Interpreting Strands in Linear Logic*†

I. Cervesato
ITT Industries
iliano@itd.nrl.navy.mil

N. Durgin
Stanford University
nad@cs.stanford.edu

M. Kanovich, A. Scedrov
University of Pennsylvania
{*maxkanov, scedrov*}*@math.upenn.edu*

Abstract

The adoption of the Dolev-Yao model, an abstraction of security protocols that supports symbolic reasoning, is responsible for many successes in protocol analysis. In particular, it has enabled using logic effectively to reason about protocols. One recent framework for expressing the basic assumptions of the Dolev-Yao model is given by strand spaces, certain directed graphs whose structure reflects causal interactions among protocol participants. We represent strand constructions as relatively simple formulas in first-order linear logic, a refinement of traditional logic known for an intrinsic and natural accounting of process states, events, and resources. The proposed encoding is shown to be sound and complete. Interestingly, this encoding differs from the multiset rewriting definition of the Dolev-Yao model, which is also based on linear logic. This raises the possibility that the multiset rewriting framework may differ from strand spaces in some subtle way, although the two settings are known to agree on the basic secrecy property.

1 Introduction

In recent years, a variety of methods have been developed for analyzing and reasoning about protocols based on cryptographic primitives. Although there are many differences among these proposals, most current formal approaches use the so-called "Dolev-Yao" model of adversary capabilities, which appears to be drawn from positions taken in [34] and from a simplified model presented in [11]. In this idealized setting, a protocol adversary is allowed to nondeterministically choose among possible actions. Messages are composed of indivisible abstract values, not sequences of bits, and encryption is modeled in an idealized way. The adversary may only send messages comprised of data it "knows" as the result of overhearing past transmissions.

The Dolev-Yao abstraction makes symbolic reasoning about crypto-protocols a viable approach. This observation has materialized in a number of successful analyses that use model checking [29, 33, 38] and on several proposals based on logic, the quintessential tool of symbolic reasoning [4, 35].

*Partially supported by DoD MURI "Semantic Consistency in Information Exchange" as ONR Grant N00014-97-1-0505, by NSF Grants CCR-9509931, CCR-9629754 and CCR-9800785, and by NRL under contract N0014-96-D2024 to various authors.

†We would like to thank John Mitchell, Patrick Lincoln, Catherine Meadows and Sylvan Pinsky for the fruitful discussions that contributed to this work.

One recent setting for stating the basic assumptions of the Dolev-Yao model is given by strand spaces [13, 14, 39]. Roughly, a strand is a linearly ordered sequence of events that represents the actions of a protocol participant. A strand space is a collection of strands, equipped with a graph structure generated by causal interaction among participants. This is closely related to Lamport's notion of causality in distributed systems [21], and a clear instance of Mazurkiewicz's definition of trace within concurrency theory [31]. Prior to a run of the protocol, each principal chooses certain data to be used in the protocol, such as keys or nonces.

In contrast, a formal definition of the Dolev-Yao model in terms of multiset rewriting with existential quantification, *MSR* [6, 7, 12], allows new values such as keys and nonces to be chosen at any time during the protocol run, as the need for new choices arises. In this formalism, a way of choosing new values is provided by the proof rules of existential quantification. The *MSR* formalism has been incorporated into a high-level specification language for authentication protocols, CAPSL [10].

In [7], we established a substantial equivalence of the *MSR* and strand space formalisms. We introduced a suitable abstraction of strand configurations that corresponds to *MSR* states, and showed that related pairs of states and configurations are equi-reachable. This is relevant for security analysis because several basic properties of security protocols (*e.g.* secrecy) can be phrased as reachability problems. However, it is not clear that all relevant properties of security protocols can be phrased in terms of reachability. Thus a more refined analysis of the *MSR* and strand space formalisms might reveal the differences between the two formalisms in regard to some subtle properties of protocols. In this paper, which may be seen as a companion to [7], we provide some preliminary steps in this direction.

The *MSR* and strand space formalisms are analyzed here in the formal setting of *linear logic* [16], a refinement of modal logic with an intrinsic and natural accounting of process states and events. The choice of linear logic is natural because of the very close connection between multiset rewriting and simple fragments of linear logic, which has been studied extensively [3, 30, 15, 19, 5]. We extend this standard correspondence to include first-order parameters and existentially quantified variables.

On the other hand, we also formally represent strand constructions as relatively simple formulas in first-order linear logic. This encoding is also shown to be sound and complete. As in our previous work on multiset rewriting specifications of security protocols [6, 7, 12], the proof rules of existential quantification provided a way of choosing new values, such as nonces or keys. However, the linear logic interpretation introduced here maintains the strand space intuition that nonces are chosen *before* the protocol is run. Let us note that this encoding differs from the standard linear logic representation of multiset rewriting. This raises the possibility that the multiset rewriting framework may differ from strand spaces in some subtle way.

Linear logic has found applications in numerous areas of Computer Science, and it has concrete prospects of influencing the field of security protocol analysis in a similar way. As a specification language, linear logic has been used to provide elegant and effective representations of many systems that share characteristics with crypto-protocols [8, 9, 17, 18]. The natural embedding of concurrent systems in linear logic [15, 20], in particular in its graph-based presentations [16], is also likely to be relevant, given the interpretation of security protocols as concurrent systems [1, 40]. Work on meta-reasoning in linear logic [32] promises to address protocol correctness [4, 35] effectively and efficiently. Finally, some of the theoretical results linear logic has brought about (*e.g.* complexity issues [23]) are expected to yield a better understanding of the most fundamental aspects of security protocols [12].

This paper is organized as follows: Section 2 recalls the notion of strand and reachability between strand configurations; Section 3 provides some background on linear logic; Section 4 describes the translation of strand constructions into linear logic, while in Section 5 we prove soundness and completeness theorems that relate strand reachability and linear logic derivability for their translation. In Section 6, we compare these results with the translation of the multiset rewriting specifications of a security protocol in linear logic.

2 Parametric Strands

In this section, we recall first the notions of strand spaces and bundles [13, 39], and then recent extensions aimed at capturing protocol execution at the level of strands [7].

An *event* is a pair consisting of a message m and an indication of whether it has been sent ($+m$) or received ($-m$) [13]. A *strand* is a finite sequence of events. We indicate strands with the letter s, the length of a strand as $|s|$, and its i-th event as s_i (for $i = 1 \ldots |s|$). A strand s is therefore a chain graph $(S, \Longrightarrow)$, where $S = \{s_i : i = 1 \ldots |s|\}$, moreover $s_i \Longrightarrow s_j$ iff $j = i + 1$, and finally the nodes s_i are labeled with events.

A *strand space* is a set of strands with an additional relation $(\longrightarrow)$ on the nodes, such that if $\nu_1 \longrightarrow \nu_2$, then $\nu_1 = +m$ and $\nu_2 = -m$; $\longrightarrow$ represents the transmission of the message m from the sender ν_1 to the receiver ν_2. A strand space is therefore a graph with two types of arrows, a *bi-graph* using the terminology in [7], $\sigma = (S, \Longrightarrow, \longrightarrow)$ with the above restriction on $\longrightarrow$. Given such σ, we will sometimes write S_σ, $\Longrightarrow_\sigma$, and $\longrightarrow_\sigma$ for S, $\Longrightarrow$, and $\longrightarrow$ respectively.

Let S^+ and S^- indicate the set of positively- and negatively-labeled nodes in S respectively. A *bundle* [13, 7] (see also [21]) is a strand space $\sigma = (S, \Longrightarrow, \longrightarrow)$ such that the bipartite graph $(S^+, S^-, \longrightarrow)$ is functional (a positive node has at most one outgoing $\longrightarrow$-edge), injective (a negative node has at most one incoming $\longrightarrow$-edge), and surjective (a negative node has at least one incoming $\longrightarrow$-edge), and $(\Longrightarrow \cup \longrightarrow)$ is acyclic [7]. In terms of protocols, the first three constraints imply that a message is sent to at most one recipient at a time, 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. Therefore, a bundle represents a snapshot of the execution of a protocol.

$$\underline{\rho(\vec{x},\vec{n})}: \quad \boxed{\vec{n}\ \textit{fresh},\ \pi(\vec{x})} \qquad \begin{array}{c} \top \\ \Downarrow \\ \pm m_1(\vec{x},\vec{n}) \\ \Downarrow \\ \pm m_2(\vec{x},\vec{n}) \\ \vdots \\ \pm m_{|\rho|-1}(\vec{x},\vec{n}) \\ \Downarrow \\ \pm m_{|\rho|}(\vec{x},\vec{n}) \\ \Downarrow \\ \bot \end{array}$$

Figure 1: A Parametric Strand

We now build on these accepted definitions and present a strand-based language for the specification of protocols and of their execution. The interested reader may consult [7] for further details.

Data such as the identity of principals and their long-term keys often constitute the stage on which the execution of a protocol takes place, and does not change as it unfolds. We represent and access this *persistent information* through a fixed multiset Π of ground atomic formulas with distinguished *persistent predicates* (*e.g.* PubK and PrvK) [7].

