UNIVERSIDAD POLITÉCNICA DE VALENCIA

Depto. Sistemas Informáticos y Computación

Máster en Ingeniería del Software, Métodos Formales y Sistemas de Información

MASTER THESIS

Sequential Protocol Composition in Maude-NPA

CANDIDATE: Sonia Santiago Pinazo

SUPERVISOR: Santiago Escobar Román

September 22, 2010

Departamento de Sistemas Informáticos y Computación

Universidad Politécnica de Valencia

Camino de Vera, s/n

46022 Valencia, Spain

Contents

Abstract

Protocols do not work alone, but together, one protocol relying on another to provide needed services. Many of the problems in cryptographic protocols arise when such composition is done incorrectly or is not well understood. In this thesis, we discuss an extension to the current syntax and operational semantics of the Maude-NPA, a protocol specification and analysis tool, to support dynamic sequential composition of protocols, so that protocols can be specified separately and composed when desired. This allows one to reason about many different compositions with minimal changes to the specification. Moreover, we show that, by a simple protocol transformation, we are able to analyze and verify this dynamic composition in the current Maude-NPA without any modification to the tool. We prove soundness and completeness of the protocol transformation with respect to the extended operational semantics, for protocol composition. Finally, we illustrate our work on some examples of protocol composition and provide some experimental results.

Chapter 1

Introduction

It is well known that many problems in the security of cryptographic protocols arise when the protocols are composed. Protocols that work correctly in one environment may fail when they are composed with new protocols in new environments, either because the properties they guarantee are not quite appropriate for the new environment, or because the composition itself is mishandled.

The importance of understanding composition has long been acknowledged, and there are a number of logical systems that support it. The Protocol Composition Logic (PCL) begun in [Durgin 01] is probably the first protocol logic to approach composition in a systematic way. Logics such as the Protocol Derivation Logic (PDL) [Cervesato 05], and tools such as the Protocol Derivation Assistant (PDA) [Anlauff 06] and the Cryptographic Protocol Shape Analyzer (CPSA) [Doghim 07] also support reasoning about composition. All of these are logical systems and tools that support reasoning about the properties guaranteed by the protocols. One uses the logic to determine whether the properties guaranteed by the protocols are adequate. This is a natural way to approach composition, since one can use these tools to determine whether the properties guaranteed by one protocol are adequate for the needs of another protocol that relies upon it. Thus in [Datta 03] PCL and in [Guttman 01] the authentication tests methodology underlying CPSA are used to analyze key exchange standards and electronic commerce protocols, respectively, via composition out of simpler components.

Less attention has been given to handling composition when model checking protocols. However, model checking can provide considerable insight into the way composition succeeds or fails. Often the desired properties of a composed protocol can be clearly stated, while the properties of the components may be less well understood. Using a model checker to experiment with different compositions and their results help us to get a better idea of what the requirements on both the subprotocols and the compositions actually are.

The problem is in providing a specification and verification environment that supports composition. In general, it is tedious to hand-code compositions. This is especially the case when one protocol is composed with other protocols in several different ways. In this thesis we propose a syntax and operational semantics for sequential protocol composition in Maude-NPA [Escobar 06, Escobar 09a], a protocol specification and analysis tool based on unification and narrowing-based backwards search. Sequential composition, in which one or more child protocols make use of information obtained from running a parent protocol, is the most common use of composition in cryptographic protocols. We show that it is possible to incorporate it via a natural extension of the operational semantics of Maude-NPA. We have implemented this protocol composition semantics via a simple program transformation without any change to the tool. We prove the soundness and completeness of the transformation with respect to the semantics.

Plan of the Thesis

The rest of the thesis is organized as follows. In Chapter 2 we introduce two protocol compositions that we will use as running examples: one example of one-parent-one-child composition, and another of one-parent-many-children composition. We provide some preliminaries about term rewriting and narrowing in Chapter 3. In Chapter 4 we give an overview of the Maude-NPA tool and its operational semantics. In Chapter 5 we describe the current Maude-NPA syntax for protocol specification and analysis. In Chapter 6 we describe the new composition syntax and semantics. In Chapter 7 we describe the operational semantics of composition and the protocol transformation. The results of our experiments on the protocols used as running example in this thesis are discussed in Chapter 8. In Chapter 9 we give soundness and completeness results for the protocol transformation. In Chapter 10 we conclude the paper and discuss related and future work.

Chapter 2

Two Motivating Examples

In both of our examples of sequential protocol composition we build on the well-known Needham-Schroeder-Lowe (NSL) protocol [Lowe 96]. These two examples give a flavor for the variants of sequential composition that are used in constructing cryptographic protocols. A single parent instance can have either many children instances, or be constrained to only one. Likewise, parent roles can determine child roles, or child roles can be unconstrained. In this thesis, we will show how all these types of composition can be specified and analyzed in Maude-NPA, using these examples as running illustrations.

2.1 Distance Bounding Protocol (NSL-DB)

The first example of protocol composition, which appeared in [Guttman 08], is an example of a one-parent, one-child protocol, which is subject to an unexpected attack not noticed before. In this protocol, the participants use NSL to agree on a secret nonce. We reproduce the NSL protocol in the following.

1. $A \to B : \{N_A, A\}_{pk(B)}$
2. $B \to A : \{N_A, N_B, B\}_{pk(A)}$
3. $A \to B : \{N_B\}_{pk(B)}$

where $\{M\}_{pk(A)}$ means message M encrypted using the public key of principal with name A, N_A and N_B are nonces generated by the respective principals, and we use the comma as message concatenation.

The agreed nonce N_A is then used in a distance bounding protocol. This is a type of protocol, originally proposed by Desmedt [Desmedt 88] for smart cards, which has received new interest in recent years for its possible application in wireless environments [Capkun 06]. The idea behind the protocol is that Bob uses the round trip time of a challenge-response protocol with Alice to compute an upper bound on her distance from him. The protocol is described as follows:

4. $B \rightarrow A : N'_B$
   Bob records the time at which he sent N'_B
5. $A \rightarrow B : N_A \oplus N'_B$
   Bob records the time he receives the response and checks the equivalence $N_A = N_A \oplus N'_B \oplus N'_B$. If it is equal, he uses the round-trip time of his challenge and response to estimate his distance from Alice

where $\oplus$ is the exclusive-or operator satisfying associativity (i.e., $X \oplus (Y \oplus Z) = (X \oplus Y) \oplus Z$) and commutativity (i.e., $X \oplus Y = Y \oplus X$) plus the properties $X \oplus X = 0$ and $X \oplus 0 = X$. Note that Bob is the initiator and Alice is the responder of the distance bounding protocol, in contrast to the NSL protocol.

This protocol must satisfy two requirements. The first is that it must guarantee that $N_A \oplus N'_B$ was sent after N'_B was received, or Alice will be able to pretend that she is closer than she is. Note that if Alice and Bob do not agree on N_A beforehand, then Alice will be able to mount the following attack: $B \rightarrow A : N'_B$ and then $A \rightarrow B : N$. Of course, $N = N'_B \oplus X$ for some X. But Bob has no way of telling if Alice computed N using N'_B and X, or if she just sent a random N. Using NSL to agree on a $X = N_A$ in advance prevents this type of attack.

Bob also needs to know that the response comes from whom it is supposed to be. In particular, an attacker should not be able to impersonate Alice.

Using NSL to agree on N_A guarantees that only Alice and Bob can know N_A, so the attacker cannot impersonate Alice. However, it should also be the case that an attacker cannot pass off Alice's response as his own. However, this is not the case for the NSL distance bounding protocol, which is subject to the following attack[1]:

a) Intruder I runs an instance of NSL with Alice as the initiator and I as the responder, obtaining a nonce N_A.

b) I then runs an instance of NSL with Bob with I as the initiator and Bob as the responder, using N_A as the initiator nonce.

c) $B \rightarrow I : N'_B$ where I does not respond, but Alice, seeing this, thinks it is for her.

d) $A \rightarrow I : N'_B \oplus N_A$ where Bob, seeing this thinks this is I's response.

If Alice is closer to Bob than I is, then I can use this attack to appear closer to Bob than he is. This attack is a textbook example of a composition failure. NSL has all the properties of a good key distribution protocol, but fails to provide all the guarantees that are needed by the distance bounding protocol. However, in this case we can fix the problem, not by changing NSL, but by changing the distance bounding protocol so that it provides a stronger guarantee:

4. $B \rightarrow A : N'_B$
5. $A \rightarrow B : h(N_A, A) \oplus N'_B$ where h is a collision-resistant hash function.

As we show in our analysis in Chapter 8, this prevents the attack. I cannot pass off Alice's nonce as his own because it is now bound to her name.

The distance bounding example is a case of a one parent, one child protocol composition. Each instance of the parent NSL protocol can have only one child distance bounding protocol, since the distance bounding protocol

[1]This is not meant as a denigration of [Guttman 08], whose main focus is on timing models in strand spaces, not the design of distance bounding protocols.

depends upon the assumption that N_A is known only by A and B. But because the distance bounding protocol reveals N_A, it cannot be used with the same N_A more than once.

2.2 Key Distribution Protocol (NSL-KD)

Our next example is a one parent, many children composition, also using NSL. This type of composition arises, for example, in key distribution protocols in which the parent protocol is used to generate a master key, and the child protocol is used to generate a session key. In this case, one wants to be able to run an arbitrary number of child protocols.

