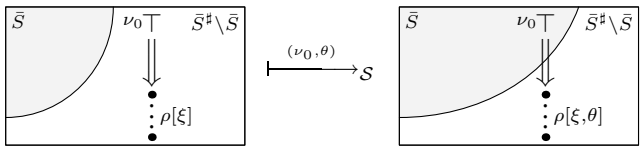
Extension of an existing strand: We proceed exactly as in Section 2, except for the fact that situations $\mathbf{S_0}$ and $\mathbf{R_0}$ in Section 3.3 do not apply.

Installation of a new strand:

We select a parametric strand ρ from $\overline{\mathcal{S}}$, instantiate it with a substitution ξ for its fresh variables and add the resulting strand $\rho[\xi]$ to $\bar{\sigma}_2^\sharp$. This corresponds to upgrading case $\mathbf{C_f}$ in Section 3.3 as outlined in the following figure. We do not formalize this transformation (call it $\mathbf{C_f}'$) it in full detail since it should be obvious how to obtain it.



Transition $\mathbf{C_i}$ is consequently upgraded to $\mathbf{C_i}'$ described in the following figure. Notice that we add the first node, $\top$, of $\rho[\xi,\theta]$ to $\bar{\sigma}_2$



As in the original case, multistep transitions are obtained by taking the reflexive and transitive closure of the above judgment.

This transformation is sound and complete with respect to our original system.

Lemma 5.1 *Let $\mathcal{S}$ be a set of parametric strands, and $(\sigma_1, \sigma_1^\sharp)$ and $(\sigma_2, \sigma_2^\sharp)$ two configurations on it. Then,*

$$(\sigma_1, \sigma_1^\sharp) \overset{\vec{o}}{\longmapsto}_{\mathcal{S}}^{*} (\sigma_2, \sigma_2^\sharp) \qquad \textit{if and only if} \qquad (\bar{\sigma}_1, \bar{\sigma}_1^\sharp) \overset{\vec{\bar{o}}}{\longmapsto}_{\overline{\mathcal{S}}}^{*} (\bar{\sigma}_2, \bar{\sigma}_2^\sharp)$$

where $\vec{\bar{o}}$ is obtained from $\vec{o}$ by extending the given transformation to traces.

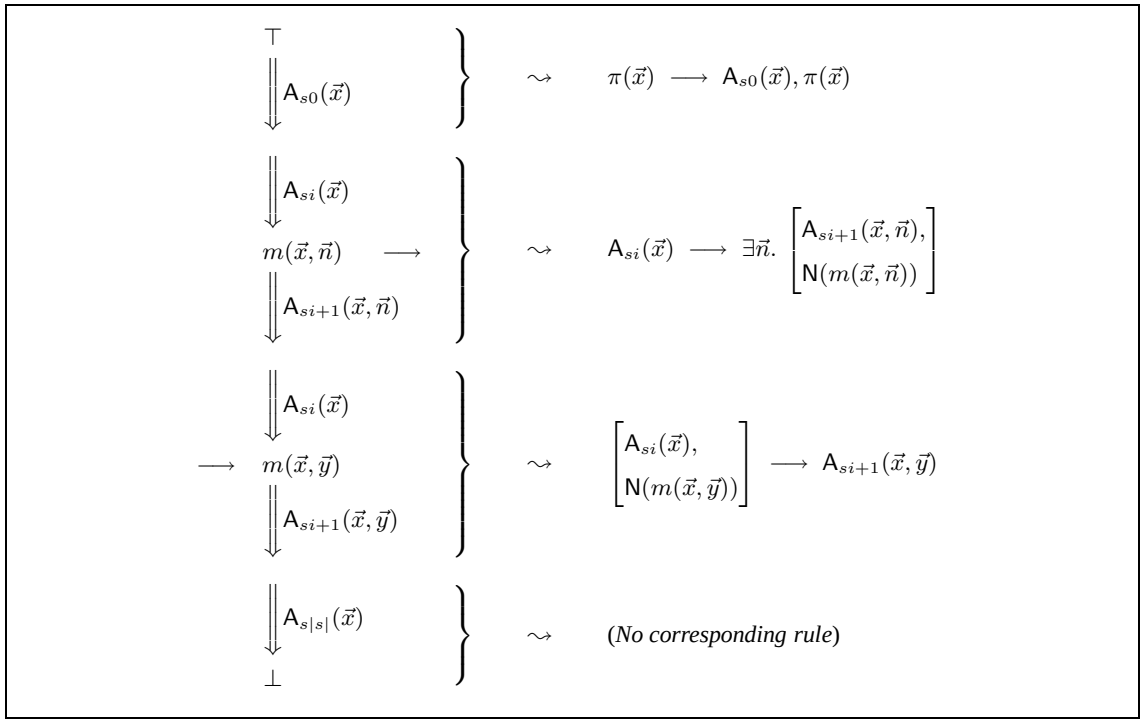


Figure 9. Transforming Extended Strands to Multiset Rewriting Rules

Proof: In the forward direction, we add the labels as from the definition (they do not constrain the construction in any way); every use of transition $\mathbf{C_f}$ that introduces a new strand is mapped to $\mathbf{C_f}'$, which also installs the node $\top$. In the reverse direction, we simply forget about labels and extra nodes. Formally, both directions require a simple structural induction. □

5.2 Translation

Given the above definitions, we are in a position to propose an transition-preserving translation that maps strand representations of security protocols to the multiset rewriting formalism. We will proceed in stages: in Section 5.2.1 we concentrate on parametric strands, in Section 5.2.2 we relate the intruder models, in Section 5.2.3 we extract a notion of state from a configuration, and finally in Section 5.2.4 we prove the correctness of our translation.

5.2.1 From Parametric Strands to Roles

We now present a translation of parametric strands to the coordinated sets of transition rules representing a role. Each node is mapped to a rule, the label of its incoming and outgoing $\Longrightarrow$-edge will be the state predicates in the antecedent and consequent, respectively, and the network message will be the message embedded in the node, its polarity dictating on which side of the arrow it should be appear. More formally, we

have the translation displayed in Figure 9, where the parameters of the added state predicates are classified as in the above definition.

Given a set of (decorated) parametric strands $\overline{\mathcal{S}}$, we write $\ulcorner\overline{\mathcal{S}}\urcorner$ for the set of protocol rules resulting from this transformation. Observe that it yields regular rules. Applying this translation to the enhanced parametric strands representing the Needham-Schroeder protocol in Figure 8 produces exactly the original transition system specification given in Figure 1.

5.2.2 From Penetrator Strands to Intruder Theory

The translation of the penetrator strands $\mathcal{P}(P_0)$ in Figure 7 is essentially the inverse of the mapping discussed in Section 4.2.2. Our target intruder model, in the multiset rewriting world, is $\mathcal{I}'$.

$$\begin{array}{rclcrcl}
\ulcorner S(m_1,m_2)\urcorner & = & \mathrm{dcmp}'(m_1,m_2) & \quad & \ulcorner C(m_1,m_2)\urcorner & = & \mathrm{cmp}'(m_1,m_2) \\
\ulcorner D(m,k)\urcorner & = & \mathrm{decr}'(m,k) & & \ulcorner E(m,k)\urcorner & = & \mathrm{encr}'(m,k) \\
\ulcorner N(n)\urcorner & = & \mathrm{nnc}'(n) & & \ulcorner M(m)\urcorner & = & \mathrm{pers}'(m) \\
\ulcorner T(m)\urcorner & = & \mathrm{dup}(m) & & \ulcorner F(m)\urcorner & = & \mathrm{del}(m) \\
\ulcorner M'(m)\urcorner & = & \textit{(see below)} & &
\end{array}$$

where we have again equipped the intruder transition rules with the obvious arguments.

Notice that no penetrator strand is made to correspond to rules rec' or snd'. When translating transition sequences from the strand world to the transition system setting, we will insert these rules whenever a message sent by a principal's strand is received by a penetrator strand, and vice-versa, respectively. We map P_0 to a multiset $\boldsymbol{I}_0$ of messages initially known to the intruder in the multiset rewriting framework: $\ulcorner P_0\urcorner = \wr \mathsf{I}(m) : m \in P_0 \wr$. Uses of $M'(m)$ with $m \in P_0$ are translated to accesses to $\mathsf{I}(m) \in \ulcorner P_0\urcorner$, possibly preceded by an application of rule dup if $M'(m)$ is accessed more than once.