A *role* is modeled as 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. A *parametric strand* for the role ρ may look as in Figure 1. 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 [13, 39], the values $\vec{n}$ are *uniquely originated*. The relationship between variables are expressed in [39] using intuitive notation, *e.g.* k^{-1} for the inverse key of k, or k_A for the public key of A. We formalize these relations by equipping ρ with *constraints* $\pi(\vec{x})$, that, without loss of generality, will be a set of persistent atomic formulas parameterized over $\vec{x}$. In this paper, it is convenient to equip each parametric strand with an *initial node* labeled with $\top$ and an *ending node* labeled $\bot$. This addition is discussed at length in [7].

A *protocol* is given as a set of roles. The model of the intruder in the style of Dolev and Yao [11, 34] 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 [7, 39]. As an example, Figure 2 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. We ask the reader to

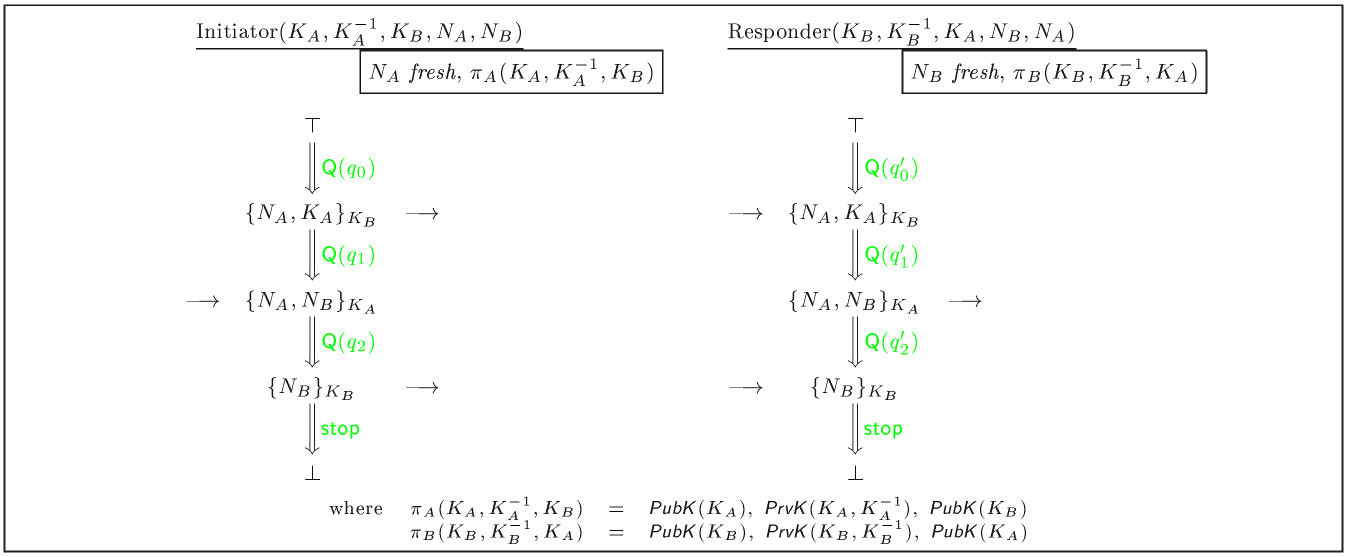


Figure 2: Extended Strand Specification of Needham-Schroeder

ignore the shaded annotations on the $\Longrightarrow$-edges for the moment.

These definitions allow us to specialize the bundles we are looking at: given a set of parametric strands $\mathcal{S}$, every strand in a bundle σ should be an initial prefix of an instantiated 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* $\mathcal{S}$.

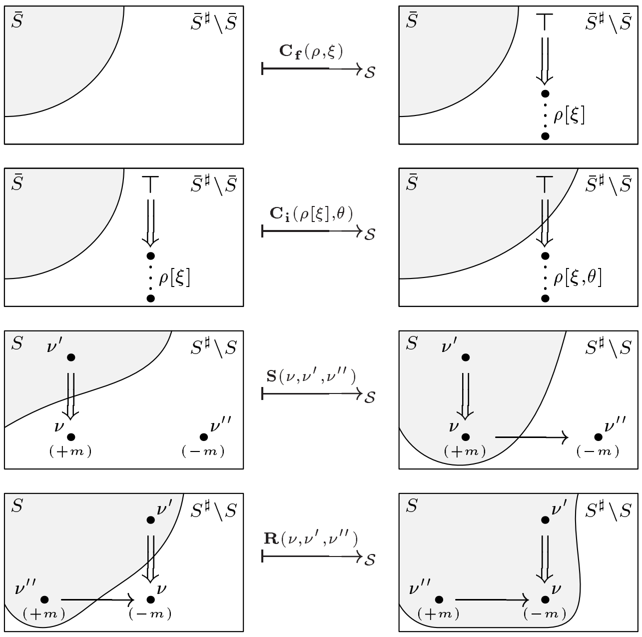
We will now give a few definitions needed to emulate the execution of a protocol with parametric strands. First, observe that the network traffic in a bundle is expressed in terms of events and of the $\longrightarrow$ relation. The edges of $\longrightarrow$ represent past traffic: messages that have 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. 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.

Given these preliminary definitions, a *configuration* over $\mathcal{S}$ is a structure $(\sigma, \sigma^\sharp)_\Sigma$ where σ is a bundle over $\mathcal{S}$, and $\sigma^\sharp$ is an extension of σ whose only additional $\longrightarrow$-edges originate in $\mathrm{Fr}(\sigma)$, cover all of $\mathrm{Fr}(\sigma)$, and point to $\sigma^\sharp \setminus \sigma$, and finally the *signature* Σ lists all the symbols that appear in $\sigma^\sharp$ (and σ).

We define the notion of *one-step transition* between two configurations $(\sigma_1, \sigma_1^\sharp)_{\Sigma_1}$ and $(\sigma_2, \sigma_2^\sharp)_{\Sigma_2}$, written $(\sigma_1, \sigma_1^\sharp)_{\Sigma_1} \overset{o}{\longmapsto}_{\mathcal{S}} (\sigma_2, \sigma_2^\sharp)_{\Sigma_2}$, by means of four rules that we call $\mathbf{C_f}$, $\mathbf{C_i}$, $\mathbf{S}$ and $\mathbf{R}$. For the sake of conciseness, we limit ourselves to an intuitive presentation based on the following sketches. A formal definition can be found in [7].



The *move* o that labels the transition arrow $\longmapsto_{\mathcal{S}}$ records the necessary information to reconstruct the transition uniquely.

Rule $\mathbf{C_f}$ describes the instantiation of a parametric strand $\rho(\vec{x}, \vec{n})$ with a substitution ξ for all its variables that are marked "fresh" ($\vec{n}$). The substituted constants must be distinct from each other and from any other value appearing in $\sigma_1^\sharp$. Rule $\mathbf{C_i}$ realizes the second stage of instantiation: it applies a substitution θ to the remaining variables $\vec{x}$ of a partially instantiated strand $\rho[\xi]$, checks that the atomic formulas resulting from instantiating the constraints $\pi(\vec{x})$ of ρ with θ satisfy Π, and install its initial node $\top$ in $\sigma_1^\sharp$ to produce $\sigma_2^\sharp$. We must perform instantiation in two stages to handle protocols where two parties exchange newly produced nonces as in the Needham-Schroeder protocol in Figure 2.

The remaining rules deal with message transmission and reception once a strand has been installed in the configuration. In particular, ν, ν' and ν'' are nodes on fully instantiated strands. Rule **S** models the action of sending a message: if the configuration at hand embeds a strand that is not fully contained in the bundle part σ_1 and the first missing node ν is positive, we add an $\longrightarrow$-arrow to a matching negative node ν'' and include ν in σ_2. Receiving a message is modeled by rule **R**: if $(\sigma_1, \sigma_1^\sharp)_{\Sigma_1}$ mentions a strand that is not fully contained in its bundle part and its first missing node ν has an incoming $\longrightarrow$-edge, we add it to the bundle.

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

Our definition of transition preserves configurations, *i.e.* if $(\sigma_1, \sigma_1^\sharp)_{\Sigma_1}$ is a configuration and $(\sigma_1, \sigma_1^\sharp)_{\Sigma_1} \overset{o}{\longmapsto}_{\mathcal{S}} (\sigma_2, \sigma_2^\sharp)_{\Sigma_2}$, then $(\sigma_2, \sigma_2^\sharp)_{\Sigma_2}$ is also a configuration. Moreover, $\Sigma_1 \subseteq \Sigma_2$. These properties extend to multistep transitions.

3 Elements of Linear Logic

The target of our interpretation of strand constructions will be a sublanguage of linear logic [16]. We choose this formalism over more traditional logics because of its interpretation of formulas as consumable resources. This provides, for example, a simple way of modeling the fact that receiving a message makes it unavailable to other recipients (unless further actions are taken).

This language, a fragment of first-order multiplicative exponential linear logic to be precise, is given by the following grammar:

A	$::= P$	*Atomic formulas*
	$\mid \mathbf{1}$	*Multiplicative unit*
	$\mid A_1 \otimes A_2$	*Multiplicative conjunction*
	$\mid A_1 \multimap A_2$	*Linear implication*
	$\mid \forall x. A$	*Universal quantification*
	$\mid \exists x. A$	*Existential quantification*

P	$::= \mathsf{N}(m)$	*Message in transit*
	$\mid \mathit{PubK}(k) \mid \ldots$	*Persistent information*
	$\mid \mathsf{Q}(q)$	*Intermediate step*
	$\mid \mathsf{stop}$	*Role completion*

We use messages and their constituents as the basis of the term language of our formulas, as described in [7]. We rely on the unary predicate symbol N to hold messages being exchanged, and we maintain the syntax we glimpsed at in the previous section for persistent information. Atoms of the form $\mathsf{Q}(q)$ identify uniquely the intermediate $\Longrightarrow$-edges in a configuration (see Section 4). Finally, the atomic constant stop will indicate the completion of a strand.