In the distance bounding example the initiator of the distance bounding protocol was always the child of the responder of the NSL protocol and vice versa. In the key distribution example, the initiator of the session key protocol can be the child of either the initiator or responder of the NSL protocol. So, we have two possible child executions after NSL:

4. $A \rightarrow B : \{Sk_A\}_{h(N_A,N_B)}$	4. $B \rightarrow A : \{Sk_B\}_{h(N_A,N_B)}$
5. $B \rightarrow A : \{Sk_A; N'_B\}_{h(N_A,N_B)}$	5. $A \rightarrow B : \{Sk_B; N'_A\}_{h(N_A,N_B)}$
6. $A \rightarrow B : \{N'_B\}_{h(N_A,N_B)}$	6. $B \rightarrow A : \{N'_A\}_{h(N_A,N_B)}$

where Sk_A is the session key generated by principal A and h is again a collision-resistant hash function.

As we show in our analysis in Chapter 8, this protocol guarantees that a dishonest principal is not able to learn the secret key of an honest principal.

Chapter 3

Background on Term Rewriting

We follow the classical notation and terminology from [TeReSe 03] for term rewriting and from [Meseguer 92, Meseguer 98] for rewriting logic and order-sorted notions. We assume an *order-sorted signature* Σ with a finite poset of sorts $(\mathsf{S}, \leq)$ and a finite number of function symbols. We assume an S-sorted family $\mathcal{X} = \{\mathcal{X}_\mathsf{s}\}_{\mathsf{s}\in\mathsf{S}}$ of disjoint variable sets with each $\mathcal{X}_\mathsf{s}$ countably infinite. $\mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ denotes the set of terms of sort s, and $\mathcal{T}_{\Sigma,\mathsf{s}}$ the set of ground terms of sort s. We write $\mathcal{T}_\Sigma(\mathcal{X})$ and $\mathcal{T}_\Sigma$ for the corresponding term algebras. We write $\mathcal{V}ar(t)$ for the set of variables present in a term t. The set of positions of a term t is written $Pos(t)$, and the set of non-variable positions $Pos_\Sigma(t)$. The subterm of t at position p is $t|_p$, and $t[u]_p$ is the result of replacing $t|_p$ by u in t. A *substitution* σ is a sort-preserving mapping from a finite subset of $\mathcal{X}$ to $\mathcal{T}_\Sigma(\mathcal{X})$.

A Σ-*equation* is an unoriented pair $t = t'$, where $t \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$, $t' \in \mathcal{T}_\Sigma(\mathcal{X})_{\mathsf{s}'}$, and s and s' are sorts in the same connected component of the poset $(\mathsf{S}, \leq)$. Given a set E of Σ-equations, order-sorted equational logic induces a congruence relation $=_E$ on terms $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})$ (see [Meseguer 98]). Throughout this paper we assume that $\mathcal{T}_{\Sigma,\mathsf{s}} \neq \emptyset$ for every sort s. We denote the E-equivalence class of a term $t \in \mathcal{T}_\Sigma(\mathcal{X})$ as $[t]_E$ and the E-equivalence classes of all terms $\mathcal{T}_\Sigma(\mathcal{X})$ and $\mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ as $\mathcal{T}_{\Sigma/E}(\mathcal{X})$ and $\mathcal{T}_{\Sigma/E}(\mathcal{X})_\mathsf{s}$, respectively.

For a set E of Σ-equations, an E-*unifier* for a Σ-equation $t = t'$ is a substitution σ s.t. $\sigma(t) =_E \sigma(t')$. A *complete* set of E-unifiers of an equation

$t = t'$ is written $CSU_E(t = t')$. We say $CSU_E(t = t')$ is *finitary* if it contains a finite number of E-unifiers.

A *rewrite rule* is an oriented pair $l \to r$, where $l \notin \mathcal{X}$ and $l, r \in \mathcal{T}_\Sigma(\mathcal{X})_{\mathsf{s}}$ for some sort $\mathsf{s} \in \mathsf{S}$. An *(unconditional) order-sorted rewrite theory* is a triple $\mathcal{R} = (\Sigma, E, R)$ with Σ an order-sorted signature, E a set of Σ-equations, and R a set of rewrite rules. A *topmost rewrite theory* (Σ, E, R) is a rewrite theory s.t. for each $l \to r \in R$, $l, r \in \mathcal{T}_\Sigma(\mathcal{X})_{\mathsf{State}}$ for a top sort State, $r \notin \mathcal{X}$, and no operator in Σ has State as an argument sort.

The rewriting relation $\to_R$ on $\mathcal{T}_\Sigma(\mathcal{X})$ is $t \xrightarrow{p}_R t'$ (or $\to_R$) if $p \in Pos_\Sigma(t)$, $l \to r \in R$, $t|_p = \sigma(l)$, and $t' = t[\sigma(r)]_p$ for some σ. The relation $\to_{R/E}$ on $\mathcal{T}_\Sigma(\mathcal{X})$ is $=_E; \to_R; =_E$, i.e., $t \to_{R/E} s$ iff $\exists u_1, u_2 \in \mathcal{T}_\Sigma(\mathcal{X})$ s.t. $t =_E u_1 \to_R u_2 =_E s$. Note that $\to_{R/E}$ on $\mathcal{T}_\Sigma(\mathcal{X})$ induces a relation $\to_{R/E}$ on $\mathcal{T}_{\Sigma/E}(\mathcal{X})$ by $[t]_E \to_{R/E} [t']_E$ iff $t \to_{R/E} t'$.

When $\mathcal{R} = (\Sigma, E, R)$ is a topmost rewrite theory, we can safely restrict ourselves to the rewriting relation $\to_{R,E}$ on $\mathcal{T}_\Sigma(\mathcal{X})$, where the rewriting relation $\to_{R,E}$ on $\mathcal{T}_\Sigma(\mathcal{X})$ is $t \xrightarrow{p}_{R,E} t'$ (or $\to_{R,E}$) if $p \in Pos_\Sigma(t)$, $l \to r \in R$, $t|_p =_E \sigma(l)$, and $t' = t[\sigma(r)]_p$ for some σ. Note that $\to_{R,E}$ on $\mathcal{T}_\Sigma(\mathcal{X})$ induces a relation $\to_{R,E}$ on $\mathcal{T}_{\Sigma/E}(\mathcal{X})$ by $[t]_E \to_{R,E} [t']_E$ iff $\exists w \in \mathcal{T}_\Sigma(\mathcal{X})$ s.t. $t \to_{R,E} w$ and $w =_E t'$.

The narrowing relation $\leadsto_R$ on $\mathcal{T}_\Sigma(\mathcal{X})$ is $t \overset{p}{\leadsto}_{\sigma,R} t'$ (or $\leadsto_{\sigma,R}$, $\leadsto_R$) if $p \in Pos_\Sigma(t)$, $l \to r \in R$, $\sigma \in CSU_\emptyset(t|_p = l)$, and $t' = \sigma(t[r]_p)$. Assuming that E has a finitary and complete unification algorithm, the narrowing relation $\leadsto_{R,E}$ on $\mathcal{T}_\Sigma(\mathcal{X})$ is $t \overset{p}{\leadsto}_{\sigma,R,E} t'$ (or $\leadsto_{\sigma,R,E}$, $\leadsto_{R,E}$) if $p \in Pos_\Sigma(t)$, $l \to r \in R$, $\sigma \in CSU_E(t|_p = l)$, and $t' = \sigma(t[r]_p)$.

The use of topmost rewrite theories provides several advantages (see [Thati 07]): (i) as pointed out above the relation $\to_{R,E}$ achieves the same effect as the relation $\to_{R/E}$, (ii) also the narrowing relation $\leadsto_{R,E}$ achieves the same effect as a more general narrowing relation $\leadsto_{R/E}$, and (iii) we obtain a completeness result between narrowing ($\leadsto_{R,E}$) and rewriting ($\to_{R/E}$).

Chapter 4

Maude-NPA's Execution Model

Given a protocol $\mathcal{P}$, we should first explain how its states are modeled algebraically. The key idea is to model such states as elements of an initial algebra $T_{\Sigma_{\mathcal{P}}/E_{\mathcal{P}}}$, where $\Sigma_{\mathcal{P}}$ is the signature defining the sorts and function symbols for the cryptographic functions and for all the state constructor symbols and $E_{\mathcal{P}}$ is a set of equations specifying the *algebraic properties* of the cryptographic functions and the state constructors. Therefore, a state is an $E_{\mathcal{P}}$-equivalence class $[t] \in T_{\Sigma_{\mathcal{P}}/E_{\mathcal{P}}}$ with t a ground $\Sigma_{\mathcal{P}}$-term. However, since the number of states $T_{\Sigma_{\mathcal{P}}/E_{\mathcal{P}}}$ is in general infinite, rather than exploring concrete protocol states $[t] \in T_{\Sigma_{\mathcal{P}}/E_{\mathcal{P}}}$ we explore *symbolic state patterns* $[t(x_1, \ldots, x_n)] \in T_{\Sigma_{\mathcal{P}}/E_{\mathcal{P}}}(X)$ on the free $(\Sigma_{\mathcal{P}}, E_{\mathcal{P}})$-algebra over a set of variables X. In this way, a state pattern $[t(x_1, \ldots, x_n)]$ represents not a single concrete state but a possibly infinite set of such states, namely all the instances of the pattern $[t(x_1, \ldots, x_n)]$ where the variables $x_1, \ldots, x_n$ have been instantiated by concrete ground terms.