5.2.3 From Configurations to States

Before we can show that the translation we just outlined preserves transition sequences, we need to extract a state from a configuration and show that steps between configurations are mapped to steps between the corresponding states.

Let $\mathcal{S}$ be a set of parametric strands, $\ulcorner\overline{\mathcal{S}}\urcorner$ its translation as a set of transition rules, and $(\sigma, \sigma^\sharp)$ a configuration over $\mathcal{S}, \mathcal{P}(P_0)$ where all penetrator strands have been completed. We define the *state associated with* $(\bar\sigma, \bar\sigma^\sharp)$, written $S_{\overline{\mathcal{S}}}(\bar\sigma, \bar\sigma^\sharp)$, as the state $(\boldsymbol{\Pi}, \boldsymbol{A}, \boldsymbol{N}, \boldsymbol{I})$ obtained as follows, where we write $\wr \ldots \wr$ for the multiset equivalent of the usual set notation $\{\ldots\}$:

- $\boldsymbol{N} = \wr \mathsf{N}(m) : \nu \in \mathrm{Fr}(\bar\sigma), \nu$ is not on a penetrator strand, and ν has label $+m \wr$.
- $\boldsymbol{I} = \wr \mathsf{I}(m) : \nu \in \mathrm{Fr}(\bar\sigma), \nu$ is on a penetrator strand,and ν has label $+m \wr$.
- $\boldsymbol{A} = \wr \mathsf{A}_{si}(\vec{t}) : s_{i-1} \overset{\mathsf{A}_{si}(\vec{t})}{\Longrightarrow} s_i \in \bar\sigma^\sharp \setminus \bar\sigma$ and $s_{i-1} \in \mathrm{Fr}(\sigma) \wr$.

Intuitively, we collect the messages in transit coming from honest principal's strands in $\boldsymbol{N}$, the current knowledge of the intruder in $\boldsymbol{I}$, and the labels of the $\Longrightarrow$-edges at the boundary between $\bar\sigma^\sharp$ and $\bar\sigma$ as the multiset of role state predicates $\boldsymbol{A}$.

5.2.4 From Move to Transition Sequences

Then, sequences of moves in the strand world and their translation as transition system steps are related as follows:

Theorem 5.2 *Let P_0 be some initial penetrator knowledge, and $\ulcorner P_0 \urcorner$ its multiset translation as defined in Section 5.2.2. Let $(\sigma_1, \sigma_1^\sharp)$ and $(\sigma_2, \sigma_2^\sharp)$ be two configurations on the penetrator strands $\mathcal{P}(P_0)$ and a set of parametric strands $\mathcal{S}$ such that all penetrator strands have been completed. For every multistep strand transition*

$$(\sigma_1, \sigma_1^\sharp) \longmapsto^{\vec{o}}_{\mathcal{P}(P_0),\mathcal{S}}{}^{*} (\sigma_2, \sigma_2^\sharp),$$

and every $I_0' \subseteq \ulcorner P_0 \urcorner$, there exists a regular multiset transition sequence $\vec{r}$ such that

$$\ulcorner P_0 \urcorner, S_{\overline{\mathcal{S}}}(\bar{\sigma}_1, \bar{\sigma}_1^\sharp) \longrightarrow^{\vec{r}}_{\mathcal{I}', \ulcorner \overline{\mathcal{S}} \urcorner}{}^{*} S_{\overline{\mathcal{S}}}(\bar{\sigma}_2, \bar{\sigma}_2^\sharp), I_0'.$$

Proof: The proof of this result proceeds by induction on the structure of $\vec{o}$. The only non-obvious aspect is that, as observed in Section 5.2.2, we need to insert applications of the rule rec' when processing a message that flows from an honest principal's to a penetrator strands. We add uses of snd' in the dual case. □

Notice that we do not need to start from the empty configuration.

The mapping from strands to multiset rewriting we have just finished outlining, and the translation from multiset rewriting to strand constructions described in Section 4 are inverse of each other. We leave the proof of this fact to the interested reader.

6 Related Work

Differences in models for distributed system have captivated the curiosity of researchers for over two decades. It was observed that they tend to fall into two paradigms: state-based languages such as Petri nets and multiset rewriting cleanly separate the factual description of the world (as a marking or a state, resp.) and the transformations that modify it (as transitions or rules, resp.); process-based languages such as the various process algebras and here strand spaces blur the distinction in favor of the self-transforming notion of a process. An early attempt to provide a Petri net semantics to CCS can be found in [12], while a reverse mapping first appeared in [4]. First-order formalisms were considered only several years later in the classical work of Berry and Boudol [2], whose state-based formalism, however, is in many ways closer to a process algebra than to multiset rewriting. Most subsequent research on the subject has been semantic in nature [30] or has investigated specialized sublanguages [6, 18]. We were unable to take direct advantage of these and other papers in the literature in our attempt to define simple syntactic translations between state-based and process-based languages that could be instantiated to the security setting described here.

To the best of our knowledge, the first investigation of the relationship among specification languages for security protocols appeared in [9]. The present paper completes that work with a layer of detail previously omitted for editorial reason and by connecting it to the body of research dedicated to relating languages for distributed systems, cryptographic protocols in particular. As observed in the introduction, numerous authors have explored the problem of connecting the numerous languages for security protocol analysis proposed in recent years.

The work of Crazzolara and Winskel [11] bears clear similarities to the present investigation: they start with SPL, a simple process-oriented language akin to the spi-calculus [1] and map it to several other formalisms, which include a form of contextual Petri nets, strand spaces, and Paulson's inductive method [29]. Once a few idiosyncracies have been ironed out, the relationship to strand spaces is relatively simple as both have a process-based semantics. The translation to Petri nets, on the other hand, is closely related to our mapping of strand spaces to multiset rewriting in Section 5 although it does not go into the same level of detail. The reverse mapping is not discussed.

An abstract investigation of the relationship between state-based and process-based specification languages, with particular emphasis on formalisms for expressing security protocols, is initiated in [3], which

is a preludes to general results linking the two paradigms. This work is directly inspired by recent research which places linear logic at the crossroad between the two paradigms [8, 27].

Additional cross-language investigations are found in [21], which notes similarities between strand spaces and multi-agent systems and proposes translations in both directions, as well as extensions to the strands space formalism. In [22], Heather unveils strong links between strand spaces and rank functions. In [24], Meadows surveys these two approaches and several more for their specific use of invariants. A different target is considered in [32], which uses strand spaces to define a semantics for BAN-like logics. Different yet is [19], which adapts strand spaces to relate the concrete cryptography found in protocol implementations and its abstraction in the Dolev-Yao model.

Following [9], several authors have made use of a dynamic semantics for strand space inspired to our parametric strands. Millen and Shmatikov [26] base their innovative constraint-solving analysis method for cryptographic protocols on a form of parametric strand without prescriptive freshness. The MITRE group, which pioneered strand spaces, embraced the related notion of *schematic* strands in [19]. A interpretation of parametric strand in linear logic was given in [8]. Further variants on the notion of parametric strands also appeared in [11, 17]

All of the above theoretical investigations allow researchers to understand precisely how their results are related, often enabling a direct transfer of properties such as secrecy and many forms of authentication as most of these formalisms ultimately rely on a trace-based semantics. This observation was put into practice in the CAPSL Intermediate Language (CIL — another close relative of multiset rewriting) [14] and the numerous "connectors" translating CIL specifications to and from other languages and tools [5, 13, 25].

7 Conclusions and Future Work

We have revised the formal connection between multiset rewriting [10, 16] and strand constructions [33, 31] previously outlined in [9]. In particular, we situated it relative to the recent body of work aimed at relating languages for describing security protocols. The formalization required a number of unexpected adjustments to both frameworks. In particular, we equipped strands with a dynamic dimension by introducing a notion of transition that allows growing bundles from a set of parametric strands. This enabled us to relate the distinct notions of traces inherent to these formalisms: bundles and multiset rewrite sequences.