The connectives we will need are $\otimes$, known as *multiplicative conjunction*, the constant $\mathbf{1}$ (*multiplicative unit*), $\multimap$ or *linear implication*, and the usual quantifiers. A resource $A_1 \otimes A_2$ consists of the sum of parts A_1 and A_2, while $A_1 \multimap A_2$ realizes resource A_2 subject to the availability of A_1 while consuming A_1 itself (this implements therefore the notion of transition). When instantiating quantifiers, we write $[t/x]A$ for the capture-free substitution of term t for variable x in formula A. We abbreviate $[t_1/x_1]([t_2/x_2](\ldots[t_n/x_n]A))$ as $[t_1/x_1, \ldots, t_n/x_n]A$.

Contexts are finite multisets of comma-separated formulas. The empty context is denoted "$\cdot$". We will use the letter Γ and Δ, possibly subscripted, to indicate contexts. A *signature* Σ is a list of constants.

The derivability judgments we will rely upon are *sequents* of the form [17]:

$$\Gamma; \Delta \vdash_\Sigma A$$

where the formulas in Γ and Δ are the resources available to produce the formula A. While the elements in Δ shall be used exactly once, the resources in Γ can be exploited arbitrarily many times, possibly zero. This convenient two-context formulation [17] is rewritten as a more common single-context sequent by augmenting Δ with the result of prefixing every formula in Γ with the *exponential modality* "!" [16]. The signature Σ lists all the constants mentioned in the sequent. Although usually omitted in presentations of (linear) logic, it simplifies our treatment of nonces.

The relevant inference rules for this language are displayed in Figure 3. The rules on the left-hand side are called *multiplicative*. Rule **id** will be used at the leaves of a derivation: it specifies that an object A can be trivially produced from A itself (the formulas in Γ are ignored). Notice that no excess resources are admitted. Rule $\multimap$**l** specifies how to use a resource $A_1 \multimap A_2$ to build an object C: if A_1 can be produced using part of the context (Δ_1), then A_2 and what remains of the context (Δ_2) are available to produce C. Rule $\otimes$**l** states that a composite resource can be broken to make its components individually available, while $\otimes$**r** specifies that $A_1 \otimes A_2$ is produced by building its parts independently. The constant $\mathbf{1}$ is the non-resource: it does not contribute to a goal (rule **1l**) and does require any resource to be established (rule **1r**). Rule **cut** permits constructing an object in stages: in order to obtain C, one can first build A with some of the available resources, and then use A to achieve C. Since C can always be produced directly from the original resources, this rule is effectively redundant, which we emphasize by displaying it in a shaded font.

The right-hand side of Figure 3 shows rule **dl** (dereliction), which makes a copy of a formula A in Γ available in Δ, and the rules concerning the quantifiers. Observe that the existential quantifiers in the context Δ are instantiated with *new* constants (rule $\exists$**l**), which we record in the signature Σ. In the right-hand side of the turnstile (rule $\exists$**r**), these quantifiers have instead the function of hiding the use of these newly introduced constants. Notice that they are instantiated with constants from Σ rather than with arbitrary terms: this is sufficient for our purposes. Instead, we allow universally quantified variables to be instantiated with arbitrary terms (rule $\forall$**l**).

4 Strands in Linear Logic

We will now describe the translation of parametric strands and configurations into the fragment of linear logic we just discussed. We shall emphasize that this encoding does not treat penetrator strands differently from regular protocol strands in any way. This adheres to the strand philosophy,

Multiplicatives

$$\frac{}{\Gamma;\, A \vdash_\Sigma A}\,\textbf{id} \qquad \frac{\Gamma;\, \Delta_1 \vdash_\Sigma A \quad \Gamma;\, \Delta_2, A \vdash_\Sigma C}{\Gamma;\, \Delta_1, \Delta_2 \vdash_\Sigma C}\,\textbf{cut}$$

$$\frac{\Gamma;\, \Delta_1 \vdash_\Sigma A_1 \quad \Gamma;\, \Delta_2, A_2 \vdash_\Sigma C}{\Gamma;\, \Delta_1, \Delta_2, A_1 \multimap A_2 \vdash_\Sigma C}\,{\multimap}\textbf{l}$$

$$\frac{\Gamma;\, \Delta, A_1, A_2 \vdash_\Sigma C}{\Gamma;\, \Delta, A_1 \otimes A_2 \vdash_\Sigma C}\,{\otimes}\textbf{l} \qquad \frac{\Gamma;\, \Delta_1 \vdash_\Sigma A_1 \quad \Gamma;\, \Delta_2 \vdash_\Sigma A_2}{\Gamma;\, \Delta_1, \Delta_2 \vdash_\Sigma A_1 \otimes A_2}\,{\otimes}\textbf{r}$$

$$\frac{\Gamma;\, \Delta \vdash_\Sigma C}{\Gamma;\, \Delta, \mathbf{1} \vdash_\Sigma C}\,\textbf{1l} \qquad \frac{}{\Gamma;\, \cdot \vdash_\Sigma \mathbf{1}}\,\textbf{1r}$$

Exponentials and quantifiers

$$\frac{\Gamma, A;\, \Delta, A \vdash_\Sigma C}{\Gamma, A;\, \Delta \vdash_\Sigma C}\,\textbf{dl}$$

$$\frac{\Gamma;\, \Delta, [t/x]A \vdash_\Sigma C}{\Gamma;\, \Delta, \forall x.\, A \vdash_\Sigma C}\,\forall\textbf{l}$$

$$\frac{\Gamma;\, \Delta, [c/x]A \vdash_{\Sigma,c} C}{\Gamma;\, \Delta, \exists x.\, A \vdash_\Sigma C}\,\exists\textbf{l}\ (*) \qquad \frac{\Gamma;\, \Delta \vdash_{\Sigma,c} [c/x]A}{\Gamma;\, \Delta \vdash_{\Sigma,c} \exists x.\, A}\,\exists\textbf{r}$$

$(*)$ c not in Σ

Figure 3: Relevant Rules of Linear Logic

and contrasts with other approaches which syntactically differentiate the intruder model (*e.g.* [6]).

Let $\rho(\vec{n}, \vec{x}, \vec{y})$ be a parametric strand with constraints "$\vec{n}$ fresh, $\pi(\vec{x})$". Let $s_0, s_1, \ldots, s_n, s_{n+1}$ be the nodes of ρ, with $s_0 = \top$, $s_{n+1} = \bot$, and for $i = 0..n$, $s_i \Longrightarrow s_{i+1}$. We define the *encoding* of node s_i (for $i = 0..n+1$), written $\ulcorner s_i \urcorner$, as follows:

$$\begin{array}{lll}
\ulcorner s_{n+1} \urcorner & = \mathsf{stop} & \\
\ulcorner s_i \urcorner & = \mathsf{Q}(q_{i-1}) \otimes (\mathsf{Q}(q_{i-1}) \otimes \mathsf{N}(m) \multimap \ulcorner s_{i+1} \urcorner) & \\
 & & \text{if } s_i = -m \\
\ulcorner s_i \urcorner & = \mathsf{Q}(q_{i-1}) \otimes (\mathsf{Q}(q_{i-1}) \multimap \mathsf{N}(m) \otimes \ulcorner s_{i+1} \urcorner) & \\
 & & \text{if } s_i = +m \\
\ulcorner s_0 \urcorner & = \otimes\pi(\vec{x}) \multimap \otimes\pi(\vec{x}) \otimes \ulcorner s_1 \urcorner &
\end{array}$$

where, given a multiset of formulas Δ, $\otimes\Delta$ is the formula obtained by taking the multiplicative conjunction of very element of Δ. For $i = 1..n$, $\ulcorner s_i \urcorner$ expresses the action in s_i by placing the sent (received) message m in the consequent (resp. antecedent) of the implication. Rule $\multimap$**l** will have the effect of inserting (resp. removing) $\mathsf{N}(m)$ into (resp. from) the context Δ. Notice that its application will also insert $\ulcorner s_i \urcorner$ into Δ, enabling in this way the next action. This technique is known as *continuation-passing style* in the programming language community.

The arguments of the conjuncts $\mathsf{Q}(q_0), \ldots, \mathsf{Q}(q_{n-1})$ are distinct variables. It is convenient to interpret them as labels for the $\Longrightarrow$-edges of ρ, as shown in Figure 2. The last arrow, leading to $s_{n+1} = \bot$, is instead labeled with the propositional letter stop. These atoms serve multiple purposes: first they provide a way to preserve the order of consecutive message transmissions or receptions, which may prove important for some applications; second their presence greatly simplifies our proofs as they implement the "locks and keys" technique [23], which has yielded faithful representations of various computational paradigms in linear logic [22, 23, 24]; third, they are a crucial device for bridging the gap with the multiset rewriting specification of a protocol [7]. Were these reasons, in particular the first one, to fall, a simpler encoding can be achieved by replacing the $\mathsf{Q}(q_i)$'s and stop with the linear logic constant $\mathbf{1}$.