In the Maude-NPA [Escobar 06, Escobar 09a], a *state* in the protocol execution is a term t of sort State, $t \in T_{\Sigma_{\mathcal{P}}/E_{\mathcal{P}}}(X)_{\mathsf{State}}$. A state is a multiset built by an associative and commutative union operator $\_\&\_$ with identity operator $\emptyset$. Each element in the multiset can be a strand or the intruder knowledge at that state.

A *strand* [Fabrega 99] represents the sequence of messages sent and received by a principal executing the protocol and is indicated by a sequence of messages[1] $[msg_1^{\pm},\ msg_2^{\pm},\ msg_3^{\pm}, \ldots,\ msg_{k-1}^{\pm},\ msg_k^{\pm}]$ where each msg_i is a term of sort Msg (i.e., $msg_i \in T_{\Sigma_{\mathcal{P}}/E_{\mathcal{P}}}(X)_{\mathsf{Msg}}$), msg^- represents an input message, and msg^+ represents an output message. For each positive message msg_i in a sequence of messages $[msg_1^{\pm},\ msg_2^{\pm},\ msg_3^{\pm}, \ldots,\ msg_i^{+}, \ldots,\ msg_{k-1}^{\pm},\ msg_k^{\pm}]$ the non-fresh variables (see below) occurring in an output message msg_i^+ must appear in previous messages $msg_1, msg_2, msg_3, \ldots, msg_{i-1}$. In Maude-NPA, strands evolve over time and thus we use the symbol $|$ to divide past and future in a strand, i.e., $[nil, msg_1^{\pm}, \ldots, msg_{j-1}^{\pm} \mid msg_j^{\pm}, msg_{j+1}^{\pm}, \ldots, msg_k^{\pm}, nil]$ where $msg_1^{\pm}, \ldots, msg_{j-1}^{\pm}$ are the past messages, and $msg_j^{\pm}, msg_{j+1}^{\pm}, \ldots, msg_k^{\pm}$ are the future messages ($msg_j^{\pm}$ is the immediate future message). The nils are present so that the bar may be placed at the beginning or end of the strand if necessary. A strand $[msg_1^{\pm}, \ldots, msg_k^{\pm}]$ is a shorthand for $[nil \mid msg_1^{\pm}, \ldots, msg_k^{\pm}, nil]$. We often remove the nils for clarity, except when there is nothing else between the vertical bar and the beginning or end of a strand. We write $\mathcal{P}$ for the set of strands in a protocol, including the strands that describe the intruder's behavior.

The *intruder knowledge* is represented as a multiset of facts unioned together with an associative and commutativity union operator _,_ with identity operator $\emptyset$. There are two kinds of intruder facts: positive knowledge facts (the intruder knows m, i.e., $m{\in}\mathcal{I}$), and negative knowledge facts (the intruder *does not yet know* m but *will know it in a future state*, i.e., $m{\notin}\mathcal{I}$), where m is a message expression.

Maude-NPA uses a special sort Msg of messages that allows the protocol specifier to describe other sorts as subsorts of the top sort Msg. The specifier can make use of a special sort Fresh in the protocol-specific signature Σ for representing fresh unguessable values, e.g., nonces. The meaning of a variable of sort Fresh is that it will never be instantiated by an E-unifier generated during the backwards reachability analysis. This ensures that if two nonces are represented using different variables of sort Fresh, they will never be identified and no approximation for nonces is necessary. We make explicit the Fresh variables $r_1, \ldots, r_k (k \geq 0)$ generated by a strand by writing

[1]We write $m^{\pm}$ to denote m^+ or m^-, indistinctively. We often write $+(m)$ and $-(m)$ instead of m^+ and m^-, respectively.

$:: r_1, \ldots, r_k ::\ [msg_1^{\pm}, \ldots, msg_n^{\pm}]$, where each r_i appears first in an output message $msg_{j_i}^{+}$ and can later be used in any input and output message of $msg_{j_i+1}^{\pm}, \ldots, msg_n^{\pm}$. Fresh variables generated by a strand are unique to that strand.

Let us recall our specification of the NSL protocol for motivation. The strands associated to the three protocol steps above are given next; having two strands, one for each principal in the protocol. Note that the first message $A \to B : \{N_A, A\}_{pk(B)}$ is represented by a message in Alice's strand sending $pk(B, n(A,r); A)^{+}$ and another message in Bob's strand receiving $pk(B, N; A)^{-}$. When a principal cannot observe the contents of a concrete part of a received message (e.g., because a key is necessary to look inside), we use a generic variable for such part of the message in the strand (as with variable N of sort *Nonce* above). We encourage the reader to compare the protocol in strand notation to the presentation of the protocol above.

- (Alice) $:: r ::\ [\ pk(B, n(A,r); A)^{+}, pk(A, n(A,r); N; B)^{-}, pk(B, N)^{+}]$
- (Bob) $:: r' ::\ [\ pk(B, N; A)^{-}, pk(A, N; n(B,r'); B)^{+}, pk(B, n(B,r'))^{-}]$

Note that r, r' are variables of sort Fresh used for nonce generation (they are special variables handled as *unique constants* in order to obtain an infinite number of available constants, see [Escobar 09a]).

There are also strands for initial intruder knowledge and intruder actions, such as concatenation, deconcatenation, encryption, decryption, etc. For example, concatenation by the intruder is described by the following strand $:: nil ::\ [(X)^{-}, (Y)^{-}, (X; Y)^{+}]$.

Our protocol analysis methodology is then based on the idea of *backward reachability analysis*, where we begin with one or more state patterns corresponding to *attack states*, and want to prove or disprove that they are *unreachable* from the set of initial protocol states. In order to perform such a reachability analysis we must describe how states change as a consequence of principals performing protocol steps and of intruder actions. This can be done by describing such state changes by means of a set $R_{\mathcal{P}}$ of *rewrite rules*, so that the rewrite theory $(\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, R_{\mathcal{P}})$ characterizes the behavior of protocol $\mathcal{P}$ modulo the equations $E_{\mathcal{P}}$. In the case in which new strands are not introduced into the state, the rewrite rules $R_{\mathcal{P}}$ obtained from the protocol strands $\mathcal{P}$ are as follows[2], where L, L_1, L_2 denote lists of input and output messages ($+m$,$-m$), IK, IK' denote sets of intruder facts ($m{\in}\mathcal{I}$,$m{\notin}\mathcal{I}$), and SS, SS' denote sets of strands:

$$[L \mid M^-, L'] \;\&\; SS \;\&\; (M{\in}\mathcal{I}, IK) \to [L, M^- \mid L'] \;\&\; SS \;\&\; (M{\in}\mathcal{I}, IK) \tag{4.1}$$
$$[L \mid M^+, L'] \;\&\; SS \;\&\; IK \to [L, M^+ \mid L'] \;\&\; SS \;\&\; IK \tag{4.2}$$
$$[L \mid M^+, L'] \;\&\; SS \;\&\; (M{\notin}\mathcal{I}, IK) \to [L, M^+ \mid L'] \;\&\; SS \;\&\; (M{\in}\mathcal{I}, IK) \tag{4.3}$$

In a *forward execution* of the protocol strands, Rule (4.1) synchronizes an input message with a message already in the channel (i.e., learned by the intruder), Rule (4.2) accepts output messages but the intruder's knowledge is not increased, and Rule (4.3) accepts output messages and the intruder's knowledge is positively increased. Note that Rule (4.3) makes explicit *when* the intruder learned a message M, which is recorded in the previous state by the negative fact $M{\notin}\mathcal{I}$. A fact $M{\notin}\mathcal{I}$ can be paraphrased as: "the intruder does not yet know M, but will learn it in the future". New strands are added to the state by explicit introduction through dedicated rewrite rules (one for each honest or intruder strand). It is also the case that when we are performing a backwards search, only the strands that we are searching for are listed explicitly, and extra strands necessary to reach an initial state are dynamically added. Thus, when we want to introduce new strands into the explicit description of the state, we need to describe additional rules for doing that, as follows:

$$\text{for each } [\ l_1,\ u^+,\ l_2\ ] \in \mathcal{P} : [\ l_1 \mid u^+, l_2\ ] \,\&\, SS\ \ \&\, (u{\notin}\mathcal{I}, IK) \to SS\ \ \&\, (u{\in}\mathcal{I}, IK) \tag{4.4}$$

where u denotes a message, l_1, l_2 denote lists of input and output messages ($+m$,$-m$), IK denotes a set of intruder facts ($m{\in}\mathcal{I}$,$m{\notin}\mathcal{I}$), and SS denotes a

[2]To simplify the exposition, we omit the fresh variables at the beginning of each strand in a rewrite rule.

set of strands. For example, intruder concatenation of two learned messages is described as follows:

$$[M_1^-, M_2^- \mid (M_1; M_2)^+] \;\&\; SS \;\&\; ((M_1; M_2) {\notin} \mathcal{I}, IK) \rightarrow SS \;\&\; ((M_1; M_2) {\in} \mathcal{I}, IK)$$

and it can be understood, in a backwards search, as "in the current state the intruder is able to learn a message that matches the pattern $M_1; M_2$ if he is able to learn message M_1 and message M_2 in prior states". In summary, for a protocol $\mathcal{P}$, the set of rewrite rules obtained from the protocol strands that are used for backwards narrowing reachability analysis *modulo* the equational properties $E_{\mathcal{P}}$ is $R_{\mathcal{P}} = \{(4.1), (4.2), (4.3)\} \cup (4.4)$.