This work did not attempt any connection between the various verification methodologies that have been successfully used in conjunction with multiset rewriting and strand spaces (or closely related languages). There are two reasons for this. First the multiset rewriting formalism was designed to be independent from any particular analysis technique, and indeed a number of proposals have been successfully applied to it. Second, such a meta-theoretic investigation is simply beyond the scope of this paper. However, it should be noted that the results here can be directly used to port any trace-based property established for one formalism to the other. This includes secrecy and most forms of authentication.

Acknowledgments

We would like to thank Joshua Guttman, Javier Thayer Fábrega, Jonathan Herzog, and Al Maneki for the stimulating discussions about strands. We are also indebted to Sylvan Pinsky for his encouragements to write down our ideas about the relationship between strand construction and our protocol theories. Finally, this work profitted from fruitful discussions with Jon Millen, Cathy Meadows, Paul Syverson, and the comments of the anonymous reviewers.

References

[1] M. Abadi and A. Gordon. A calculus for cryptographic protocols: the spi calculus. *Information and Computation*, 148(1):1–70, 1999.

[2] G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96(1):217–248, 1992.

[3] S. Bistarelli, I. Cervesato, G. Lenzini, and F. Martinelli. Relating Process Algebras and Multiset Rewriting for Immediate Decryption Protocols. In V. Gorodetski, V. Skormin, and L. Popyack, editors, *Second International Workshop on Mathematical Methods, Models and Architectures for Computer Networks Security — MMM'03*, pages 86–99, St. Petersburg, Russia, 20–24 September 2003. Springer-Verlag LNAI 2776.

[4] G. Boudol, G. Roucairol, and R. de Simone. Petri nets and algebraic calculi of processes. *Advances in Petri Nets 1985*, pages 41–58, 1986.

[5] S. Brackin, C. Meadows, and J. Millen. CAPSL interface for the NRL protocol analyzer. In *2nd IEEE Workshop on Application-Specifi c Software Engineering and Technology (ASSET '99)*. IEEE Computer Society, 1999.

[6] N. Busi and R. Gorrieri. A Petri net semantics for pi-calculus. In *Proc. of the 6th International Conference on Concurrency Theory — CONCUR'95*, pages 145–159, Philadelphia, PA, 1995.

[7] I. Cervesato. Typed MSR: Syntax and Examples. In V. Gorodetski, V. Skormin, and L. Popyack, editors, *First International Workshop on Mathematical Methods, Models and Architectures for Computer Networks Security — MMM'01*, pages 159–177, St. Petersburg, Russia, May 2001. Springer-Verlag LNCS 2052.

[8] I. Cervesato, N. Durgin, M. I. Kanovich, and A. Scedrov. Interpreting Strands in Linear Logic. In H. Veith, N. Heintze, and E. Clark, editors, *2000 Workshop on Formal Methods and Computer Security — FMCS'00*, Chicago, IL, July 2000.

[9] I. Cervesato, N. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. A Comparison between Strand Spaces and Multiset Rewriting for Security Protocol Analysis. In M. Okada, B. Pierce, A. Scedrov, H. Tokuda, and A. Yonezawa, editors, *Software Security: Theories and Systems*, pages 356–383, Tokyo, Japan, 2003. Springer-Verlag LNCS 2609.

[10] I. Cervesato, N. A. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. In P. Syverson, editor, *Proceedings of the 12th IEEE Computer Security Foundations Workshop — CSFW'99*, pages 55–69, Mordano, Italy, June 1999. IEEE Computer Society Press.

[11] F. Crazzolara and G. Winskel. Events in security protocols. In P. Samarati, editor, *Proceedings of the 8th ACM Conference on Computer and Communications Security*, pages 96–105, Philadelphia, PA, USA, Nov. 2001. ACM Press.

[12] F. de Cindio, G. de Michelis, L. Pomello, and C. Simone. Milner's communicating systems and Petri nets. In Pagnoni, A. and Rozenberg, G., editors, *Informatik-Fachberichte 66: Application and Theory of Petri Nets — Selected Papers from the Third European Workshop on Application and Theory of Petri Nets, Varenna, Italy, September 27–30, 1982*, pages 40–59. Springer-Verlag, 1983.

[13] G. Denker. Design of a CIL connector to Maude. In H. V. andN. Heintze and E. Clarke, editors, *Workshop on Formal Methods and Computer Security*. Carnegie Mellon University, July 2000.

[14] G. Denker and J. K. Millen. CAPSL Intermediate Language. In N. Heintze and E. Clarke, editors, *Proceedings of the Workshop on Formal Methods and Security Protocols — FMSP*, Trento, Italy, July 1999.

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

[16] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Multiset rewriting and the complexity of bounded security protocols. *Journal of Computer Security*, 12:247–311, 2004.

[17] N. Durgin, J. C. Mitchell, and D. Pavlovic. A compositional logic for protocol correctness. In *14th IEEE Computer Security Foundations Workshop — CSFW'01*, pages 241–255, Cape Breton, Canada, 11–13 June 2001. IEEE Computer Society Press.

[18] J. Engelfriet and T. Gelsema. Multisets and structural congruence of the π-calculus with replication. *Theor. Comput. Sci.*, 211(1-2):311–337, 1999.

[19] J. Guttman, J. Thayer Fábrega, and L. Zuck. The faithfulness of abstract protocol analysis: message authentication. In P. Samarati, editor, *Proceedings of the 8th ACM Conference on Computer and Communications Security*, pages 186–195, Philadelphia, PA, USA, Nov. 2001. ACM Press.

[20] J. D. Guttman and F. J. T. Fábrega. Authentication tests and the structure of bundles. *Theoretical Computer Science*, 283(2):333–380, 2002.

[21] J. Y. Halpern and R. Pucella. On the relationship between strand spaces and multi-agent systems. In P. Samarati, editor, *Proceedings of the 8th ACM Conference on Computer and Communications Security*, pages 106–115, Philadelphia, PA, USA, Nov. 2001. ACM Press.

[22] J. Heather. Strand spaces and rank functions: More than distant cousins. In *15th IEEE Computer Security Foundations Workshop — CSFW'01*, pages 104–116, Cape Breton, Canada, 24–26 June 2002. IEEE Computer Society Press.

[23] A. Maneki. Honest functions and their application to the analysis of cryptographic protocols. In P. Syverson, editor, *Proceedings of the 12th IEEE Computer Security Foundations Workshop — CSFW'99*, pages 83–89, Mordano, Italy, June 1999. IEEE Computer Society Press.

[24] C. Meadows. Invariant generation techniques in cryptographic protocol analysis. In *13th IEEE Computer Security Foundations Workshop — CSFW'00*, pages 159–169, Cambridge, UK, 3–5 July 2000. IEEE Computer Society Press.

[25] J. Millen. A CAPSL connector to Athena. In H. Veith, N. Heintze, and E. Clarke, editors, *Workshop of Formal Methods and Computer Security*. CAV, 2000.

[26] J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In P. Samarati, editor, *Proceedings of the 8th ACM Conference on Computer and Communications Security*, pages 166–175, Philadelphia, PA, USA, Nov. 2001. ACM Press.

[27] D. Miller. Encryption as an abstract data-type: An extended abstract. In I. Cervesato, editor, *Proceedings of the 2nd Workshop on Foundations of Computer Security — FCS'03*, pages 3–14, Ottawa, Canada, 26–27 June 2003.

[28] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

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

[30] V. Sassone, M. Nielsen, and G. Winskel. Models for concurrency: Towards a classifi cation. *Theoretical Computer Science*, 170(1–2):297–348, 1996.

[31] D. Song. Athena: a new effi cient automatic checker for security protocol analysis. In *Proceedings of the Twelth IEEE Computer Security Foundations Workshop*, pages 192–202, Mordano, Italy, June 1999. IEEE Computer Society Press.

[32] P. Syverson. Towards a strand semantics for authentication logic. In S. Brookes, A. Jung, M. Mislove, and A. Scedrov, editors, *Electronic Notes in Theoretical Computer Science*, volume 20. Elsevier, 2000.