The encoding of ρ is achieved by appropriately quantifying the free variables in $\ulcorner s_0 \urcorner$:

$$\ulcorner \rho \urcorner = \exists q_o, \ldots, q_{n-1}.\, \exists \vec{n}.\, \forall \vec{x}, \vec{y}.\, \ulcorner s_0 \urcorner$$

Existentially quantifying the q_i's, as well as $\vec{n}$, guarantees that they will be instantiated with constants distinct from any other value in use. Applying this encoding to the Needham-Schroeder protocol specified in Figure 2 yields the linear logic formulas in Figure 4.

In order to define the encoding of a configuration, we need to extend this notation to partially and fully instantiated strands. Let ξ be a substitution for the "fresh" variables $\vec{n}$ of ρ. Then

$$\ulcorner \rho[\xi] \urcorner = [\xi, u_0/q_0, \ldots, u_{n-1}/q_{n-1}] \forall \vec{x}, \vec{y}.\, \ulcorner s_0 \urcorner,$$

where $u_0, \ldots, u_{n-1}$ are distinct constants. If furthermore θ is a substitution for the remaining variables $\vec{x}, \vec{y}$ of ρ, we define

$$\ulcorner \rho[\xi, \theta] \urcorner = [\theta]([\xi, u_0/q_0, \ldots, u_{n-1}/q_{n-1}] \ulcorner s_0 \urcorner).$$

Observe that θ can mention some of the constants newly introduced by ξ. We extend the notation $\ulcorner s_i \urcorner$, for $i = 0..n+1$, to the case where s_i is a node in a fully instantiated strand (clearly, the q_i's will have been replaced with u_i's).

We shall now encode configurations. A configuration $(\sigma, \sigma^\sharp)_\Sigma$ comprises three types of information: 1) an account of how this situation has been reached (as σ and the strands in $\sigma^\sharp$ that have been instantiated, but not yet used); 2) a description of the current situation (in $\mathrm{Fr}(\sigma)$); and 3) a summary of the future actions that can be performed (in $\sigma^\sharp \setminus \sigma$). We will ignore the first aspect since it will be partially captured through the notion of derivation. The representation of $\mathrm{Fr}(\sigma)$ will simply be the conjunction of the messages in it (or $\mathbf{1}$ if none is present):

$$\ulcorner \mathrm{Fr}(\sigma) \urcorner = \bigotimes \wr \mathsf{N}(m) : m \in \mathrm{Fr}(\sigma) \int.$$

where we write $\wr \ldots \int$ for the multiset equivalent of the usual set notation $\{\ldots\}$. As for $\sigma^\sharp \setminus \sigma$, we take the conjunction of the representation of each fully instantiated residual strand in it, plus the representation of the strands that are only partially instantiated:

$$\begin{array}{rl}
\ulcorner \sigma^\sharp \setminus \sigma \urcorner = \bigotimes \wr \ulcorner s_{i+1} \urcorner : & s = \rho[\xi, \theta] \text{ in } \sigma^\sharp, \\
 & s_i \in \sigma_S,\ \text{and } s_{i+1} \in \sigma_S^\sharp \setminus \sigma_S \int \\
\otimes \bigotimes \wr \ulcorner \rho[\xi] \urcorner : \rho[\xi] \text{ in } \sigma^\sharp \int. &
\end{array}$$

For the sake of conciseness, we define the representation of a configuration $(\sigma, \sigma^\sharp)_\Sigma$ as

$$\ulcorner (\sigma, \sigma^\sharp)_\Sigma \urcorner = \ulcorner \mathrm{Fr}(\sigma, \sigma^\sharp) \urcorner \otimes \ulcorner \sigma \setminus \sigma^\sharp \urcorner.$$

$$
\begin{array}{rcll}
\exists q_0, q_1, q_2.\, \exists n_A.\, \forall k_A, k_A^{-1}, k_B, n_B. \quad \otimes\pi_A(k_A, k_A^{-1}, k_B) & \multimap & \otimes\pi_A(k_A, k_A^{-1}, k_B) \otimes \mathsf{Q}(q_0) & \otimes (\\
\mathsf{Q}(q_0) & \multimap & \mathsf{N}(\{k_A, n_A\}_{k_B}) \otimes \mathsf{Q}(q_1) & \otimes (\\
\mathsf{Q}(q_1) \otimes \mathsf{N}(\{n_A, n_B\}_{k_A}) & \multimap & \mathsf{Q}(q_2) & \otimes (\\
\mathsf{Q}(q_2) & \multimap & \mathsf{N}(\{n_B\}_{k_B}) & \otimes \mathsf{stop})))
\\[1em]
\exists q_0', q_1', q_2'.\, \exists n_B.\, \forall k_B, k_B^{-1}, k_A, n_A. \quad \otimes\pi_B(k_B, k_B^{-1}, k_A) & \multimap & \otimes\pi_B(k_B, k_B^{-1}, k_A) \otimes \mathsf{Q}(q_0') & \otimes (\\
\mathsf{Q}(q_0') \otimes \mathsf{N}(\{k_A, n_A\}_{k_B}) & \multimap & \mathsf{Q}(q_1') & \otimes (\\
\mathsf{Q}(q_1') & \multimap & \mathsf{Q}(q_2') \otimes \mathsf{N}(\{n_A, n_B\}_{k_A}) & \otimes (\\
\mathsf{Q}(q_2') \otimes \mathsf{N}(\{n_B\}_{k_B}) & \multimap & & \quad \mathsf{stop})))
\end{array}
$$

Figure 4: Linear Logic Translation of the Needham-Schroeder Protocol

We will make the encoding we have just presented more concrete by means of an example. The upper part of Figure 5 shows a configuration $(\sigma, \sigma^\sharp)$ representing an initial stage of Lowe's attack [26] on the Needham-Schroeder protocol in Figure 2. It contains six strands: an initiator, a responder, and one instance of each of the four penetrator strands M' (access to the intruder's initial knowledge), D (decryption), M (access to public information) and E (encryption). A detailed discussion of penetrator strands can be found in [7]. Constants are indicated using a different font than variables (*e.g.* $\mathsf{n_A}$ as opposed to n_A). In this configuration, the initiator has executed its first action, strands M' and D have been completed, strands M and E are fully instantiated but still have to execute their first action, while the responder strand has been only partially instantiated. The only message in transit (the fringe) is $(\mathsf{n_A}, \mathsf{k_A})$.

The second box in figure 5 shows the encoding of $(\sigma, \sigma^\sharp)$ in linear logic. Observe that, whenever the border between σ and $\sigma^\sharp \setminus \sigma$ crosses an active strand, the atomic formula $\mathsf{Q}(\mathsf{u}_i^\rho)$ corresponding to the intersected $\Longrightarrow$-edge appears as a conjunct in $\ulcorner \sigma^\sharp \setminus \sigma \urcorner$. Similarly, each terminated strand contributes an occurrence of stop. The residual of every active strand yields a formula with implications. Finally, notice that the representation of the partially instantiated responder strand accounts for the only quantifiers appearing in $\ulcorner(\sigma, \sigma^\sharp)\urcorner$.

5 Soundness and Completeness

We will now show that, given the above encoding, reachability among configurations is mapped to the derivability of their representation in linear logic, and vice versa. Constructing a derivation that mimics a sequence of moves in the strand world, formally stated in the following theorem, is fairly simple.

Theorem 5.1 *(Soundness)*
Let $\mathcal{S}$ be a set of parametric strands and $(\sigma_1, \sigma_1^\sharp)_\Sigma$, $(\sigma_2, \sigma_2^\sharp)_{\Sigma, \vec{n}}$ two configurations over $\mathcal{S}$. If there is a move sequence $\vec{o}$ such that

$$(\sigma_1, \sigma_1^\sharp)_\Sigma \overset{\vec{o}}{\longmapsto}{}^*_{\mathcal{S},\Pi} [\Sigma'/\vec{n}](\sigma_2, \sigma_2^\sharp)_{\Sigma,\Sigma'}$$

for some instantiation of variables $\vec{n}$ with fresh distinct constants Σ', then there exists a linear logic derivation $\mathcal{D}$ of the sequent

$$\ulcorner \mathcal{S} \urcorner;\, \Pi, \ulcorner(\sigma_1, \sigma_1^\sharp)_\Sigma\urcorner \vdash_\Sigma \exists \vec{n}. \ulcorner(\sigma_2, \sigma_2^\sharp)_{\Sigma,\vec{n}}\urcorner \otimes \Pi.$$

Proof: By induction on the structure of $\vec{o}$. If the move sequence is processed right-to-left, we obtain a cut-free derivation. Operating forward (left-to-right) requires the use of the **cut** rule (which can subsequentially be eliminated). □

In the above proof, each move is simulated by the application of a number of linear logic rules. This finer granularity is a hindrance when considering a derivation that relates the encoding of two configurations, and trying to read off the move sequences that have actually been applied: these micro-steps can be intermingled in arbitrary ways. This forces us to break our completeness proof into a number of stages aimed at disentangling the given linear logic derivation. First we reduce ourselves to a purely multiplicative setting by pushing **dl** and the quantifier rules at the bottom of the given derivation.