The way to analyze *backwards* reachability is then relatively easy, namely to run the protocol "in reverse." This can be achieved by using the set of rules $R_{\mathcal{P}}^{-1}$, where $v \longrightarrow u$ is in $R_{\mathcal{P}}^{-1}$ iff $u \longrightarrow v$ is in $R_{\mathcal{P}}$. Reachability analysis can be performed *symbolically*, not on concrete states but on symbolic state patterns $[t(x_1, \ldots, x_n)]_{E_{\mathcal{P}}}$ by means of *narrowing modulo* $E_{\mathcal{P}}$ (see Chapter 3).

Chapter 5

Current Syntax for Protocol Specification

In this chapter, we briefly describe how to specify a protocol and all its relevant items in the current version of the Maude-NPA. For further information we refer the reader to [Escobar 09b]. Note that, since we are using Maude also as the specification language, each declaration has to be ended by a space and a period.

5.1 Specifying the Protocol Syntax

We begin by specifying *sorts*. In general, sorts are used to specify different types of data, that are used for different purposes. The Maude-NPA tool always assumes that the sort Msg is the top sort, but it allows user-defined subsorts of Msg that can be specified by the user for a more accurate protocol specification and analysis.

To illustrate the definition of sorts, we use the NSL protocol described in Chapter 2. For this protocol we need to define sorts to distinguish names, nonces and encrypted data. This is specified as follows:

```
sorts Name Nonce Enc .
subsort Name Nonce Enc < Msg . subsort Name < Public .
```

We can now specify the different operators needed in NSL[1]. A nonce generated by principal A is denoted by $n(A, r)$, where r is a unique variable of sort Fresh. Concatenation of two messages, e.g., N_A and N_B, is denoted by the operator $\_;\_$, e.g., $n(A, r)\ ;\ n(B, r')$. Encryption of a message M with the public key K_A of principal A is denoted by $pk(A, M)$, e.g., $\{N_B\}_{pk(B)}$ is denoted by $pk(B, n(B, r'))$. Encryption with a secret key is denoted by $sk(A, M)$. We begin with the public/private encryption operators.

```
op pk : Name Msg -> Enc .
op sk : Name Msg -> Enc .
```

Next we specify some principal names. For NSL, we have three constants of sort `Name`, `a` (for "Alice"), `b` (for "Bob"), and `i` (for the "Intruder"). We now need two more operators, one for nonces, and one for concatenation. The nonce operator is specified as follows.

```
op n : Name Fresh -> Nonce .
```

Note that the nonce operator has an argument of sort `Fresh` to ensure uniqueness. The argument of type `Name` is not strictly necessary, but it provides a convenient way of identifying which nonces are generated by which principal. This makes searches more efficient, since it allows us to keep track of the originator of a nonce throughout a search. Finally, we come to the message concatenation operator. In Maude-NPA, we specify concatenation via an infix operator "`_;_`" defined as follows:

```
op _;_ : Msg  Msg  -> Msg [gather (e E)] .
```

5.2 Algebraic Properties

Next, we specify the algebraic properties of the symbols defined above for the NSL protocol. There are two types of algebraic properties in Maude-NPA: (i) *equational axioms*, such as commutativity, or associativity-commutativity,

[1]We omit the Maude `frozen` attribute of operators, see [Escobar 09b].

called *axioms*, and (ii) *equational rules*, called *equations*. Axioms are specified within the operator declarations whereas equations are specified separately.

An equation is oriented into a rewrite rule in which the left–hand side of the equation is *reduced* to the right–hand side[2]. In NSL, we use two equations specifying the relationship between public and private key encryption, as follows:

```
var X : Msg . var A : Name .
eq pk(A,sk(A,X)) = X .
eq sk(A,pk(A,X)) = X .
```

Note that there are restrictions on the equations that can be included here, since the narrowing-based unification algorithm provided by the tool for those equations must be finitary, see [Escobar 09b]. For instance, the exclusive-or symbol of the DB protocol is specified as follows:

```
eq X * X * Y = Y .
eq X * X = null .
eq X * null = X .
```

Note that the redundant equational property $X * X * Y = Y$ is necessary in Maude-NPA for coherence purposes; see [Escobar 09a].

5.3 Specifying the Strands

As explained in Chapter 4, the protocol itself and the intruder capabilities are both specified using strands. We use the keyword STRANDS-PROTOCOL for storing the principal strands:

```
eq STRANDS-PROTOCOL =
 :: r ::
  [nil | +(pk(B,n(A,r) ; A)), -(pk(A,n(A,r) ; NB ; B )),
         +(pk(B,NB)), nil]
 &
```

[2]We omit the Maude nonexec attribute of equations, see [Escobar 09b].

```
:: r ::
 [nil | -(pk(B,NA ; A)), +(pk(A,NA ; n(B,r) ; B)),
        -(pk(B,n(B,r))), nil] .
```

The next thing to specify is the actions of the intruder, or Dolev-Yao rules [Dolev 83]. These specify the operations an intruder can perform. Each such action can be specified by an intruder strand consisting of a sequence of negative nodes, followed by a single positive node. If the intruder can (non-deterministically) find more than one term as a result of performing one operation (as in deconcatenation), we specify each of these outcomes by separate strands. Every operation that can be performed by the intruder, and every term that is initially known by the intruder, should have a corresponding intruder strand. For each operation used in the protocol we should consider whether or not the intruder can perform it, and specify a corresponding intruder strand that describes the conditions under which the intruder can perform it. For the NSL protocol, we have four operations: encryption with a public key (pk), decryption with a private key (sk), concatenation (_;_), and deconcatenation. We use the keyword STRANDS-DOLEVYAO for storing the principal strands:

```
eq STRANDS-DOLEVYAO
 = :: nil :: [ nil | -(X), -(Y), +(X ; Y), nil ] &
   :: nil :: [ nil | -(X ; Y), +(X), nil ] &
   :: nil :: [ nil | -(X ; Y), +(Y), nil ] &
   :: nil :: [ nil | -(X), +(sk(i,X)), nil ] &
   :: nil :: [ nil | -(X), +(pk(Ke,X)), nil ] .
```

5.4 Protocol Analysis

Next, we describe how to analyze a protocol in practice. First, we explain how a protocol state looks like, and how an attack state is specified in the protocol. Then, we explain how the actual protocol analysis is performed.

In Maude-NPA, each state associated to the protocol execution (i.e., a backwards search) is represented by a term with four different components separated by the symbol `||` in the following order: (1) the set of current strands, (2) the current intruder knowledge, (3) the sequence of messages encountered so far in the backwards execution, (4) some auxiliary data, and (5) the "never pattern", a technique to reduce the search space, associated to that state,

Strands `||` Intruder Knowledge `||` Message Sequence `||` Auxiliary Data `||` Never Pattern.

The first component, the set of current strands, indicates in particular how advanced each strand is in the execution process (by the placement of the bar). The second component contains messages that the intruder already knows (we use symbol `_inI` for the notation $m {\in} \mathcal{I}$) and messages that the intruder currently doesn't know (we use symbol `_!inI` for the notation $m {\notin} \mathcal{I}$) but will learn in the future. The third, fourth, and fifth components are irrelevant for the purposes of this work, see [Escobar 09b].

An initial state is the final result of the backwards reachability process and is described as follows:

1. in an initial state, all strands have the bar at the beginning, i.e., all strands are of the form $:: r_1, \ldots, r_j :: [\ nil \mid m_1{}^{\pm},\ \ldots,\ m_k{}^{\pm}\ ]$;

2. in an initial state, all the intruder knowledge is negative, i.e., all the items in the intruder knowledge are of the form $m {\notin} \mathcal{I}$.

From an initial state, no further backwards reachability steps are possible.

Attack states describe not just single concrete attacks, but *attack patterns* (or if you prefer *attack situations*), which are specified symbolically as terms (with variables) whose instances are the final attack states we are looking for. Given an attack pattern, Maude-NPA tries to either find an instance of the attack or prove that no instance of such attack pattern is possible. We can specify more than one attack state. Thus, we designate each attack state with a natural number.

When specifying an attack state, the user should specify only the first two components of the attack state: (i) a set of strands expected to appear in the attack, and (ii) some positive intruder knowledge. The message sequence, auxiliary data components, and never pattern should have just the empty symbol `nil`.

Note that the attack state is indeed a term with variables but the user does not have to provide the variables denoting "the remaining strands", "the remaining intruder knowledge", and the two variables for the two last state components. These variables are symbolically inserted by the tool.

For example, to prove that the NSL protocol fixes the bug found in the Needham-Schroeder Public Key protocol (NSPK), i.e., the intruder cannot learn the nonce generated by Bob, we should specify the following attack state:

```
eq ATTACK-STATE(0) =
  :: r ::
    [nil, -(pk(b,a ; NA)), +(pk(a,NA ; n(b,r) ; b)),
          -(pk(b,n(b,r))) | nil]
    || n(b,r) inI
    || nil
    || nil
    || nil .
```

which cannot reach an initial state and has a finite search space, proving it secure.

Chapter 6

Syntax for Protocol Specification and Composition

We begin by describing the new syntactic features we need to make explicit in each protocol to later define sequential protocol compositions. Each strand in a protocol specification in the Maude-NPA is now extended with *input parameters* and *output parameters*. Input parameters are a sequence of variables of different sorts placed at the beginning of a strand. Output parameters are a sequence of terms placed at the end of a strand. Any variable contained in an output parameter must appear either in the body of the strand, or as an input parameter. The strand notation we will now use is $[\{\overrightarrow{I}\}, \overrightarrow{M}, \{\overrightarrow{O}\}]$ where $\overrightarrow{I}$ is a list of input parameter variables, $\overrightarrow{M}$ is a list of positive and negative terms in the strand notation of the Maude-NPA, and $\overrightarrow{O}$ is a list of output terms all of whose variables appear in $\overrightarrow{M}$ or $\overrightarrow{I}$. The input and output parameters describe the exact assumptions about each principal.