[33] J. Thayer Fábrega, J. Herzog, and J. Guttman. Strand spaces: Why is a security protocol correct? In *Proceedings of the 1998 IEEE Symposium on Security and Privacy*, pages 160–171, Oakland, CA, May 1998. IEEE Computer Society Press.

[34] J. Thayer Fábrega, J. Herzog, and J. Guttman. Mixed strand spaces. In P. Syverson, editor, *Proceedings of the 12th IEEE Computer Security Foundations Workshop — CSFW'99*, pages 72–82, Mordano, Italy, June 1999. IEEE Computer Society Press.

A Protocol Signatures

In this appendix, we give a formal description of the syntax of the various entities used in this paper. We also characterize the persistent information we have been relying upon with respect to this language.

A.1 Language

First, we rigorously define the language of terms and the predicate symbols we have been using in the body of this paper in order to specify a protocol in the multiset notation. It is meant to be extensible if the need arises, therefore it should not be considered complete. Clearly, this is only one of the many possible formalizations, not necessarily the best one. This attempt goes along the lines of [16], although it is not identical to it.

Sorts

We first declare a number of sorts to classify the various entities we are working with.

msg			:	sort	(*Messages*)
principal	::	msg	:	sort	(*Principals*)
key	::	msg	:	sort	(*Keys*)
nonce	::	key	:	sort	(*Nonces*)
text	::	msg	:	sort	(*Data*)

We rely on subsorting for simplicity (written "::"). Since nonces are often used as session keys, we declare nonce to be a subsort of key. By transitivity, it is also a subsort of msg.

We write m for messages, k for keys, n for nonces, t for texts, and a, b, ... for principal names. We use the corresponding capital letters for variables of the same sorts. We generally indicate the owner(s) of a key with a subscript. For example, the public key of a could be k_a, a shared key between a and b could be indicated as k_{ab}. We sometimes write k_a^{-1} for the inverse of key k_a.

Whenever the effect of the protocol is to transmit information that is not generated within the protocol itself (*e.g.* a file or a credit card number), we tag it with sort text. This is slightly more precise than classifying it as a generic msg.

Messages

Complex messages themselves are constructed from simpler messages by means of the operations of encryption and composition, declared below. Other operations, *e.g.* hashing, are declared similarly if the need arises.

$\{\_\}\_$	:	$\mathsf{msg} \times \mathsf{key} \rightarrow \mathsf{msg}$	(*Encryption*)
$(\_,\_)$	:	$\mathsf{msg} \times \mathsf{msg} \rightarrow \mathsf{msg}$	(*Composition*)

We do not include explicit decomposition and decryption operators as these functionalities are achieved by pattern matching on terms mentioning the above constructs. This simplification is adequate for our modeling purposes. Furthermore, it avoids the introduction of any non-trivial equational theory at the level of terms. Observe that this is not a limitation of our model, but a simplification: these operations could be included as primitives, and their equational theory operationally specified by means of rewrite rules.

As anticipated in Section 2, we often work with parametric messages, written $m(\vec{x})$, which differ from proper messages by the fact that some submessages can be variables among $\vec{x} = \{x_1, \ldots, x_v\}$. Some of the x_i may not occur in m. Instantiating a parametric message $m(\vec{x})$ with messages $\vec{t} = \{t_1, \ldots, t_v\}$ matching (or specializing) the sorts of $\vec{x}$ is written $m(\vec{t})$. We write $\vec{t}/\vec{x}$ for the corresponding substitution, call it δ. We denote the application of a substitution δ to a parametric message m as $m[\delta]$.

Network

Information broadcast over the network has the form $\mathsf{N}(m)$, where N is declared as follows:

$\mathsf{N}(\_)$	:	$\mathsf{msg} \rightarrow \underline{\mathsf{atom}}$	(*Message in transit*)

An equivalent, but not as general, specification of network messages establishes a distinct predicate for each legal message class that can be exchanged during a run of a protocol.

Roles

We declare the sort role to classify role names, used, for example, to distinguish the initiator and the responder of the Needham-Schroeder protocols.

$$\begin{array}{lcll} \mathsf{role} & : & \underline{\mathsf{sort}} & (\textit{Role}) \\ \mathsf{A}_{\rho l}(\_,\ldots,\_) & : & \mathsf{principal} \times \tau^* \to \underline{\mathsf{atom}} & \\ & & \text{with } \rho : \mathsf{role},\ l \in L,\ \text{and } \tau :: \mathsf{msg} & \end{array}$$

The predicates $\mathsf{A}_{\rho l}(a, \vec{m})$ are intended to hold the local data $\vec{m}$ of a principal a in role ρ during the run of the successive steps of the protocol. Such data typically includes a's keys, the identity and public/shared keys of the message recipients (other principals or some server), nonces, external messages, etc.

We shall assume that for each role ρ, there are finitely many state predicates $\mathsf{A}_{\rho l}$, where l is a label in a partially ordered set L. In most cases, the indices l are successive numbers ($l \in \mathbb{N}$ and $l = 0 \ldots l_\rho$). We give a more general definition to accommodate roles that can take conditional or non-deterministic actions.

Intruder knowledge

The predicate symbol I is used to hold the knowledge of the intruder in a distributed way.

$$\mathsf{I}(\_) \quad : \quad \mathsf{msg} \to \underline{\mathsf{atom}}$$

Initialization

Finally, we have a number of predicates intended to organize the static information known to and about the various parties in a protocol. This list is by no means exhaustive.

$$\begin{array}{ll} \mathit{Pr}(\_) & : \mathsf{principal} \to \underline{\mathsf{atom}} \\ \mathit{Foe}(\_) & : \mathsf{principal} \to \underline{\mathsf{atom}} \\ \mathit{PubK}(\_,\_) & : \mathsf{principal} \times \mathsf{key} \to \underline{\mathsf{atom}} \\ \mathit{PrvK}(\_,\_) & : \mathsf{principal} \times \mathsf{key} \to \underline{\mathsf{atom}} \\ \mathit{ShK}(\_,\_,\_) & : \mathsf{principal} \times \mathsf{principal} \times \mathsf{key} \to \underline{\mathsf{atom}} \\ \mathit{KeyP}(\_,\_) & : \mathsf{key} \times \mathsf{key} \to \underline{\mathsf{atom}} \\ \mathit{Txt}(\_,\_) & : \mathsf{principal} \times \mathsf{text} \to \underline{\mathsf{atom}} \end{array}$$

As a convention, we write initialization predicates in a slanted font to distinguish them from other symbols. The information they hold is expected not to change during the execution of a protocol.

Pr declares the known principals. It is actually redundant as the same information is conveyed by typing. *Foe* identifies the subset of the principals that are in league with the intruder (this could also be obtained via subsorting). *PubK* and *PrvK* are intended to relate a principal and its public and private keys, respectively. *ShK* indicates what keys are shared between which principals. *KeyP* records which pairs of keys are inverse of each other. Finally, *Txt* contains some piece of text that a principal may know before any run of the protocol takes place (and wants to transmit it).

We collect these declarations in a *signature* that we call Σ. We generally keep Σ implicit. We assume that all the expressions we are considering are well-typed according to the contents of Σ.

A.2 Persistent Information

We discussed how to specify a protocol (given a signature Σ) in the body of this paper (Section 2.2). We complete this presentation by making more precise the contents of the initialization theory $\mathbf{\Pi}$.

As we said, some information is given prior to any run of a protocol, and does not change during its execution. This include the principals that are allowed to take part into it including the dishonest ones (we may assume the intruder cannot bribe an otherwise honest principal during the execution), their keys unless the protocol models key distribution, the identity and public key of the servers, etc. We store this persistent information as a set of *ground facts* that we call $\mathbf{\Pi}$. Given the declarations in the previous section, it contains some number of facts of the following form:

- $\mathsf{Pr}(a)$.
- $\mathsf{Foe}(a)$.
- $\mathsf{ShK}(a, b, k_{ab})$.
- $\mathsf{PubK}(a, k_a)$.
- $\mathsf{PrvK}(a, k_a^{-1})$.
- $\mathsf{KeyP}(k, k^{-1})$.
- $\mathsf{Txt}(a, t)$.