Lemma 5.2 *Let $\mathcal{S}$ be a set of parametric strands and $(\sigma_1, \sigma_1^\sharp)_\Sigma$, $(\sigma_2, \sigma_2^\sharp)_{\Sigma,\vec{n}}$ two configurations over $\mathcal{S}$. If there is a cut-free derivation $\mathcal{D}$ of the sequent*

$$\ulcorner \mathcal{S} \urcorner;\, \Pi, \ulcorner(\sigma_1, \sigma_1^\sharp)_\Sigma\urcorner \vdash_\Sigma \exists \vec{n}. \ulcorner(\sigma_2, \sigma_2^\sharp)_{\Sigma,\vec{n}}\urcorner \otimes \Pi$$

then there exists an instantiation of variables $\vec{n}$ with fresh distinct constants Σ', a configuration $(\sigma_0, \sigma_0^\sharp)_{\Sigma,\Sigma'}$, and a cut-free derivation $\mathcal{D}^*$ of the sequent

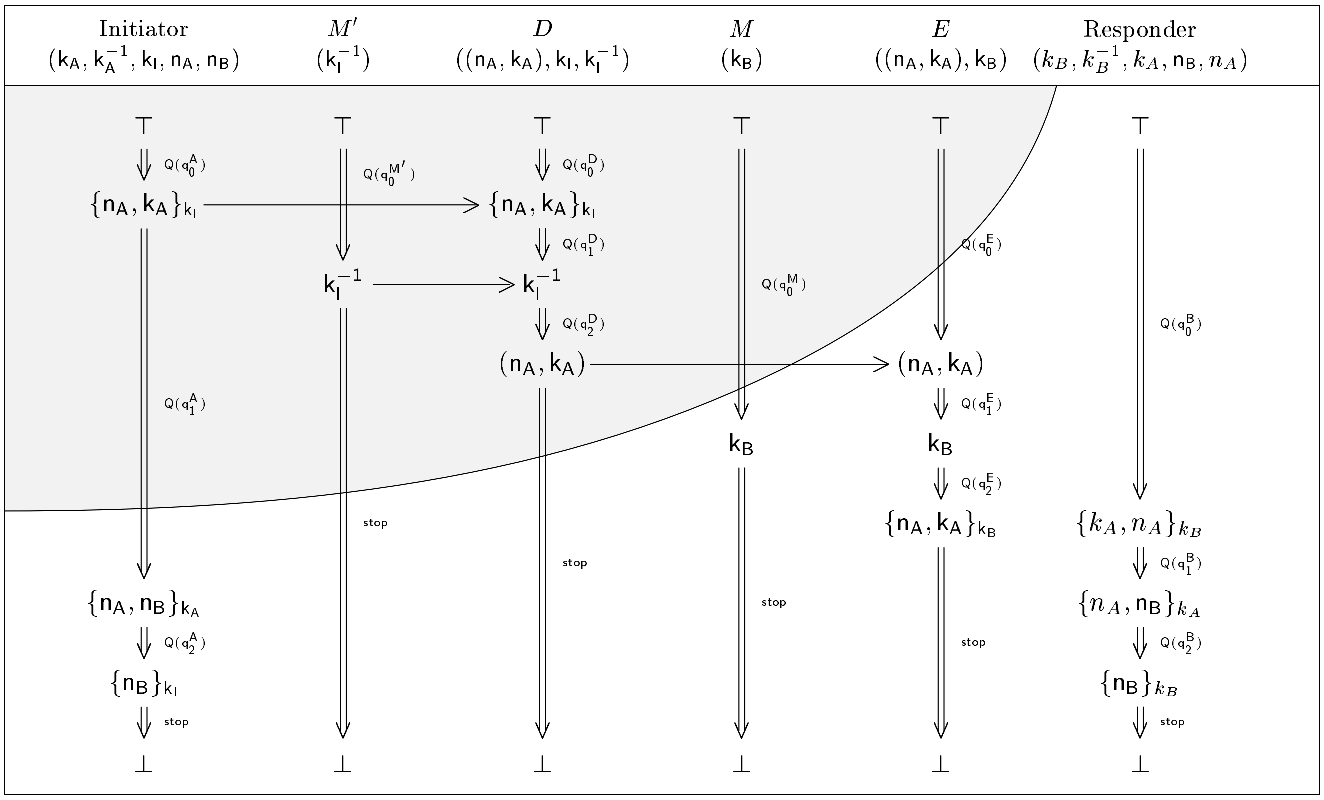
$$\ulcorner \mathcal{S} \urcorner; \Delta \vdash_{\Sigma,\Sigma'} \ulcorner[\Sigma'/\vec{n}](\sigma_2, \sigma_2^\sharp)_{\Sigma,\Sigma'}\urcorner \otimes \Pi$$

*where $\Delta = \Pi, \ulcorner(\sigma_1, \sigma_1^\sharp)_\Sigma\urcorner, \ulcorner(\sigma_0, \sigma_0^\sharp)_{\Sigma,\Sigma'}\urcorner$, that does use neither **dl** nor any of the quantifier rules $\forall$**l**, $\exists$**l**, $\exists$**r**.*

Proof: We exploit the relative permutability of these inference rules, as described in [25]. More precisely, we apply the following four steps:

1. We permute rule **dl** below every other rule. The resulting derivation consists then of a sequence of applications of **dl** followed by a subderivation $\mathcal{D}'$ that does not use this rule. The applications of **dl** correspond to committing to the parametric strands that will be used to produce $(\sigma_2, \sigma_2^\sharp)_{\Sigma,\vec{n}}$ (once instantiated, they will correspond to $(\sigma_0, \sigma_0^\sharp)_{\Sigma,\Sigma'}$).

2. We permute rule $\exists$**l** to the bottom of $\mathcal{D}'$, which enables us to consider a subderivation $\mathcal{D}''$ that does not contain these rules. These uses of $\exists$**l** correspond to picking the new constants that appear in Σ' beforehand.



$$
\ulcorner \mathrm{Fr}(\sigma) \urcorner = \mathsf{N}((\mathsf{n_A}, \mathsf{k_A}))
$$

$$
\ulcorner \sigma^\sharp \setminus \sigma \urcorner = \left\{
\begin{array}{lll}
& \mathsf{Q}(\mathsf{u}_1^{\mathsf{A}}) \otimes (\\
& \quad \mathsf{Q}(\mathsf{u}_1^{\mathsf{A}}) \otimes \mathsf{N}(\{\mathsf{n_A}, \mathsf{n_B}\}_{\mathsf{k_A}}) \multimap \mathsf{Q}(\mathsf{u}_2^{\mathsf{A}}) \otimes (& \mathrm{Initiator}(\mathsf{k_A}, \mathsf{k_A}^{-1}, \mathsf{k_I}, \mathsf{n_A}, \mathsf{n_B}) \\
& \quad\quad \mathsf{Q}(\mathsf{u}_2^{\mathsf{A}}) \multimap \mathsf{N}(\{\mathsf{n_B}\}_{\mathsf{k_I}}) \otimes \mathsf{stop})) \\
\otimes & \mathsf{stop} & M'(\mathsf{k_I}^{-1}) \\
\otimes & \mathsf{stop} & D((\mathsf{n_A}, \mathsf{k_A}), \mathsf{k_I}, \mathsf{k_I}^{-1}) \\
\otimes & \mathsf{Q}(\mathsf{u}_0^{\mathsf{M}}) \otimes (\mathsf{Q}(\mathsf{q}_0^{\mathsf{M}}) \multimap \mathsf{N}(\mathsf{k_B}) \otimes \mathsf{stop}) & M(\mathsf{k_B}) \\
\otimes & \mathsf{Q}(\mathsf{u}_0^{\mathsf{E}}) \otimes (\\
& \quad \mathsf{Q}(\mathsf{u}_0^{\mathsf{E}}) \otimes \mathsf{N}((\mathsf{n_A}, \mathsf{k_A})) \multimap \mathsf{Q}(\mathsf{u}_1^{\mathsf{E}}) \otimes (\\
& \quad\quad \mathsf{Q}(\mathsf{u}_1^{\mathsf{E}}) \otimes \mathsf{N}(\mathsf{k_B}) \multimap \mathsf{Q}(\mathsf{u}_2^{\mathsf{E}}) \otimes (& E((\mathsf{n_A}, \mathsf{k_A}), \mathsf{k_B}) \\
& \quad\quad\quad \mathsf{Q}(\mathsf{u}_2^{\mathsf{E}}) \multimap \mathsf{N}(\{\mathsf{n_A}, \mathsf{k_A}\}_{\mathsf{k_B}}) \otimes \mathsf{stop}))) \\
\otimes & (\forall k_B, k_B^{-1}, k_A, n_A. \\
& \quad \otimes \pi_B(k_B, k_B^{-1}, k_A) \multimap \otimes \pi_B(k_B, k_B^{-1}, k_A) \otimes \mathsf{Q}(\mathsf{u}_0^{\mathsf{B}}) \otimes (\\
& \quad\quad \mathsf{Q}(\mathsf{u}_0^{\mathsf{B}}) \otimes \mathsf{N}(\{k_A, n_A\}_{k_B}) \multimap \mathsf{Q}(\mathsf{u}_1^{\mathsf{B}}) \otimes (& \mathrm{Responder}(k_B, k_B^{-1}, k_A, \mathsf{n_B}, n_A) \\
& \quad\quad\quad \mathsf{Q}(\mathsf{u}_1^{\mathsf{B}}) \multimap \mathsf{Q}(\mathsf{u}_2^{\mathsf{B}}) \otimes \mathsf{N}(\{n_A, \mathsf{n_B}\}_{k_A}) \otimes (\\
& \quad\quad\quad\quad \mathsf{Q}(\mathsf{u}_2^{\mathsf{B}}) \otimes \mathsf{N}(\{\mathsf{n_B}\}_{k_B}) \multimap \mathsf{stop}))))
\end{array}
\right.
$$