In the following, we first describe our syntax for protocol specification and then introduce a new syntax for protocol composition. Similarly to the Maude syntax for modules, we define a protocol modularly as follows:

```
prot Name is
    sorts Sorts .
    subsorts Subsorts .
    Operators
    Variables
    Equations
    DYStrands
    Strands
endp
```

where *Name* is a valid Maude module name, *Sorts* is a valid Maude-NPA declaration of sorts, *Subsorts* is a valid Maude-NPA declaration of subsorts, *Operators* is a valid Maude-NPA declaration of operators, *Variables* is a valid Maude-NPA declaration of variables to be used in the equational properties and in the honest and Dolev-Yao strands, *Equations* is a valid Maude-NPA declaration of equational properties, *DYStrands* is a sequence of valid Maude-NPA Dolev-Yao strands, each starting with the word `DYstrand` and ending with a period, and *Strands* is a sequence of valid Maude-NPA strands, each starting with the word `strand` and ending with a period.

In the following, we provide as an example, the specification of the three protocols of Chapter 2 using the new syntax proposed above. By comparing with the example specification of Chapter 5 the reader can note that the new syntax we propose is just slightly different from the current one but is more expressive for our purpose of dealing with sequential protocol composition in the Maude-NPA.

Example 1 The following description of the NSL protocol contains more technical details than the informal description of NSL in Chapter 2.

```
prot NSL is
  --- Sort Information
  sorts Name Nonce Enc .
  subsort Name Nonce Enc < Msg .
  subsort Name < Public .
  --- Public/private encryption
  op pk : Name Msg -> Enc . op sk : Name Msg -> Enc .
  --- Principals
  ops a b i : -> Name . --- Alice Bob Intruder
  --- Nonce operator
  op n : Name Fresh -> Nonce .
  --- Concatenation operator
  op _;_ : Msg Msg -> Msg [gather (e E)] .
  --- Variables
  vars X Y : Msg . var r : Fresh .
  vars A B : Name . var N : Nonce .
  --- Encryption/Decryption Cancellation
  eq pk(A,sk(A,X)) = X .
  eq sk(A,pk(A,X)) = X .
  --- Dolev-Yao Strands
  DYstrand :: nil :: [nil | -(X), -(Y), +(X ; Y), nil] .
  DYstrand :: nil :: [nil | -(X ; Y), +(X), nil] .
  DYstrand :: nil :: [nil | -(X ; Y), +(Y), nil] .
  DYstrand :: nil :: [nil | -(X), +(sk(i,X)), nil] .
  DYstrand :: nil :: [nil | -(X), +(pk(A,X)), nil] .
  --- Strands
  strand [init] :: r :: [{A,B} | +(pk(B,n(A,r);A)),
                -(pk(A,n(A,r);N;B)), +(pk(B,N)), {A,B,n(A,r),N}] .
  strand [resp] :: r :: [{A,B} | -(pk(B,N;A)), +(pk(A,N;n(B,r);B)),
                -(pk(B,n(B,r))), {A,B,N,n(B,r)}] .
endp
```

Note that we allow each honest or Dolev-Yao strand to be *labeled* (e.g. `init` or `resp`), in contrast to the standard Maude-NPA syntax for strands. These strand labels play an important role in our protocol composition method as explained below.

■

Example 2 Similarly to the NSL protocol, there are several technical details missing in the previous informal description of DB. The specification of the DB protocol using the new syntax is as follows:

```
prot DB is
  --- Sort Information
  sorts Name Nonce .
  subsort Name Nonce < Msg .
  subsort Name < Public .
  --- Principals
  ops a b i : -> Name . --- Alice Bob Intruder
  --- Nonce operator
  op n : Name Fresh -> Nonce .
  --- Exclusive-or operator
  op _*_ : Msg  Msg  -> Msg [assoc comm gather (e E)] .
  op null : -> Msg .
  --- Variables
  vars X Y : Msg . var r : Fresh .
   vars A B : Name . var N : Nonce .
  --- XOR equational rules
  eq X * X * Y = Y .
  eq X * X = null .
  eq X * null = X .
  --- Dolev-Yao Strands
  DYstrand :: nil :: [nil | -(X), -(Y), +(X * Y), nil] .
  --- Strands
  strand [init]
  :: r :: [{A,B,NA} | +(n(B,r)), -(n(B,r)*NA), {A,B,NA,n(B,r)}] .
  strand [resp]
  :: nil :: [{A,B,NA} | -(NB), +(NB * NA), {A,B,NA,NB}] .
endp
```

The exclusive-or operator $\_\oplus\_$ is written as `_*_`. Since Maude-NPA does not yet include timestamps, we do not include all the actions relevant to calculating time intervals, sending timestamps, and checking them.

In this protocol specification, it is made clear that the nonce N_A used by the initiator is a parameter and is never generated by A during the run of DB. However, the initiator B does generate a new nonce.

■

Example 3 The previous informal description of the KD protocol also lacks several technical details, which we supply here. The whole protocol specification with the new syntax is as follows:

```
prot KD is
  --- Sort Information
  sorts Name Nonce Key Hash Enc .
  subsort Name Nonce Key Enc  < Msg .
  subsort Name < Public .
  subsort Hash < Key .
  --- Principals
  ops a b i : -> Name . --- Alice Bob Intruder
  --- Nonce operator
  op n : Name Fresh -> Nonce .
  --- Concatenation operator
  op _;_ : Msg  Msg -> Msg [gather (e E)] .
  --- Hash operator
  op h :  Msg Msg -> Hash .
  --- Key operator
  op skey : Name Fresh -> Key .
  --- Encryption Operators
  op e : Key Msg  -> Enc .
  op d : Key Msg ->  Enc .
  --- Variables
  var X : Msg . var r : Fresh . vars A B : Name .
  var N : Nonce . vars K SK : Key .
  --- Encryption/Decryption Cancellation
  eq d(K,e(K,X)) = X . eq e(K,d(K,X)) = X .
  --- Dolev-Yao Strands
  DYstrand :: nil :: [ nil | -(X), -(K), +(e(K,X)), nil ] .
  DYstrand :: nil :: [ nil | -(X), -(K), +(d(K,X)), nil ] .
  --- Strands
  strand [init] :: r :: [ {A,B,K} | +(e(K,skey(A,r)),
    -(e(K,skey(A,r) ; N)), +(e(K, N)), {A,B,K,skey(A,r),N}] .
  strand [resp] :: r :: [ {A,B,K} | -(e(K,SK)),
    +(e(K,SK ; n(B,r))), -(e(K,n(B,r))), {A,B,K,SK,n(B,r)} ] .
endp
```

Encryption of a message M with key K is denoted by $e(K, M)$, e.g., $\{N'_B\}_{h(N_A,N_B)}$ is denoted by $e(h(n(A,r), n(B,r')),\ n(B,r''))$. Cancellation properties of encryption and decryption are described using the equations $e(X, d(X, Z)) = Z$ and $d(X, e(X, Z)) = Z$. Session keys are written $skey(A, r)$, where A is the principal's name and r is a Fresh variable.

■

Sequential composition of two strands describes a situation in which one strand (the child strand) can only execute after the parent strand has its completed execution. Each composition of two strands is obtained by matching the output parameters of the parent strand with the input parameters of the child strand in a user-defined way. Note that it may be possible for a single parent strand to have more than one child strand.

The relevant fact in the DB protocol is that both nonces are required to be unknown to an attacker before they are sent, but the nonce originating from the responder must be previously agreed upon between the two principals. Therefore, this protocol is usually composed with another protocol ensuring secrecy and authentication of nonces. Furthermore, according to [Guttman 08], there are two extra issues related to the DB protocol that must be considered:

- the initiator of the previous protocol plays the role of the responder in DB and viceversa, and
- nonces generated by the parent protocol cannot be shared by more than one child so that an initiator of NSL will be connected to one and only one responder of DB.

In our working example, we use the NSL protocol to provide these capabilities.