None of the other predicates declared in the previous section may appear in $\mathbf{\Pi}$. For any principals a and b, and keys k and k', we make the following assumptions on $\mathbf{\Pi}$:

1. if $\mathsf{PubK}(a, k) \in \mathbf{\Pi}$, then there is k^{-1} such that $\mathsf{PrvK}(a, k^{-1}) \in \mathbf{\Pi}$ and $\mathsf{KeyP}(k, k^{-1}) \in \mathbf{\Pi}$.
2. if $\mathsf{ShK}(a, b, k) \in \mathbf{\Pi}$, then $\mathsf{ShK}(b, a, k) \in \mathbf{\Pi}$ and $\mathsf{KeyP}(k, k) \in \mathbf{\Pi}$.
3. if $\mathsf{KeyP}(k, k') \in \mathbf{\Pi}$, then $\mathsf{KeyP}(k', k) \in \mathbf{\Pi}$.

The set $\mathbf{\Pi}$ can be viewed as an *initialization theory*.

B Configurations vs. Move Sequences in Dynamic Strands

The definition of execution for parametric strands given in Section 3 embeds two distinct notions of traces for strand constructions. On the one hand, configurations give a precise account of which events have taken place, abstracting from their temporal occurrence order, but taking into consideration their dependencies both in terms of the ordering of steps (captured by $\Longrightarrow$-edges) and message transmission/reception (expressed by the $\longrightarrow$-arrows). On the other hand, the move sequence $\vec{o}$ that labels the transition arrow also indicates which steps have taken place, but imposes a linear occurrence order on them. We will now relate these two notions.

B.1 From Configurations to Move Sequences

Notice that, with the exception of $\mathbf{C_f}$ and $\mathbf{C_i}$, each move inserts exactly one node in a configuration. Moreover, the very possibility of making such an insertion is regulated by the two types of edges. Therefore, we can think of a bundle as specifying a partial order of the occurrence of individual moves (the ordering relation is the transitive closure of the union of $\Longrightarrow$ and $\longrightarrow$). Instead, a move sequence linearizes the set of moves into a total order. In general, we can linearize a configuration $(\sigma, \sigma^\sharp)$ as a sequence of moves in many ways. The following definition imposes constraints on the form of acceptable move sequences.

Given a configuration $(\sigma, \sigma^\sharp)$ with $\sigma = (S, \Longrightarrow, \longrightarrow)$ and $\sigma^\sharp = (S^\sharp, \Longrightarrow^\sharp, \longrightarrow^\sharp)$, we define $O_{(\sigma,\sigma^\sharp)}$ as the set of move sequences $\vec{o} = (o_1, \ldots, o_n)$ such that, for $i = 1, \ldots, n$:

(a) $n = n^{\mathbf{So}} + n^{\mathbf{Ro}} + n^{\mathbf{S}} + n^{\mathbf{R}} + n^{\mathbf{Cf}} + n^{\mathbf{Ci}}$, where $n^{\mathbf{So}}$ and $n^{\mathbf{Ro}}$ are respectively the number of initial sending and receiving nodes in σ, $n^{\mathbf{S}}$ and $n^{\mathbf{R}}$ are the number of non-initial sending and receiving nodes in σ respectively, $n^{\mathbf{Cf}}$ is the total number of strands (possibly only partially instantiated) in $\sigma^\sharp$, and $n^{\mathbf{Ci}}$ is the number of fully instantiated strands among these. Moreover, $\vec{o}$ contains exactly n^τ move of each type τ above. Observe that $|S| = n^{\mathbf{So}} + n^{\mathbf{Ro}} + n^{\mathbf{S}} + n^{\mathbf{R}}$.

(b) For each $o_i = (\nu_i, \bar{\nu}_i^p, \bar{\nu}_i^s)$, we have that $\nu_i \in S$. Moreover, if $o_j = (\nu_j, \bar{\nu}_j^p, \bar{\nu}_j^s)$, then $\nu_i \neq \nu_j$ for $i \neq j$.

(b') For each $\nu \in S$, there is an index i such that $o_i = (\nu, \bar{\nu}_i^p, \bar{\nu}_i^s)$.

(c) If $o_i = (\nu_i, -, \bar{\nu}_i^s)$, then ν_i is initial in S, and there is a unique index $j < i$ such that $o_j = (\nu_j, \theta_j)$ and $\nu_i = \nu_j[\theta_j]$.

(c') For every initial node ν in σ, there is a unique index i such that $o_i = (\nu, -, \bar{\nu}_i^s)$.

(d) If $o_i = (\nu_i, \nu_i^p, \bar{\nu}_i^s)$, then there is a unique index $j < i$ with $o_j = (\nu_j, \bar{\nu}_j^p, \bar{\nu}_j^s)$ such that $\nu_j \Longrightarrow \nu_i$ is in σ and $\nu_i^p = \nu_j$.

(d') For every non-initial node ν in σ with parent ν^p, there is a unique index i such that $o_i = (\nu, \nu^p, \bar{\nu}_i^s)$.

(e) If $o_i = (\nu_i, \bar{\nu}_i^p, -)$, then there is a unique index $j < i$ with $o_j = (\nu_j, \bar{\nu}_j^p, \nu_j^s)$ such that $\nu_j \longrightarrow \nu_i$ in σ and $\nu_j^s = \nu_i$.

(e') For every receiving node ν in σ with sender ν^s, there is a unique index i such that $o_i = (\nu, \bar{\nu}^p, -)$.

(f) If $o_i = (\nu_i, \bar{\nu}_i^p, \nu_i^s)$, then $\nu_i^s \in S^\sharp$ and there are unique indices $k < j < i$ with $o_k = (\rho^k, \xi_k)$ and $o_j = (\nu_j, \theta_j)$ such that $\rho_0^k[\xi_k] = \nu_j$ and there is an index l such that $\rho_l^k[\xi_k, \theta_j] = \nu_i^s$ and $\nu_i \longrightarrow^\sharp \nu_i^s$ in $\sigma^\sharp$.

(f') For every sending node ν in σ with receiver ν^s, there is a unique index i such that $o_i = (\nu, \bar{\nu}^p, \nu^s)$.

(g) If $o_i = (\nu_i, \theta_i)$, then there a unique index $j < i$ such that $o_j = (\rho^j, \xi_j)$ and $\rho_0^j[\xi_j] = \nu_i$, and there is a fully instantiated strand ρ in $\sigma^\sharp$ such that $\rho = \rho^j[\xi_j, \theta_i]$.

(g') For every fully instantiated strand ρ in $\sigma^\sharp$, there are unique indices $i < j$ with $o_i = (\rho^i, \xi_i)$ and $o_j = (\nu_j, \theta_j)$, such that $\rho = \rho^i[\xi_i, \theta_j]$ and $\nu_j = \rho_0^i[\xi_i]$.

(h) If $o_i = (\rho^i, \xi_i)$, then either $\rho^i[\xi_i]$ is a partially instantiated strand in $\sigma^\sharp$, or there is a unique index $j > i$ such that $o_j = (\nu_j, \theta_j)$, and $\rho_0^i[\xi_i] = \nu_j$ (and $\rho^j[\xi_j, \theta_i]$ is a fully instantiated strand in $\sigma^\sharp$).

(h') For every partially instantiated strand ρ in $\sigma^\sharp$, there is a unique index i such $o_i = (\rho^i, \xi_i)$ and $\rho = \rho^i[\xi_i]$.

The left-hand side column specifies sufficient and mostly internal coherence conditions so that a move sequence belongs to $O_{(\sigma,\sigma^\sharp)}$, while the right-hand side column gives the corresponding necessary conditions.

Any legal move sequence $\vec{o}$ from $(\cdot, \cdot)$ to any configuration $(\sigma, \sigma^\sharp)$ is an element of $O_{(\sigma,\sigma^\sharp)}$. This is formalized in the following completeness result.