$$
\begin{array}{lrll}
\text{where} & \ulcorner M' \urcorner = & \exists q_0. \forall x. & \mathsf{P_0}(x) \multimap \mathsf{P_0}(x) \otimes \mathsf{Q}(q_0) \otimes (\\
& & & \quad \mathsf{Q}(q_0) \multimap \mathsf{N}(x) \otimes \mathsf{stop}) \\
& \ulcorner D \urcorner = & \exists q_0, q_1, q_2. \forall m, k, k'. & \mathsf{PrivK}(k, k') \multimap \mathsf{PrivK}(k, k') \otimes \mathsf{Q}(q_0) \otimes (\\
& & & \quad \mathsf{Q}(q_0) \otimes \mathsf{N}(\{m\}_k) \multimap \mathsf{Q}(q_1) \otimes (\\
& & & \quad\quad \mathsf{Q}(q_1) \otimes \mathsf{N}(k') \multimap \mathsf{Q}(q_2) \otimes (\\
& & & \quad\quad\quad \mathsf{Q}(q_2) \multimap \mathsf{N}(m) \otimes \mathsf{stop}))) \\
& \ulcorner M \urcorner = & \exists q_0. \forall k. & \mathsf{PubK}(k) \multimap \mathsf{PubK}(k) \otimes \mathsf{Q}(q_0) \otimes (\\
& & & \quad \mathsf{Q}(q_0) \multimap \mathsf{N}(k) \otimes \mathsf{stop}) \\
& \ulcorner E \urcorner = & \exists q_0, q_1, q_2. \forall m, k. & \mathsf{PubK}(k) \multimap \mathsf{PubK}(k) \otimes \mathsf{Q}(q_0) \otimes (\\
& & & \quad \mathsf{Q}(q_0) \otimes \mathsf{N}(m) \multimap \mathsf{Q}(q_1) \otimes (\\
& & & \quad\quad \mathsf{Q}(q_1) \otimes \mathsf{N}(k) \multimap \mathsf{Q}(q_2) \otimes (\\
& & & \quad\quad\quad \mathsf{Q}(q_2) \multimap \mathsf{N}(\{m\}_k) \otimes \mathsf{stop})))
\end{array}
$$

Figure 5: A Translation: Lowe's attack on the Needham-Schroeder Protocol

3. We permute rules $\exists\mathbf{r}$ to the end of $\mathcal{D}''$, obtaining a subderivation $\mathcal{D}'''$ that does not mention these rules. The applications of $\exists\mathbf{r}$ correspond to hiding Σ' in the overall derivation.

4. Finally, we permute every use of $\forall\mathbf{r}$ down past every other rule, making explicit a subderivation $\mathcal{D}^*$ with the required characteristics. The applications of $\forall\mathbf{l}$ complete the instantiation of $(\sigma_0, \sigma_0^\sharp)_{\Sigma,\Sigma'}$. □

When interpreted at the strand level, Lemma 5.2 specifies that a move sequence can be rearranged so that parametric strands are chosen and instantiated before any message is exchanged. More specifically, all uses of $\mathbf{C_f}$ happen first, followed by all applications of $\mathbf{C_i}$. Only then, can $\mathbf{S}$- and $\mathbf{R}$-moves take place.

The next step consists in grouping together the rule applications that correspond to each move in the multiplicative part of the given derivation. This provides a simple way of identifying moves $\mathbf{S}$ and $\mathbf{R}$.

Lemma 5.3 *Let $\mathcal{S}$ be a set of parametric strands and $(\sigma_1, \sigma_1^\sharp)_\Sigma$, $(\sigma_2, \sigma_2^\sharp)_\Sigma$ two configurations over $\mathcal{S}$. If there is a cut-free derivation $\mathcal{D}^*$ of the sequent*

$$\ulcorner\mathcal{S}\urcorner; \Pi, \ulcorner(\sigma_1, \sigma_1^\sharp)_\Sigma\urcorner \vdash_\Sigma \ulcorner(\sigma_2, \sigma_2^\sharp)_\Sigma\urcorner \otimes \Pi$$

that uses only rules from the left half of Figure 3, then there exists a cut-free derivation $\mathcal{D}^{**}$ of this sequent such that

- *Every use of $\otimes\mathbf{r}$ appear just below $\mathbf{id}$, $\mathbf{1r}$ or $\otimes\mathbf{r}$;*
- *Rules $\mathbf{1l}$ and $\otimes\mathbf{l}$ are applied eagerly.*

Proof: Again, we take advantage of the permutability results in [25]. Rule $\otimes\mathbf{r}$ can be pushed up past any other rule (except $\mathbf{id}$ and $\mathbf{1r}$). On the other hand, $\otimes\mathbf{l}$ and $\mathbf{1l}$ can always be permuted down, as long as the nesting of subformulas is respected (clearly if the left-hand side contains a formula $(A \otimes B) \multimap C$, a proof fragment that dismantles this formula must apply $\otimes\mathbf{l}$ *above* $\multimap\mathbf{l}$). □

At this point, we have the means to extract a sequence of moves from a linear logic derivation that relates the encoding of two configurations.

Theorem 5.4 *(Completeness)*
Let $\mathcal{S}$ be a set of parametric strands and $(\sigma_1, \sigma_1^\sharp)_\Sigma$, $(\sigma_2, \sigma_2^\sharp)_{\Sigma,\vec{n}}$ two configurations over $\mathcal{S}$. If there is a linear logic derivation $\mathcal{D}$ of the sequent

$$\ulcorner\mathcal{S}\urcorner; \Pi, \ulcorner(\sigma_1, \sigma_1^\sharp)_\Sigma\urcorner \vdash_\Sigma \exists\vec{n}. \ulcorner(\sigma_2, \sigma_2^\sharp)_{\Sigma,\vec{n}}\urcorner \otimes \Pi$$

then there exists an instantiation of variables $\vec{n}$ with fresh distinct constants Σ' and a move sequence $\vec{\sigma}$ such that

$$(\sigma_1, \sigma_1^\sharp)_\Sigma \stackrel{\vec{\sigma}}{\longmapsto}^*_{\mathcal{S},\Pi} [\Sigma'/\vec{n}](\sigma_2, \sigma_2^\sharp)_{\Sigma,\Sigma'}.$$

Proof: The use of Lemma 5.2 followed by Lemma 5.3 to $\mathcal{D}$ yields a derivation structured as in Figure 6, from which moves over configuration can easily be read off (shown on the right of the schematic derivation). □

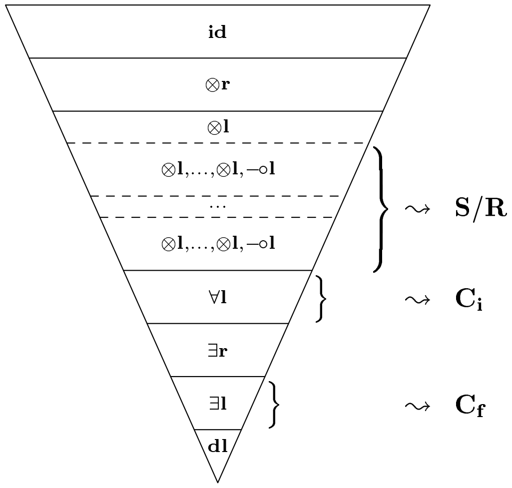


Figure 6: Completeness Argument

A fine analysis of this proof reveals that linear logic derivations enable a form of abstraction that move sequences do not achieve. Indeed, the tree structure of a derivation does not always impose a total order on independent transitions. This is a very mild form of non-determinism compared with the explicit concurrency present in bundles [7]. We expect to get a model closer to bundles by considering graph-based formulations of linear logic such as proof nets [16]. Furthermore, game-theoretic investigations of linear logic [2] have produced methods for obtaining very strong forms of completeness which could be relevant in this setting.

6 Interpreting Multiset Rewriting in Linear Logic

In this section, we briefly describe how multiset rewriting techniques can be conveniently used to express security protocols (Section 6.1). We then show how the resulting specification is translated into linear logic and state the expected correctness results (Section 6.2). Finally, we compare the linear logic expressions we derive from the strand and multiset rewriting specifications of a protocol (Section 6.3).

6.1 Multiset Rewriting for Cryptoprotocols

A *multiset* 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 M and N by taking their *multiset union*, denoted "M, N". The elements we will consider here will be first-order atomic formulas $A(\vec{t})$ over some signature.