6.1 Specifying Sequential Composition

Similarly to the syntax for protocols, we define protocol composition as follows[1]:

prot *Name* is *Name1* ; *Name2*
  $a_1\{\overrightarrow{O_1}\}$; $\{\overrightarrow{I_1}\}b_1$ [1-1] (or [1-*]) .
  $\vdots$
  $a_n\{\overrightarrow{O_n}\}$; $\{\overrightarrow{I_n}\}b_n$ [1-1] (or [1-*]) .
endp

where *Name* is a valid Maude-NPA module name, *Name1* and *Name2* are protocol names previously defined, $a_1, \ldots, a_n$ are labels of strands in protocol *Name1*, and $b_1, \ldots, b_n$ are labels of strands in protocol *Name2*. The expressions $[1-1]$ (or $[1-*]$) indicate whether a one-to-one (or a one-to-many) composition is desired for those two strands. Furthermore, for each composition $a_i\{\overrightarrow{O_i}\};\{\overrightarrow{I_i}\}b_i$, strand definition $::\ \overrightarrow{r_{a_i}}\ ::\ [\{\overrightarrow{I_{a_i}}\}, \overrightarrow{a_i}, \{\overrightarrow{O_{a_i}}\}]$ for role a_i in protocol *Name1*, and strand definition $::\ \overrightarrow{r_{b_i}}\ ::\ [\{\overrightarrow{I_{b_i}}\}, \overrightarrow{b_i}, \{\overrightarrow{O_{b_i}}\}]$ for role b_i in protocol *Name2*, we have that:

1. variables are properly renamed, i.e. $V_{ab} = \mathcal{V}ar(\overrightarrow{I_i}) \cup \mathcal{V}ar(\overrightarrow{O_i})$, $V_a = \mathcal{V}ar(\overrightarrow{I_{a_i}}) \cup \mathcal{V}ar(\overrightarrow{O_{a_i}})$, $V_b = \mathcal{V}ar(\overrightarrow{I_{b_i}}) \cup \mathcal{V}ar(\overrightarrow{O_{b_i}})$, and $V_{ab} \cap V_a \cap V_b = \emptyset$;
2. the variables of $\overrightarrow{I_i}$ must appear in $\overrightarrow{O_i}$ (no extra variables are allowed in a protocol composition);
3. the formal output parameters $\overrightarrow{O_{a_i}}$ must match the actual output parameters $\overrightarrow{O_i}$, i.e., $\exists\sigma_a$ s.t. $\overrightarrow{O_{a_i}} =_{E_\mathcal{P}} \sigma_a(\overrightarrow{O_i})$; and
4. the actual input parameters $\overrightarrow{I_i}$ must match the formal input parameters $\overrightarrow{I_{b_i}}$, i.e., $\exists\sigma_b$ s.t. $\overrightarrow{I_i} =_{E_\mathcal{P}} \sigma_b(\overrightarrow{I_{b_i}})$.

Note that for each composition, if there are substitutions σ_a and σ_b as described above, then there is a substitution σ_{ab} combining both, i.e., $\sigma_{ab}(X) = \sigma_a(\sigma_b(X))$ for any variable X, and then $\sigma_a(\overrightarrow{I_i}) =_{E_\mathcal{P}} \sigma_{ab}(\overrightarrow{I_{b_i}})$. This ensures

[1]Operator and sort renaming is indeed necessary, as in the Maude module importation language, but we do not consider those details in this thesis.

that any protocol composition is feasible and avoids the possibility of failing protocol compositions.

Let us consider again our two NSL and DB protocols and their composition. Note that we do not have to modify either the NSL or the DB specification above. The composition of both protocols is specified as follows:

```
prot NSL-DB is NSL ; DB
  NSL.init {A,B,NA,NB} ; {A,B,NA} DB.resp [1-1] .
  NSL.resp {A,B,NA,NB} ; {A,B,NA} DB.init [1-1] .
endp
```

Let us now consider the NSL and KD protocols and their composition. The composition of both protocols, which is an example of a one-to-many composition, is specified as follows:

```
prot NSL-KD is NSL ; KB
  NSL.init {A,B,NA,NB} ; {A,B,h(NA,NB)} KD.resp [1-*] .
  NSL.init {A,B,NA,NB} ; {A,B,h(NA,NB)} KD.init [1-*] .
  NSL.resp {A,B,NA,NB} ; {A,B,h(NA,NB)} KD.init [1-*] .
  NSL.resp {A,B,NA,NB} ; {A,B,h(NA,NB)} KD.resp [1-*] .
endp
```

In the remainder of this thesis we remove irrelevant parameters (i.e. input parameters for strands with no parents, and output parameters for strands with no children) in order to simplify the exposition.

Chapter 7

Maude-NPA's Composition Execution Model

In this chapter we define a concrete execution model for the one-to-one and one-to-many protocol compositions by extending the Maude-NPA execution model. However, we show that, by a simple protocol transformation, we are able to analyze and verify this dynamic composition in the current Maude-NPA tool. We prove soundness and completeness of the protocol transformation with respect to the extended operational semantics in Chapter 9.

7.1 Composition Execution Model

As explained in Chapter 4, the operational semantics of protocol execution and analysis is based on rewrite rules denoting state transitions which are applied *modulo* the algebraic properties $E_{\mathcal{P}}$ of the given protocol $\mathcal{P}$. Therefore, in the one-to-one and one-to-many cases we must add new state transition rules in order to deal with protocol composition. Maude-NPA performs backwards search modulo $E_{\mathcal{P}}$ by reversing the transition rules expressed in a forward way; see [Escobar 06, Escobar 09a]. Again, we define forward rewrite rules which will happen to be executed in a backwards way.[1]

[1]Note however, that we represent unification explicitly via a substitution σ instead of implicitly via variable equality as in Chapter 4. This is because output and input parameters are not required to match, e.g. in the composition NSL-KD, the output parameters

for each one-to-one composition $\{a\{\overrightarrow{O}\};\{\overrightarrow{I}\}b\}\,[1{-}1]$ with
strand definition $[\{\overrightarrow{I_a}\},\overrightarrow{a},\{\overrightarrow{O_a}\}]$ for protocol a,
strand definition $[\{\overrightarrow{I_b}\},\overrightarrow{b},\{\overrightarrow{O_b}\}]$ for protocol b,
and substitutions σ_a,σ_{ab} s.t. $\overrightarrow{O_a} =_{E_{\mathcal{P}}} \sigma_a(\overrightarrow{O})$ and $\sigma_a(\overrightarrow{I}) =_{E_{\mathcal{P}}} \sigma_{ab}(\overrightarrow{I_b})$,
we add the following rule :

$$\begin{aligned}&[\overrightarrow{a}\mid\{\overrightarrow{O_a}\}]\ \&\ [nil\mid\{\sigma_{ab}(\overrightarrow{I_b})\},\sigma_{ab}(\overrightarrow{b})]\,\&\,SS\,\&\,IK\\ \rightarrow\ &[\overrightarrow{a},\{\overrightarrow{O_a}\}\mid nil]\ \&\ [\{\sigma_{ab}(\overrightarrow{I_b})\}\mid\sigma_{ab}(\overrightarrow{b})]\,\&\,SS\,\&\,IK\end{aligned}\tag{7.1}$$

$$\begin{aligned}&[\overrightarrow{a}\mid\{\overrightarrow{O}\}]\ \&\ [\underline{nil}\mid\{\sigma_{ab}(\overrightarrow{I_b})\},\sigma_{ab}(\overrightarrow{b})]\,\&\,SS\,\&\,IK\\ \rightarrow\ &[\{\sigma_{ab}(\overrightarrow{I_b})\}\mid\sigma_{ab}(\overrightarrow{b})]\,\&\,SS\,\&\,IK\end{aligned}\tag{7.2}$$

Figure 7.1: Semantics for one-to-one composition

In the one-to-one composition, we add the state transition rules of Figure 7.1. Rule 7.1 composes a parent and a child strand already present in the current state. Rule 7.2 adds a parent strand to the current state and composes it with an existing child strand. Note that since a strand specification is a symbolic specification representing many concrete instances and the same applies to a composition of two protocol specifications, we need to relate actual and formal parameters of the protocol composition w.r.t. the two protocol specifications by using the substitutions σ_a and σ_{ab} in Figure 7.1, which are exactly those substitutions σ_a and σ_{ab} explained in Section 6.1. For example, given the following composition of the NSL-DB protocol

```
NSL.init {A,B,NA,NB} ; {A,B,NA} DB.resp [1-1] .
```

where NSL.init and DB.resp are, respectively, as follows:

```
[NSL.init] :: r :: [ +(pk(B,n(A,r);A)), -(pk(A,n(A,r);N;B)),
                      +(pk(B,N)),  {A,B,n(A,r),N} ] .
[DB.resp] :: nil :: [ {A',B',NA}, -(NB), +(NB * NA) ] .
```

of the parent strand are $\{A,B,N_A,N_B\}$ whereas the input parameters of the child strand are $\{A,B,h(N_A,N_B)\}$.

for each one-to-many composition $\{a\{\overrightarrow{O}\};\{\overrightarrow{I}\}b\}\,[1{-}*]$ with
strand definition $[\{\overrightarrow{I_a}\}, \overrightarrow{a}, \{\overrightarrow{O_a}\}]$ for protocol a,
strand definition $[\{\overrightarrow{I_b}\}, \overrightarrow{b}, \{\overrightarrow{O_b}\}]$ for protocol b,
and substitutions σ_a, σ_{ab} s.t. $\overrightarrow{O_a} =_{E_\mathcal{P}} \sigma_a(\overrightarrow{O})$ and $\sigma_a(\overrightarrow{I}) =_{E_\mathcal{P}} \sigma_{ab}(\overrightarrow{I_b})$,
we add one Rule 7.1, one Rule 7.2, and the following rule :

$$\begin{aligned}&[\overrightarrow{a} \mid \{\overrightarrow{O_a}\}] \;\&\; [nil \mid \{\sigma_{ab}(\overrightarrow{I_b})\}, \sigma_{ab}(\overrightarrow{b})] \,\&\, SS \,\&\, IK\\ \rightarrow\; &[\overrightarrow{a} \mid \{\overrightarrow{O_a}\}] \;\&\; [\{\sigma_{ab}(\overrightarrow{I_b})\} \mid \sigma_{ab}(\overrightarrow{b})] \,\&\, SS \,\&\, IK\end{aligned} \tag{7.3}$$

Figure 7.2: Semantics for one-to-many composition

we add the following transition rule for Rule (7.1) using substitution $\sigma_{ab} = \{$`A'` $\mapsto$ `A`, `B'` $\mapsto$ `B`, `NA` $\mapsto$ `n(A,r)`$\}$ where both the parent and the child strands are present and thus synchronized

```
:: r :: [ +(pk(B,n(A,r);A)), -(pk(A,n(A,r);N;B)), +(pk(B,N))
        | {A,B,n(A,r),N} ]
:: nil :: [ nil | {A,B,n(A,r)}, -(NB), +(NB * n(A,r)) ] & SS & IK
->
:: r :: [ +(pk(B,n(A,r);A)), -(pk(A,n(A,r);N;B)), +(pk(B,N)),
          {A,B,n(A,r),N} | nil ]
:: nil :: [{A,B,n(A,r)} | -(NB), +(NB * n(A,r)) ] & SS & IK
```

The reader can check how bars are moved in the parent and child strands to denote that the output parameters from the parent and the input parameters from the child have been synchronized and also how the repeated variables are used for proper data propagation (these variables being the effect of applying the σ_{ab} substitution.