Property B.1

Let $(\sigma, \sigma^\sharp)$ be a configuration over a set $\mathcal{S}$ of parametric strands and $\vec{o}$ a move sequence such that $(\cdot, \cdot) \stackrel{\vec{o}}{\longmapsto}{}^*_{\mathcal{S}} (\sigma, \sigma^\sharp)$. Then $\vec{o} \in O_{(\sigma,\sigma^\sharp)}$.

Proof:

The proof proceeds by induction on the length of the move sequence $\vec{o}$, checking that each element in it satisfies the above definition.

The base case, where $\vec{o} = \cdot$, trivially satisfies the property.

Assume then that $(\cdot,\cdot) \stackrel{\vec{o}}{\longmapsto}_{\mathcal{S}}^{*} (\hat{\sigma}, \hat{\sigma}^\sharp)$ with $\vec{o} = (o_1, \ldots, o_n) \in O_{(\hat{\sigma},\hat{\sigma}^\sharp)}$, and that $(\hat{\sigma}, \hat{\sigma}^\sharp) \stackrel{o_{n+1}}{\longmapsto}_{\mathcal{S}} (\sigma, \sigma^\sharp)$. We will then show that for every such move o_{n+1}, it happens that $(\vec{o}, o_{n+1}) \in O_{(\sigma,\sigma^\sharp)}$. We proceed by cases.

$\mathbf{S_0}$: Let $o_{n+1} = (\nu, -, \nu'')$. We need to examine the various conditions of the definition of $O_{(\sigma,\sigma^\sharp)}$.

- **(a)** By induction hypothesis $|\vec{o}| = \hat{n} = \hat{n}^{\mathbf{S_o}} + \hat{n}^{\mathbf{R_o}} + \hat{n}^{\mathbf{S}} + \hat{n}^{\mathbf{R}} + \hat{n}^{\mathbf{C_f}} + \hat{n}^{\mathbf{C_i}}$, moreover $\vec{o}$ contains exactly $\hat{n}^\tau$ moves of each type τ, and the constituents of $(\hat{\sigma}, \hat{\sigma}^\sharp)$ are related to these numbers as from the definition.

  Now, $n = |(\vec{o}, o_{n+1})| = \hat{n} + 1$. For $\tau \neq \mathbf{S_0}$, the number of moves of type τ in $(\vec{o}, o_{n+1})$ does not change, so that $n^\tau = \hat{n}^\tau$; instead $n^{\mathbf{S_o}} = \hat{n}^{\mathbf{S_o}} + 1$. Moreover, $(\sigma, \sigma^\sharp)$ differs from $(\hat{\sigma}, \hat{\sigma}^\sharp)$ only by the addition of a single initial sending node (ν) to the bundle part of the configuration (the definition of $O_{(\sigma,\sigma^\sharp)}$ is not directly concerned with arrows). This entails that condition **(a)** holds for $(\vec{o}, o_{n+1})$.

Since the first n elements of $(\vec{o}, o_{n+1})$ are unchanged with respect to $\vec{o}$, the induction hypothesis allows us to conclude that conditions **(b)**, **(c)**, **(d)**, **(e)**, **(f)**, **(g)**, and **(h)** still hold for these elements. We will therefore check that they hold also for o_{n+1}. Since this move has type $\mathbf{S_0}$, only conditions **(b)**, **(c)**, and **(f)** are applicable. We will check them in turn:

- **(b)** By definition of $\mathbf{S_0}$ transition, $\nu \in S$. Moreover, since ν is new in S (i.e. $\nu \in S$ but $\nu \notin \hat{S}$), there is clearly no index $i \leq n$ such that $o_i = (\nu_i, \bar{\nu}^p, \bar{\nu}^s)$ such that $\nu_i = \nu$.
- **(c)** Again by definition of $\mathbf{S_0}$ transition, ν is the initial node of a fully instantiated strand $\hat{\rho}$ in $\hat{\sigma}^\sharp$. Therefore, by condition **(g')** on $\vec{o}$, there are unique indices $i < j \leq n$ with $o_i = (\rho^i, \xi_i)$ and $o_j = (\nu_j, \theta_j)$, such that $\hat{\rho} = \rho^i[\xi_i, \theta_j]$ and $\nu_j = \rho_0^i[\xi_i]$. In particular, $\nu = \nu_j[\theta_j] = \rho_0^i[\xi_i, \theta_j] = \hat{\rho}_0$.
- **(f)** By definition of $\mathbf{S_0}$ transition, $\nu'' \in \hat{S}^\sharp \setminus \hat{S}$ and it lies on a fully instantiated strand $\hat{\rho}$. Again by condition **(g')** on $\vec{o}$, there are unique indices $i < j \leq n$ with $o_i = (\rho^i, \xi_i)$ and $o_j = (\nu_j, \theta_j)$, such that $\hat{\rho} = \rho^i[\xi_i, \theta_j]$ and $\nu_j = \rho_0^i[\xi_i]$. If ν'' has index l in $\hat{\rho}$, then $\nu'' = \rho_l^i[\xi_i, \theta_j]$. Moreover, again by definition of $\mathbf{S_0}$, we have that $\nu \longrightarrow^\sharp \nu''$ is in $\sigma^\sharp$.

Conditions **(b')**, **(c')**, **(d')**, **(e')**, **(f')**, **(g')** and **(h')** specify a family of mappings from nodes in the bundle part and strands in the parametric strand space part of the configuration to moves. By induction hypothesis, there exist such mappings from $(\hat{\sigma}, \hat{\sigma}^\sharp)$ to $\vec{o}$. Since $(\sigma, \sigma^\sharp)$ and $(\vec{o}, o_{n+1})$ can be seen as extensions of $(\hat{\sigma}, \hat{\sigma}^\sharp)$ and $\vec{o}$ respectively, we will simply extend these mappings by relating the additions to the configuration and the new element o_{n+1} of the move sequence. Because the added move has type $\mathbf{S_0}$, only conditions **(b')**, **(c')** and **(f')** apply.

- **(b')** We associate the index $n + 1$ to the added node ν. Again, the indices the other nodes in S are mapped to remain the same.
- **(c')** Again, we map ν to $n + 1$ and leave the rest unchanged.
- **(f')** Ditto.

S: Let $o_{n+1} = (\nu, \nu', \nu'')$. The proof proceeds as in the previous case, except that we need to examine conditions **(d)** and **(d')** instead of **(c)** and **(c')**. Condition **(d')** is handled similarly to the other primed cases above. We will develop the remaining condition:

(d) By definition of $\mathbf{S}$ transition, $\nu' \in \hat{S}$ and $\nu' \Longrightarrow \nu$ in σ. By induction hypothesis, condition **(b')**, there is a unique index $j \leq n$ such that $o_j = (\nu', \bar{\nu}_j^p, \bar{\nu}_j^s)$.

$\mathbf{R_0}$**:** Let $o_{n+1} = (\nu, -, -)$. The proof proceeds as in the previous cases, except that we need to examine conditions **(e)** and **(e')**. The latter is handled like the other primed cases above. We will focus on **(e)**:

(e) By definition of $\mathbf{R_0}$ transition, $\nu'' \in \hat{S}$ and $\nu'' \longrightarrow \nu$ in σ. By induction hypothesis, condition **(b')**, there is a unique index $j \leq n$ such that $o_j = (\nu'', \bar{\nu}_j^p, \bar{\nu}_j^s)$.

$\mathbf{R}$**:** In this case, $o_{n+1} = (\nu, \nu', -)$. The proof proceeds as in the previous cases, and we have examined all the relevant subcases.

$\mathbf{C_f}$**:** Let $o_{n+1} = (\rho, \xi)$. The considerations made in the previous cases can readily be extended to this situation. Condition **(a)** is treated similarly, the remaining non-primed conditions hold identically for the elements of $\vec{o}$, and the primed conditions are processed as above. We need however to check that condition **(h)** holds for o_{n+1} (it is the only non-primed condition applicable to a move of this kind).

(h) This condition can be fulfilled in two alternative ways: either by mapping the considered move to a partially instantiated strand in the configuration, or by showing that it is connected to a subsequent move of type $\mathbf{C_i}$. We must clearly adopt the first option and associate o_{n+1} to $\rho[\xi]$ in $\sigma^\sharp$.