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

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

$$
\begin{array}{lrcl}
 & & \underline{\text{Initiator}} & \\
\mathsf{r_{A0}}: & \pi_{\mathsf{A0}}(K_A, K_A^{-1}) & \longrightarrow & \mathsf{A}_0(K_A, K_A^{-1}), \pi_{\mathsf{A0}}(K_A, K_A^{-1}) \\
\mathsf{r_{A1}}: & \mathsf{A}_0(K_A, K_A^{-1}), \pi_{\mathsf{A1}}(K_B) & \longrightarrow & \exists N_A.\, \mathsf{A}_1(K_A, K_A^{-1}, K_B, N_A), \mathsf{N}(\{N_A, K_A\}_{K_B}), \pi_{\mathsf{A1}}(K_B) \\
\mathsf{r_{A2}}: & \mathsf{A}_1(K_A, K_A^{-1}, K_B, N_A), \mathsf{N}(\{N_A, N_B\}_{K_A}) & \longrightarrow & \mathsf{A}_2(K_A, K_A^{-1}, K_B, N_A, N_B) \\
\mathsf{r_{A3}}: & \mathsf{A}_2(K_A, K_A^{-1}, K_B, N_A, N_B) & \longrightarrow & \mathsf{A}_3(K_A, K_A^{-1}, K_B, N_A, N_B), \mathsf{N}(\{N_B\}_{K_B}) \\
 & & \underline{\text{Responder}} & \\
\mathsf{r_{B0}}: & \pi_{\mathsf{B0}}(K_B, K_B^{-1}) & \longrightarrow & \mathsf{B}_0(K_B, K_B^{-1}), \pi_{\mathsf{B0}}(K_B, K_B^{-1}) \\
\mathsf{r_{B1}}: & \mathsf{B}_0(K_B, K_B^{-1}), \mathsf{N}(\{N_A, K_A\}_{K_B}), \pi_{\mathsf{B1}}(K_A) & \longrightarrow & \mathsf{B}_1(K_A, K_B, K_B^{-1}, N_A), \pi_{\mathsf{B1}}(A) \\
\mathsf{r_{B2}}: & \mathsf{B}_1(K_A, K_B, K_B^{-1}, N_A) & \longrightarrow & \exists N_B.\, \mathsf{B}_2(K_A, K_B, K_B^{-1}, N_A, N_B), \mathsf{N}(\{N_A, N_B\}_{K_A}) \\
\mathsf{r_{B3}}: & \mathsf{B}_2(K_A, K_B, K_B^{-1}, N_A, N_B), \mathsf{N}(\{N_B\}_{K_B}) & \longrightarrow & \mathsf{B}_3(K_A, K_B, K_B^{-1}, N_A, N_B)
\end{array}
$$

$$
\text{where}\quad
\begin{array}{rcl}
\pi_{\mathsf{A0}}(K_A, K_A^{-1}) & = & \mathit{PubK}(K_A), \mathit{PrvK}(K_A, K_A^{-1}) \\
\pi_{\mathsf{A1}}(K_B) & = & \mathit{PubK}(K_B)
\end{array}
\qquad
\begin{array}{rcl}
\pi_{\mathsf{B0}}(K_B, K_B^{-1}) & = & \mathit{PubK}(K_B), \mathit{PrvK}(K_B, K_B^{-1}) \\
\pi_{\mathsf{B1}}(K_A) & = & \mathit{PubK}(K_A)
\end{array}
$$

Figure 7: Multiset Rewriting Specification of the Needham-Schroeder Protocol

where r is a label and $\exists\vec{n}$ indicates that the constants that instantiate $\vec{n}$ 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 M over a signature Σ, a rule $r:\ F(\vec{x}) \longrightarrow \exists\vec{n}.\, G(\vec{x}, \vec{n})$ is *applicable* if $M = F(\vec{t}), M'$, for terms $\vec{t}$. Then, *applying* r to M yields the multiset $N = G(\vec{t}, \vec{c}), M'$ where the constants $\vec{c}$ are fresh (in particular, they do not appear in M), $\vec{x}$ and $\vec{n}$ have been instantiated with $\vec{t}$ and $\vec{c}$ respectively, and the facts $F(\vec{t})$ in M have been replaced with $G(\vec{t}, \vec{c})$ to produce N. The new signature is $\Sigma, \vec{c}$. We denote the application of a single rule and of zero or more rewrite rules by means of the *one-step* and *multistep transition* judgments:

$$M_\Sigma \xrightarrow{r}_{\mathcal{R}} N_{\Sigma'} \qquad\qquad M_\Sigma \xrightarrow{\vec{r}}{}^{*}_{\mathcal{R}} N_{\Sigma'}$$

respectively, where Σ and Σ' are the signatures over which M and N are respectively defined. The labels r and $\vec{r}$ identify which rule(s) have been applied and the terms $\vec{t}$ used to instantiate $\vec{x}$. Thus, $\vec{r}$ acts as a complete trace of the execution.

We model protocols by means of specifically tailored multiset rewriting systems. We call this approach *MSR*. Without loss of generality, we consider here a slightly simplified version of the model introduced in [6, 12]. We rely upon the following atomic formulas:

Network messages: Network messages are modeled by the predicate $\mathsf{N}(m)$, where m is a message being transmitted.

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.

Intruder knowledge: The adversary's knowledge is held in a distributed way in facts of the form $\mathsf{I}(m)$, where m is some piece of information captured or fabricated by the intruder.

Persistent information: We express persistent information exactly as we did in the case of strands in Section 2, *i.e.* by means of a multiset Π of ground facts.

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

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

where, as in previous sections, $\pi(\vec{x})$ denotes a multiset of persistent atomic formulas that may mention variables among $\vec{x}$. 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:

$$
\begin{array}{lll}
\textbf{send:} & & \mathsf{A}_{\rho i}(\vec{x}), \pi(\vec{x}, \vec{z}) \\
 & \longrightarrow & \exists\vec{n}.\, \mathsf{A}_{\rho i+1}(\vec{x}, \vec{z}, \vec{n}), \mathsf{N}(m(\vec{x}, \vec{z}, \vec{n})), \pi(\vec{x}, \vec{z}) \\
\textbf{receive:} & & \mathsf{A}_{\rho i}(\vec{x}), \mathsf{N}(m(\vec{x}, \vec{y})), \pi(\vec{x}, \vec{y}, \vec{z}) \\
 & \longrightarrow & \mathsf{A}_{\rho i+1}(\vec{x}, \vec{y}, \vec{z}), \pi(\vec{x}, \vec{y}, \vec{z})
\end{array}
$$

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 in the received message $\vec{y}$ (*e.g.* the sender's public key) or previously known information $\vec{x}$. Situations where a principal both sends and receive 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. As an example, Figure 7 shows the encoding of our running example in the *MLR* notation.

The behavior of the intruder according to the Dolev-Yao model [11, 34] is similarly specified as a set of rewrite rules [6]. We will refer to them as $\mathcal{I}$. A *state* is then a multiset of ground facts $S = \Pi, A, N, I$, where A is a multiset of role states $\mathsf{A}_{\rho i}(\vec{t})$, N is multiset of messages $\mathsf{N}(m)$ currently in transit, and I summarizes the intruder's knowledge $\mathsf{I}(m)$. In particular the initial state is just Π, I_0, where I_0 contains the information (*e.g.* keys) initially known to the intruder.

6.2 Mapping to Linear Logic

The close affinity between multiset rewriting and simple fragments of linear logic has been known for a long time [3, 30, 15, 19, 5]. We extend this standard correspondence to take parameters and existentially quantified variables into consideration. A generic multiset M is mapped to the tensor product $\bigotimes M$ of its constituents, or $\mathbf{1}$ if M is empty. A multiset rewrite rule

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

is translated into the following linear logic formula, that we call $\ulcorner r \urcorner$:

$$\forall \vec{x}.\ \bigotimes F(\vec{x}) \multimap \exists \vec{n}.\ \bigotimes G(\vec{x}, \vec{n}).$$

The encoding $\ulcorner \mathcal{R} \urcorner$ of a set $\mathcal{R}$ of multiset rewriting rules is the union of the translation of its elements.

Given this simple encoding, multiset rewriting transitions correspond to linear logic derivations and reachability is mapped to derivability.

Theorem 6.1 *(Soundness and Completeness)*
Let $\mathcal{R}$ be a set of multiset rewriting rules, let S be a state over signature Σ, and let S' be a state over Σ and variables $\vec{n} = (n_1 ... n_k)$. If there is a transition sequence $\vec{r}$ such that

$$S_{\Sigma} \xrightarrow{\vec{r}}{}^{*}_{\mathcal{R}} [\vec{c}/\vec{n}] S'_{\Sigma, \vec{c}}$$

for some instantiation with distinct fresh constants $\vec{c} = (c_1 ... c_k)$, there exist a linear logic derivation $\mathcal{D}$ of the sequent

$$\ulcorner \mathcal{R} \urcorner;\ \bigotimes S \vdash_{\Sigma} \exists \vec{n}.\ \bigotimes S',$$

and vice versa. □

The proof of this result follows the pattern of theorems 5.1 and 5.4. In particular, the soundness part relies on a simple induction on the structure of the transition sequence $\vec{r}$. The completeness direction requires transforming the derivation $\mathcal{D}$ to a suitable normal form before extracting multiset rewriting rule applications.

6.3 Comparison

Strands were originally aimed at analyzing completed protocol runs in term of the observed causal interactions among the participants. Parametric strands, briefly described in Section 2 and fully investigated in [7], extend this framework with the possibility of giving executable specifications of security protocols. This same objective guided the design of *MSR*.