One-to-many composition uses the rules in Figure 7.1 for the first child plus an additional rule for subsequent children, described in Figure 7.2. Rule 7.3 composes a parent strand and a child strand but the bar in the parent strand is not moved, in order to allow further backwards child compositions. For example, given the following composition of the NSL-KD protocol

```
  NSL.resp {A,B,NA,NB} ; {A,B,h(NA,NB)} KD.init [1-*] .
```

where NSL.resp and KD.init are, respectively, as follows:

```
[NSL.resp] :: r :: [ -(pk(B,NA;A)), +(pk(A,NA;n(B,r);B)),
                     -(pk(B,n(B,r))), {A,B,NA,n(B,r)}] .
[KD.init] :: r' :: [ {A',B',K}, +(e(K,skey(A',r')),
                     -(e(K,skey(A',r');N)), +(e(K, N))] .
```

we add the following transition rule for Rule (7.3) using substitution $\sigma_{ab} = \{$`A'` $\mapsto$ `A`, `B'` $\mapsto$ `B`, `K` $\mapsto$ `h(NA,n(B,r))`$\}$:

```
:: r :: [ -(pk(B,NA;A)), +(pk(A,NA;n(B,r);B)), -(pk(B,n(B,r)))
          | {A,B,NA,n(B,r)}] .
:: r' :: [ nil | {A,B,h(NA,n(B,r))}, +(e(h(NA,n(B,r)),skey(A,r')),
           -(e(h(NA,n(B,r)),skey(A,r') ; N)),
           +(e(h(NA,n(B,r)), N)) ] .
& SS & IK
->
:: r :: [ -(pk(B,NA;A)), +(pk(A,NA;n(B,r);B)), -(pk(B,n(B,r))),
          {A,B,NA,n(B,r)} | nil ] .
:: r' :: [ {A,B,h(NA,n(B,r))} | +(e(h(NA,n(B,r)),skey(A,r')),
           -(e(h(NA,n(B,r)),skey(A,r') ; N)),
           +(e(h(NA,n(B,r)), N)) ] .
& SS & IK
```

Thus, for a protocol composition $\mathcal{P}_1;\mathcal{P}_2$, the rewrite rules governing protocol execution are $R^{\circ}_{\mathcal{P}_1;\mathcal{P}_2} = \{(4.1),(4.2),(4.3)\} \cup (4.4) \cup (7.1) \cup (7.2) \cup (7.3)$.

7.2 Protocol Composition by Protocol Transformation

Instead of implementing a new version of the Maude-NPA generating new transition rules for each protocol composition, we have defined a *protocol*

$$\Phi(\mathcal{P}_1;\mathcal{P}_2) = \begin{cases} \text{add strand } [\overrightarrow{a}, -(role_b(r)), +(role_a(r) \;.\; \sigma_{ab}(\dot{I}_b))] \text{ and} \\ \text{strand } [+(role_b(r)), -(role_a(r) \;.\; \dot{I}_b), \overrightarrow{b}\,] \\ \quad \text{whenever } \; \{a\{\overrightarrow{O}\};\{\overrightarrow{I}\}b\}\,[1{-}1] \text{ in } \mathcal{P}_1;\mathcal{P}_2, \\ \qquad\qquad\qquad \text{strand definition } [role_a][\{\overrightarrow{I_a}\}, \overrightarrow{a}, \{\overrightarrow{O_a}\}] \text{ for protocol } a, \\ \qquad\qquad\qquad \text{strand definition } [role_b][\{\overrightarrow{I_b}\}, \overrightarrow{b}, \{\overrightarrow{O_b}\}] \text{ for protocol } b, \\ \qquad\qquad\qquad \exists \sigma_a, \sigma_{ab} \text{ s.t. } \overrightarrow{O_a} =_{E_{\mathcal{P}}} \sigma_a(\overrightarrow{O}) \text{ and } \sigma_a(\overrightarrow{I}) =_{E_{\mathcal{P}}} \sigma_{ab}(\overrightarrow{I_b}) \\ \qquad\qquad\qquad \text{and } r \text{ is a fresh variable} \\ \text{add strand } [\overrightarrow{a}, +(role_a(r) \;.\; \sigma_{ab}(\dot{I}_b))] \text{ and strand } [-(role_a(r) \;.\; \dot{I}_b), \overrightarrow{b}\,] \\ \quad \text{whenever } \; \{a\{\overrightarrow{O}\};\{\overrightarrow{I}\}b\}\,[1{-}*] \text{ in } \mathcal{P}_1;\mathcal{P}_2, \\ \qquad\qquad\qquad \text{strand definition } [role_a][\{\overrightarrow{I_a}\}, \overrightarrow{a}, \{\overrightarrow{O_a}\}] \text{ for protocol } a, \\ \qquad\qquad\qquad \text{strand definition } [role_b][\{\overrightarrow{I_b}\}, \overrightarrow{b}, \{\overrightarrow{O_b}\}] \text{ for protocol } b, \\ \qquad\qquad\qquad \exists \sigma_a, \sigma_{ab} \text{ s.t. } \overrightarrow{O_a} =_{E_{\mathcal{P}}} \sigma_a(\overrightarrow{O}) \text{ and } \sigma_a(\overrightarrow{I}) =_{E_{\mathcal{P}}} \sigma_{ab}(\overrightarrow{I_b}) \\ \qquad\qquad\qquad \text{and } r \text{ is a fresh variable} \end{cases}$$

Figure 7.3: Protocol Transformation

transformation that achieves the same effect using the current Maude-NPA tool.

The protocol transformation is given in Figure 7.3. Its output is a single, composed protocol specification where:

1. Sorts, symbols, and equational properties of both protocols are put together into a single specification[2]. Strands of both protocols are transformed and added to this single specification as described in Figure 7.3.

2. For each composition we transform the input parameters $\{\overrightarrow{I_b}\}$ into an input message exchange of the form $-(\overrightarrow{I_b})$, and the output parameters $\{\overrightarrow{O_a}\}$ into an output message exchange of the form $+(\sigma_{ab}(\overrightarrow{I_b}))$. The sort Param of these messages is disjoint from the sort Msg used by the protocol in the honest and intruder strands. This ensures that they are harmless, since no intruder strand will be able to use them. In order to avoid type conflicts, we use a *dot* for concatenation within protocol composition exchange messages, e.g. input parameters $\overrightarrow{I} = \{A, B, NA\}$ are transformed into the sequence $\dot{I} = A \;.\; B \;.\; NA$.

3. Each composition is uniquely identified by using a composition iden-

[2]Note that we allow shared items but require the user to solve any possible conflict. Operator and sort renaming is an option, as in the Maude module importation language, but we do not consider those details in this paper.

tifier (a variable of sort Fresh). Strands exchange such composition identifier by using input/output messages of the form $role_j(r)$, which make the role explicit. The sort Role of these messages is disjoint from the sorts Param and Msg.

(a) In a one-to-one protocol composition, the child strand uniquely generates a fresh variable that is added to the area of fresh identifiers at the beginning of its strand specification. This fresh variable must be passed from the child to the parent before the parent generates its output parameters and sends it back again to the child.

(b) In a one-to-many protocol composition, the parent strand uniquely generates a fresh variable that is passed to the child. Since an (a priori) unbounded number of children will be composed with it, no reply of the fresh variable is expected by the parent from the children.