$\mathbf{C_i}$**:** Let $o_{n+1} = (\nu_0, \theta)$. The verification of the unprimed conditions proceeds as above: in particular we can assume that conditions **(b)** through **(h)** hold for the elements of $\vec{o}$. We are then left with checking that **(g)** holds for o_{n+1} (no other unprimed condition applies).

(g) By definition of $\mathbf{C_i}$, $\hat{\sigma}^\sharp$ contains a partially instantiated strand $\hat{\rho}$ with initial node ν_0. By induction hypothesis and condition **(h')** on $\hat{\rho}$, there is a unique index $j \leq n$ such that $o_j = (\rho^j, \xi_j)$ and $\hat{\rho} = \rho^j[\xi_j]$. Clearly, $\nu_0 = \rho_0^j[\xi_j]$. Moreover, $\rho[\theta] = \rho^j[\xi_j, \theta]$ is a fully instantiated strand in $\sigma^\sharp$.

The verification of the primed conditions is more subtle than in the previous cases since moves of type $\mathbf{C_i}$ have the unique characteristic of not simply extending a configuration, but actually *replacing* a partially instantiated strand $\hat{\rho}$ with a ground instance $\hat{\rho}[\theta]$. Therefore, we not only need to map the newly inserted strand $\hat{\rho}[\theta]$ to some index (which will clearly be $n+1$), but also to upgrade the conditions associated with the elided $\hat{\rho}$.

By condition **(h')** on $\hat{\rho}$, there exists an index $i \leq n$ such that $o_i = (\rho^i, \xi_i)$ and $\hat{\rho} = \rho^i[\xi_i]$. Then, condition **(h)** applied to o_i in $\vec{o}$ because $\rho^i[\xi_i]$ belonged to $\hat{\sigma}^\sharp$. We must verify that it still applies in $(\vec{o}, o_{n+1})$ although $\rho^i[\xi_i]$ is not present in $\sigma^\sharp$. We fall back to the second alternative in the definition of **(h)**: $n+1$ is an index greater than i such that $o_{n+1} = (\nu_0, \theta)$ and, by construction, $\nu_0 = \rho_0^i[\xi_i]$. Clearly, this index is unique since the use of any other transition of type $\mathbf{C_i}$ would have instantiated ρ_i, making ν_0 unavailable.

This concludes the proof of this property. Most of the proofs below rely on similar techniques. Therefore, whenever this is the case, we will only present proof sketches and refer the reader to this detailed proof. □

Moreover, any $\vec{o}$ in $O_{(\sigma,\sigma^\sharp)}$ is a legal move sequence from $(\cdot, \cdot)$ to $(\sigma, \sigma^\sharp)$, as expressed by the following soundness result.

Property B.2

Let $(\sigma, \sigma^\sharp)$ be a configuration over a set $\mathcal{S}$ of parametric strands, then for each $\vec{o} \in O_{(\sigma,\sigma^\sharp)}$, the multistep transition $(\cdot,\cdot)\longmapsto^*_{\mathcal{S}}(\sigma,\sigma^\sharp)$ is well-defined.

Proof: We proceed by induction on the structure of the configuration. For this purpose, we need to define two well-orderings, both denoted with $\prec$, one over parametric strand spaces, and the other over configurations themselves.

Let $\sigma_1^\sharp$ and $\sigma_2^\sharp$ be two parametric strand spaces. We say that $\sigma_1^\sharp \prec \sigma_2^\sharp$ if either of the following condition holds:

- $\sigma_2^\sharp = \sigma_1^\sharp \cup \{\rho\}$ for a partially instantiated strand ρ.
- $\sigma_2^\sharp = \sigma_1^\sharp - \{\rho\} \cup \{\rho[\theta]\}$ for a partially instantiated strand ρ in $\sigma_1^\sharp$ and a substitution θ such that $\rho[\theta]$ is fully instantiated.
- there is a parametric strand space $\sigma^\sharp$ such that $\sigma_1^\sharp \prec \sigma^\sharp$ and $\sigma^\sharp \prec \sigma_2^\sharp$.

Let $(\sigma_1, \sigma_1^\sharp)$ and $(\sigma_2, \sigma_2^\sharp)$ be two configurations. We say that $(\sigma_1, \sigma_1^\sharp) \prec (\sigma_2, \sigma_2^\sharp)$ if either of the following conditions hold:

- σ_1 is a proper subgraph of σ_2 and $\sigma_1^\sharp = \sigma_2^\sharp$;
- $\sigma_1 = \sigma_2$ and $\sigma_1^\sharp \prec \sigma_2^\sharp$;
- there is a configuration $(\sigma, \sigma^\sharp)$ such that $(\sigma_1, \sigma_1^\sharp) \prec (\sigma, \sigma^\sharp)$ and $(\sigma, \sigma^\sharp) \prec (\sigma_2, \sigma_2^\sharp)$.

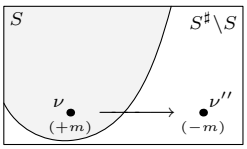
It is easy to show that this is indeed a well-ordering. Observe that its minimum is the empty configuration $(\cdot,\cdot)$.

In the base case, in which $(\sigma, \sigma^\sharp)$ is $(\cdot,\cdot)$, the property at hand holds trivially since only the empty move sequence is an element of $O_{(\cdot,\cdot)}$.

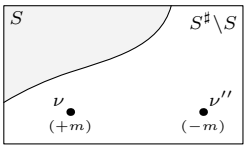
Let then assume that $\vec{o} = (o_1, \ldots, o_{n+1}) \in O_{(\sigma,\sigma^\sharp)}$. We will show that there is a configuration $(\hat{\sigma}, \hat{\sigma}^\sharp) \prec (\sigma, \sigma^\sharp)$ such that $(\hat{\sigma}, \hat{\sigma}^\sharp)\overset{o_{n+1}}{\longmapsto}_{\mathcal{S}}(\sigma, \sigma^\sharp)$ and $(o_1, \ldots, o_n) \in O_{(\hat{\sigma},\hat{\sigma}^\sharp)}$. It then follows, by induction hypothesis and the definition of multistep transition, that $(\cdot,\cdot)\longmapsto^*_{\mathcal{S}}(\sigma,\sigma^\sharp)$.

The determination of $(\hat{\sigma}, \hat{\sigma}^\sharp)$ proceeds by cases on the structure of o_{n+1}. For the sake of conciseness, we will only analyze the situation in which $o_{n+1} = (\nu, -, \nu'')$, i.e. it is intended to witness a transition of type $\mathbf{S_0}$.

$\mathbf{o_{n+1}} = (\nu, -, \nu'')$**:** By condition **(b)** of the definition of $O_{(\sigma,\sigma^\sharp)}$, we know that $\nu \in S$. By condition **(c)**, ν has no predecessor in σ. By condition **(f)**, $\nu'' \in S^\sharp \setminus S$ and $\nu \longrightarrow^\sharp \nu''$, from which we deduce that ν and ν'' are a sending and a receiving node, respectively. We are therefore in the following situation:



Since $(\sigma, \sigma^\sharp)$ is a configuration, the process of removing ν from S (but not from $S^\sharp$) and (ν, ν'') from $\longrightarrow^\sharp$ yields another configuration, displayed below. Call it $(\hat{\sigma}, \hat{\sigma}^\sharp)$. By definition of $\prec$, we have that $(\hat{\sigma}, \hat{\sigma}^\sharp) \prec (\sigma, \sigma^\sharp)$.



A transition of type $\mathbf{S_0}$ with ν as the sending node and ν'' as the receiving node is then enabled in $(\hat{\sigma}, \hat{\sigma}^\sharp)$. The corresponding move is precisely o_{n+1}, and the resulting transition is $(\sigma, \sigma^\sharp)$. Indeed we have that $(\hat{\sigma}, \hat{\sigma}^\sharp) \overset{o_{n+1}}{\longmapsto}_{\mathcal{S}} (\sigma, \sigma^\sharp)$.