In [7], we established a substantial equivalence of these two formalisms: we devised a suitable abstraction of strand configurations that corresponds to *MSR* states, and showed that related pairs of states and configurations are equireachable. (Indeed, strand and *MSR* transitions induce an approximate bisimulation upon them. Were we to collapse the predicate symbols N and I and eliminate the *MSR* intruder rules that relate them, we would obtain an exact bisimulation.) This is relevant for security analysis because several properties of security protocols (*e.g.* secrecy) can be phrased as reachability problems.

Our results in Sections 5 and 6.2 show that both strand constructions and *MSR* can be expressed in linear logic in such a way that reachability corresponds to derivability. A close inspection of our translations reveals however substantial differences in the resulting formulas. First, a parametric strand is mapped to a single reusable linear logic formula, while the corresponding notion of role in *MSR* yields a separate multi-use clause for each message transmission or reception, plus one to account for role generation. Second, all quantifiers appear at the head of the translation of a parametric strand, while they are distributed among several clauses and possibly nested within connectives in the case of *MSR*. Standard linear logic equalities are insufficient to prove the equivalences of these mappings. In fact, these translations, although faithfully capturing corresponding behaviors, are not logically equivalent. This leaves us with following partially completed diagram:

$$\begin{tikzcd} \text{MSR} \arrow[rr, leftrightarrow, "{[7]}"] \arrow[d, leftrightarrow] & & \text{Strands} \arrow[d, leftrightarrow] \\ \text{LL} \arrow[rr, leftrightarrow, dashed, "?"'] & & \text{LL} \end{tikzcd}$$

Although *MSR* and strands agree on basic secrecy, they might possibly differ on more refined security properties such as perhaps Lowe's notion of agreement [27, 28] or Schneider's definition of precedence [37, 36]. We suspect that a fine analysis of the relationship between *MSR* and strands in the framework of linear logic may expose such differences.

7 Conclusions

This paper may be seen as a companion to [7], where we showed that as far as the basic secrecy property is concerned (more precisely, reachability), strand spaces [13] and multiset rewriting with existential quantification, *MSR* [6, 12] are equivalent settings for the Dolev-Yao model of security protocol analysis Multiset rewriting is known to be closely related to certain fragments of linear logic [3, 30, 15, 19, 5].

Another, direct representation of strand spaces in linear logic is introduced in this paper and shown to be sound and complete. Linear logic theories obtained by this encoding are not to logically equivalent to the linear logic theories related to *MSR*, in general. This raises the possibility that strand spaces and *MSR* might differ on complex properties of protocols beyond basic secrecy. We propose linear logic as an appropriate logical setting for expressing properties of protocols, motivated by a natural way in which linear logic deals with computational state.

References

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

[2] Abramsky, S., Jagadeesan, R., and Malacaria, P. Full abstraction for PCF. In *Proc. TACS '94* (1994), Springer-Verlag, LNCS 789, pp. 1–15.

[3] Asperti, A. A logic for concurrency. Manuscript, Nov. 1978.

[4] Burrows, M., Abadi, M., and Needham, R. A logic of authentication. *Proceedings of the Royal Society, Series A 426*, 1871 (1989), 233–271.

[5] Cervesato, I. Petri nets and linear logic: a case study for logic programming. In *Proceedings of the 1995 Joint Conference on Declarative Programming — GULP-PRODE'95* (Marina di Vietri, Italy, 11–14 September 1995), M. Alpuente and M. I. Sessa, Eds., Palladio Press, pp. 313–318.

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

[7] Cervesato, I., Durgin, N. A., Lincoln, P. D., Mitchell, J. C., and Scedrov, A. Relating strands and multiset rewriting for security protocol analysis. In *13th IEEE Computer Security Foundations Workshop — CSFW'00* (Cambrige, UK, 3–5 July 2000), P. Syverson, Ed., IEEE Computer Society Press. To appear; available at `www.cs.stanford.edu/~iliano/papers/csfw00.ps.gz`.

[8] Cervesato, I., and Pfenning, F. A linear logical framework. In *Proceedings of the Eleventh Annual Symposium on Logic in Computer Science — LICS'96* (New Brunswick, NJ, July 1996), E. Clarke, Ed., IEEE Computer Society Press, pp. 264–275.

[9] Chirimar, J. L. *Proof Theoretic Approach to Specification Languages.* PhD thesis, Department of Computer and Information Science, University of Pennsylvania, 1995.

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

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

[12] Durgin, N., Lincoln, P., Mitchell, J., and Scedrov, A. Undecidability of bounded security protocols. In *Proceedings of the Workshop on Formal Methods and Security Protocols — FMSP* (Trento, Italy, July 1999), N. Heintze and E. Clarke, Eds.

[13] Fábrega, F. J. T., Herzog, J. C., and Guttman, J. D. Strand spaces: Why is a security protocol correct? In *Proceedings of the 1998 IEEE Symposium on Security and Privacy* (Oakland, CA, May 1998), IEEE Computer Society Press, pp. 160–171.

[14] Fábrega, F. J. T., Herzog, J. C., and Guttman, J. D. Mixed strand spaces. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop — CSFW'99* (Mordano, Italy, June 1999), P. Syverson, Ed., IEEE Computer Society Press, pp. 72–82.

[15] Gehlot, V., and Gunter, C. Normal process representatives. In *Proc. 5-th IEEE Symp. on Logic in Computer Science* (Philadelphia, June 1990).

[16] Girard, J.-Y. Linear logic. *Theoretical Computer Science 50* (1987), 1–102.

[17] Hodas, J., and Miller, D. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation 110* (1994), 327–365.

[18] Kanovich, M., Okada, M., and Scedrov, A. Specifying real-time finite-state systems in linear logic. In *Proc. COTIC '98* (Nice, France, 1998), Electronic Notes in Theoretical Computer Science 16(1).

[19] Kanovich, M. I. Linear logic as a logic of computation. *Annals of Pure and Applied Logic 67*, 1–3 (May 1994), 183–212.

[20] Kobayashi, N., Shimizu, T., and Yonezawa, A. Distributed concurrent linear logic programming. *Theoretical Computer Science 227* (1999), 185–220.

[21] Lamport, L. Time, clocks and the ordering of events in a distributed system. *Communications of the ACM 21*, 7 (July 1978), 558–565.

[22] Lincoln, P., Mitchell, J., and Scedrov, A. Optimization complexity of linear logic proof games. *Theoretical Computer Science 227* (1999), 299–331.

[23] Lincoln, P., Mitchell, J., Scedrov, A., and Shankar, N. Decision problems for propositional linear logic. *Annals of Pure and Applied Logic 56* (1992), 239–311.

[24] Lincoln, P., Scedrov, A., and Shankar, N. Linearizing intuitionistic implication. *Annals of Pure and Applied Logic 60* (1993), 151–177.

[25] Lincoln, P., and Shankar, N. Proof search in first-order linear logic and other cut-free sequent calculi. In *Ninth Annual Symposium on Logic in Computer Science* (Paris, France, 1994), S. Abramsky, Ed., IEEE Computer Society Press, pp. 282–291.

[26] Lowe, G. An attack on the Needham-Schroeder public-key protocol. *Info. Proc. Letters 56* (1995), 131–133.

[27] Lowe, G. A hierarchy of authentication specifications. In *Proceedings of 10th IEEE Computer Security Foundations Workshop* (Rockport, MA, June 1997), IEEE Computer Society Press, pp. 31–43.

[28] Lowe, G. Casper: A compiler for the analysis of security protocols. *Journal of Computer Security 6* (1998), 53–84.

[29] Marrero, W., Clarke, E. M., and Jha, S. Model checking for security protocols. In *Proceedings of the 1997 DIMACS Workshop on Design and Formal Verification of Security Protocols* (1997).

[30] Martí-Oliet, N., and Meseguer, J. From Petri nets to linear logic. In *Conference on Category Theory and Computer Science* (1989), Springer-Verlag LNCS 389, pp. 313–337.

[31] Mazurkiewicz, A. Trace theory. In *Advances in Petri nets.* Springer-Verlag LNCS 255, 1986, pp. 279–324.

[32] McDowell, R., and Miller, D. A logic for reasoning with higher-order abstract syntax and induction. In *Proc. LICS'97* (Warsaw, Poland, 1997), IEEE Computer Society Press, pp. 434–445.

[33] Meadows, C. The NRL protocol analyzer: an overview. *J. Logic Programming 26*, 2 (1996), 113–131.

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

[35] Paulson, L. Proving properties of security protocols by induction. In *10th IEEE Computer Security Foundations Workshop* (1997), pp. 70–83.

[36] Schneider, S. Modelling security properties with CSP. Technical Report CSD-TR-96-04, Royal Holloway, University of London, 1996.

[37] Schneider, S. Verifying authentication protocols with CSP. In *Proceedings of 10th IEEE Computer Security Foundations Workshop* (Rockport, MA, June 1997), IEEE Computer Society Press, pp. 3–17.

[38] Shmatikov, V., and Stern, U. Efficient finite-state analysis for large security protocols. In *Proceedings of the 11th Computer Security Foundations Workshop* (Rockport, MA, 1998), IEEE Computer Society Press, pp. 106–115.

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

[40] Stal, D., Tavares, S., and Meijer, H. Backward state analysis of cryptographic protocols using coloured Petri nets. In *Proceedings of the 1994 Workshop on Selected Areas in Cryptography — SAC'94* (Kingston, Ontario, Canada, May 1994), pp. 107–118.