The other cases are treated similarly. □

B.2 From Move Sequences to Configurations

If $\vec{o}$ describes the transition from $(\cdot, \cdot)$ to a configuration $(\sigma, \sigma^\sharp)$, the individual moves in $\vec{o}$ contain enough information to playback the sequence of moves and exactly reconstruct $(\sigma, \sigma^\sharp)$. This is done as follows.

Given a sequence of moves $\vec{o} = (o_1, \ldots, o_l)$, we define the *configuration associated with* $\vec{o}$, written $(\sigma_{\vec{o}}, \sigma^\sharp_{\vec{o}})$, as the triples $(S_{\vec{o}}, \Longrightarrow_{\vec{o}}, \longrightarrow_{\vec{o}})$ and $(S^\sharp_{\vec{o}}, \Longrightarrow^\sharp_{\vec{o}}, \longrightarrow^\sharp_{\vec{o}})$ given as follows:

- $(S^\sharp_{\vec{o}}, \Longrightarrow^\sharp_{\vec{o}}) = \{\rho^i[\xi_i, \theta_j] \ : \ (\rho^i, \xi_i) \in \vec{o}, (\nu_j, \theta_j) \in \vec{o}, \text{ and } \nu_j = \rho^i_0[\xi_i]\}$
  $\cup \{\rho^i[\xi_i] \ : \ (\rho^i, \xi_i) \in \vec{o}$ and there is no $(\nu_j, \theta_j) \in \vec{o}$
  such that $\nu_j = \rho^i_0[\xi_i]\}$.
- $\longrightarrow^\sharp_{\vec{o}} = \{(\nu_i, \nu^s_i) \ : \ (\nu_i, \bar{\nu}^p_i, \nu^s_i) \in \vec{o}\}$.
- $S_{\vec{o}} = \{\nu_i \ : \ (\nu_i, \bar{\nu}^p_i, \bar{\nu}^s_i) \in \vec{o}\}$.
- $\Longrightarrow_{\vec{o}} = (\Longrightarrow^\sharp_{\vec{o}})_{|S_{\vec{o}}} = \{(\nu_i, \nu^p_i) \ : \ (\nu_i, \nu^p_i, \bar{\nu}^s_i) \in \vec{o}\}$.
- $\longrightarrow_{\vec{o}} = (\longrightarrow^\sharp_{\vec{o}})_{|S_{\vec{o}}} = \{(\nu_i, \nu^s_i) \ : \ (\nu_i, \bar{\nu}^p_i, \bar{\nu}^s_i) \in \vec{o}$ and $(n^s_i, \bar{\nu}^p_j, -) \in \vec{o}\}$.

where $R_{|S}$ is the subrelation of R that only contains edges with extremes in S. It is easy to prove that, in the last two cases, the alternate definitions are equivalent.

Now, if $\vec{o}$ labels a transition from $(\cdot, \cdot)$ to some configuration $(\sigma, \sigma^\sharp)$, then $(\sigma_{\vec{o}}, \sigma^\sharp_{\vec{o}})$ is isomorphic to $(\sigma, \sigma^\sharp)$. We have the following expected result.

Property B.3
Let $(\sigma, \sigma^\sharp)$ be a configuration and $\vec{o}$ a move sequence such that $(\cdot, \cdot) \overset{\vec{o}}{\longmapsto}^*_{\mathcal{S}} (\sigma, \sigma^\sharp)$. Then, $(\sigma_{\vec{o}}, \sigma^\sharp_{\vec{o}})$ is a configuration and there is an isomorphism between $(\sigma_{\vec{o}}, \sigma^\sharp_{\vec{o}})$ and $(\sigma, \sigma^\sharp)$.

Proof: The proof proceeds by induction on the length of $\vec{o}$.

If $\vec{o} = \cdot$, the result follows immediately.

If the move sequence has the form $(\vec{o}, \hat{o})$ with $(\cdot, \cdot) \overset{\vec{o}}{\longmapsto}^*_{\mathcal{S}} (\hat{\sigma}, \hat{\sigma}^\sharp)$ and moreover $(\hat{\sigma}, \hat{\sigma}^\sharp) \overset{\hat{o}}{\longmapsto}_{\mathcal{S}} (\sigma, \sigma^\sharp)$, the induction hypothesis allows us to assume that $(\hat{\sigma}_{\vec{o}}, \hat{\sigma}^\sharp_{\vec{o}})$ and $(\hat{\sigma}, \hat{\sigma}^\sharp)$ are isomorphic. We then show by cases on the structure of the move $\hat{o}$ that this isomorphism is preserved by the extension of $\vec{o}$ with any legal move that leads to $(\sigma, \sigma^\sharp)$. We will spare the reader further details from this simple, but rather tedious proof. □

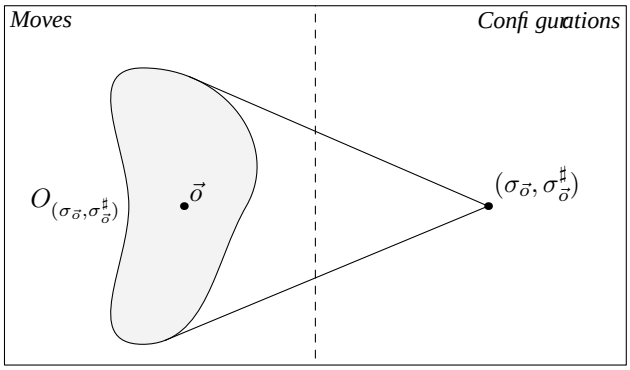


Figure 10. Relating Moves and Bundles

The two constructions we have just defined are essentially inverse of each other, as schematized in Figure 10. Given a configuration, the first returns the set of all the move sequences that produce it. Given a move sequence, the second returns the resulting configuration. In particular, observe that, when starting from a configuration, chaining these transformations yields the same configuration. However, if we start from a move sequence, their cascaded application will return the set of all sequences that construct its same target configuration. These remarks are summarized in the following corollary and in Figure 10.

Corollary B.4
Let $(\sigma,\sigma^\sharp)$ be a configuration over a set of parametric strands $\mathcal{S}$.

1. *For every $\vec{o}$ such that $(\cdot,\cdot)\stackrel{\vec{o}}{\longmapsto}{}^*_{\mathcal{S}}(\sigma,\sigma^\sharp)$, we have that $\vec{o}\in O_{(\sigma_{\vec{o}},\sigma^\sharp_{\vec{o}})}$.*
2. *For every $\vec{o}\in O_{(\sigma,\sigma^\sharp)}$, $(\sigma_{\vec{o}},\sigma^\sharp_{\vec{o}})$ is isomorphic to $(\sigma,\sigma^\sharp)$.*

Proof: The first statement reduces to Property B.1 after observing that $O_{(\sigma_{\vec{o}},\sigma^\sharp_{\vec{o}})}=O_{(\sigma,\sigma^\sharp)}$ since $(\sigma_{\vec{o}},\sigma^\sharp_{\vec{o}})$ is isomorphic to $(\sigma,\sigma^\sharp)$. The second part is a consequence of Properties B.2 and B.3. □

These considerations allow us to extract a useful notion of equivalence between move sequences: $\vec{o}_1$ and $\vec{o}_2$ are *equivalent* if they produce the same configuration, which can be tested by verifying whether $(\sigma_{\vec{o}_1},\sigma^\sharp_{\vec{o}_1})$ and $(\sigma_{\vec{o}_2},\sigma^\sharp_{\vec{o}_2})$ are isomorphic. The equivalence class to which a move sequence $\vec{o}$ belongs is therefore $O_{(\sigma_{\vec{o}},\sigma^\sharp_{\vec{o}})}$. Notice also that, in general, symmetry considerations do not allow selecting a unique element of $O_{(\sigma,\sigma^\sharp)}$ as "the" normal move sequence from $(\cdot,\cdot)$ to a configuration $(\sigma,\sigma^\sharp)$: this suggests that $(\sigma_{\vec{o}},\sigma^\sharp_{\vec{o}})$ is the most compact representation of the equivalence class $O_{(\sigma,\sigma^\sharp)}$ of $\vec{o}$.