For example, for the following one-to-one protocol composition in NSL-DB

```
  NSL.init {A,B,NA,NB} ; {A,B,NA} DB.resp [1-1] .
```

where NSL.init and DB.resp are, respectively, as follows:

```
[NSL.init] :: r :: [ +(pk(B,n(A,r);A)), -(pk(A,n(A,r);N;B)),
                     +(pk(B,N)), {A,B,n(A,r),N} ] .
[DB.resp] :: nil :: [ {A,B,NA}, -(NB), +(NB * NA) ] .
```

we have the following two transformed strands:

```
[NSL.init] :: r :: [ +(pk(B,n(A,r);A)), -(pk(A,n(A,r);N;B)),
                     +(pk(B,N)), -(db-resp(r#)),
                     +(nsl-init(r#) . A . B . n(A,r)) ] .
[DB.resp] :: r# :: [ +(db-resp(r#)), -(nsl-init(r#) . A . B . NA),
                     -(NB), +(NB * NA) ] .
```

For the current $1-1$ composition of NSL with the distance-bounding protocol DB, we can formulate the composition as a single protocol with these two transformed strands containing these extra synchronization messages for protocol composition, as follows.

```
prot NSL-DB is
 --- Synchronization for composition
 sort Params Msg$ Role .
 subsort Params Role Msg$ < Msg .
 --- Roles
 ops init-nsl resp-nsl : Fresh -> Role .
 ops init-db  resp-db :  Fresh -> Role .
 --- Composition
 sort ParamSeq .
 subsort Msg$ < ParamSeq .
 op _._ : ParamSeq ParamSeq -> ParamSeq [gather (e E)] .
 op {_} : ParamSeq -> Params .
 --- Sort Information
 sorts Name Nonce  Enc .
 subsort Name Nonce Enc < Msg$ .
 subsort Name < Public .
 --- Encoding operators for public/private encryption
 op pk : Name Msg$ -> Enc .
 op sk : Name Msg$ -> Enc .
 --- Principals
 ops a b i : -> Name . --- Alice Bob Intruder
 --- Nonce operator
 op n : Name Fresh -> Nonce .
 --- Concatenation operator
 op _;_ : Msg$  Msg$  -> Msg$ [gather (e E)] .
 --- Exclusive-or operator
 op _*_ : Msg$ Msg$ -> Msg$ [assoc comm] .
 op null : -> Msg$ .
 --- Variables
 vars X Y Z : Msg$ . vars A B : Name .
 vars r r' r'' r# : Fresh .
 vars A B C : Name .
 vars NA NB N : Nonce .
 vars X Y Z H : Msg$ .
 vars P Q : Name .
 --- Encryption/Decryption Cancellation
 eq pk(A,sk(A,Z)) = Z .  eq sk(A,pk(A,Z)) = Z .
 --- XOR equational rules
 eq X * X * Y = Y . eq X * X = null . eq X * null = X .
 --- Dolev-Yao Strands
 DYstrand :: nil :: [ nil | -(X), -(Y), +(X * Y), nil ] .
 DYstrand :: nil :: [ nil | -(X), -(Y), +(X ; Y), nil ] .
 DYstrand :: nil :: [ nil | -(X ; Y), +(X), nil ] .
 DYstrand :: nil :: [ nil | -(X ; Y), +(Y), nil ] .
 DYstrand :: nil :: [ nil | -(X), +(sk(i,X)), nil ] .
 DYstrand :: nil :: [ nil | -(X), +(pk(A,X)), nil ] .
 --- Strands
 strand [nsl-init] :: r :: [ nil | +(pk(B,n(A,r) ; A)),
    -(pk(A, n(A,r) ; NB ; B )), +(pk(B, NB)),
    -(resp-db(r#)), +({init-nsl(r#) . A . B . n(A,r)}), nil] .
 strand [nsl-resp] :: r :: [ nil | -(pk(B,NA ; A)),
    +(pk(A, NA ; n(B,r) ; B)), -(pk(B,n(B,r))),
    -(init-db(r#)), +({resp-nsl(r#) . A . B  . NA }), nil] .
 strand [db-init] :: r', r# :: [ nil | +(init-db(r#)),
    -({resp-nsl(r#) . A . B . NA  }), +(n(B,r')),
    -(NA * n(B,r')), nil] .
 strand [db-res‘] :: r# :: [ nil | +(resp-db(r#)),
    -({init-nsl(r#) . A . B . NA }), -(N), +( NA * N), nil ]
endp
```

For the following one-to-many protocol composition in the NSL-KD

```
  NSL.resp {A,B,NA,NB} ; {A,B,h(NA,NB)} KD.init [1-*] .
```

where NSL.init and KD.init are, respectively, as follows:

```
[NSL.resp] :: r :: [ -(pk(B,N;A)), +(pk(A,N;n(B,r);B)),
                     -(pk(B,n(B,r))), {A,B,N,n(B,r)}] .
[KD.init] :: r :: [ {A,B,K},  +(e(K,skey(A,r)),
                      -(e(K,skey(A,r) ; N')), +(e(K,N')) ] .
```

we have the following two transformed strands:

```
[NSL.resp] :: r,r# :: [-(pk(B,N;A)), +(pk(A,N;n(B,r);B)),
                      -(pk(B,n(B,r))),
                      +(nsl-init(r#) . A . B . h(N,n(B,r))) ] .
[KD.init] :: r :: [ -(nsl-init(r#) . A . B . K), +(e(K,skey(A,r)),
                     -(e(K,skey(A,r) ; N')), +(e(K,N')) ] .
```

The whole composition of NSL with the key distribution protocol KD, can be formulated as a single protocol too, as follows.

```
prot NSL-KD is
  --- Synchronization for composition
  sorts Params Role Msg$ . subsort Params Role Msg$ < Msg .
  --- Roles
  ops init-nsl resp-nsl : Fresh -> Role .
  ops init-key resp-key : Fresh -> Role .
  --- Composition
  sort ParamSeq .
  subsorts Msg$ Role < ParamSeq .
  op _._ : ParamSeq ParamSeq -> ParamSeq [gather (e E)] .
  op {_} : ParamSeq -> Params .
  --- Sort Information
  sorts Name Nonce Key Hash Enc .
  subsort Name Nonce Key Hash Enc < Msg$ .
  subsort Name < Public .
  subsort Hash < Key .
  --- Encoding operators for public/private encryption
  op pk : Name Msg$ -> Enc .
  op sk : Name Msg$ -> Enc .
  --- Principals
  ops a b i : -> Name . --- Alice Bob Intruder
  --- Nonce operator
  op n : Name Fresh -> Nonce .
  --- Concatenation operator
  op _;_ : Msg$  Msg$  -> Msg$ [gather (e E)] .
  --- Exclusive-or operator
  op _*_ : Msg$ Msg$ -> Msg$ [assoc comm] .
  op null : -> Msg$ .
  --- Hash operator
  op h :  Msg$ Msg$ -> Hash .
  --- Key operator
  op skey : Name Fresh -> Key .
  --- Encryption Operators
  op e : Key Msg$  -> Enc . op d : Key Msg$ ->  Enc .
  --- Variables
  vars X Y Z : Msg$ . vars A B : Name . var K : Key .
  vars r r' r'' r# : Fresh .
  vars NA NB N : Nonce .
  vars P Q : Name .
  vars KA KB : Key .
  --- Encryption/Decryption Cancellation
  eq pk(A,sk(A,Z)) = Z .
  eq sk(A,pk(A,Z)) = Z .
  eq d(K,e(K,Z)) = Z .
  eq e(K,d(K,Z)) = Z .
  --- XOR equational  rules
  eq X * X * Y = Y . eq X * X = null . eq X * null = X .
  --- Dolev-Yao Strands
  DYstrand :: nil :: [ nil | -(X), -(Y), +(X * Y), nil ] .
  DYstrand :: nil :: [ nil | -(X), -(Y), +(X ; Y), nil ] .
  DYstrand :: nil :: [ nil | -(X ; Y), +(X), nil ] .
  DYstrand :: nil :: [ nil | -(X ; Y), +(Y), nil ] .
  DYstrand :: nil :: [ nil | -(X), +(sk(i,X)), nil ] .
  DYstrand :: nil :: [ nil | -(X), +(pk(A,X)), nil ] .
  DYstrand :: nil :: [ nil | -(X), -(K), +(e(K,X)), nil ] .
  DYstrand :: nil :: [ nil | -(X), -(K), +(d(K,X)), nil ] .
  strand [nsl-init] :: r , r# :: [ nil | +(pk(B,n(A,r) ; A)),
     -(pk(A, n(A,r) ; NB ; B )), +(pk(B, NB)),
     +({init-nsl(r#) . A . B . h(n(A,r) , NB) }), nil ] .
  strand [nsl-resp] :: r , r# :: [ nil | -(pk(B,NA ; A)),
     +(pk(A, NA ; n(B,r) ; B)), -(pk(B,n(B,r))),
     +({resp-nsl(r#) . A . B . h(NA , n(B,r))}), nil ] .
  strand [kd-init] :: r' :: [ nil | -({init-nsl(r#) . A . B . K }),
     +(e(K, skey(A, r'))) , -(e(K, skey(A,r') ; N)), +(e(K, N)), nil] .
  strand [kd-resp] :: r' :: [ nil | -({resp-nsl(r#) . A . B . K }),
     -(e(K, KA)), +(e(K, KA ; n(B,r'))), -(e(K, n(B,r'))), nil ]  &
  strand [kd-init] :: r' :: [ nil | -({resp-nsl(r#) . A . B . K }),
     +(e(K, skey(B,r'))), -(e(K, skey(B,r') ; N)), +(e(K, N)), nil ] .
  strand [kd-resp] :: r' :: [ nil | -({init-nsl(r#) . A . B . K }),
     -(e(K, KB )), +(e(K, KB ; n(A,r'))), -(e(K, n(A,r'))),  nil ]  .
endp
```

We leave the proof of soundness and completeness of the protocol transformation with respect to the extended operational semantics until Chapter 9 and consider protocol composition in practice in the next section.

Chapter 8

Formal Analysis

In this chapter we present some experimental results for protocol composition based on the transformed protocol composition. First, we show the attack for the NSL-DB explained in Section 2.1. Then we fix the NSL-DB protocol using a hash function, also explained in Section 2.1 and show that the protocol is verified as secure by our tool, i.e., the search space is finite and an attack is not found. Finally, we show that the NSL-KD is also verified as secure by the Maude-NPA. All the experiments, including the source Maude-NPA files and the generated outputs, can be found at `http://www.dsic.upv.es/~ssantiago/composition.html`. Every time that we show a protocol secure, we also show that a regular execution can be performed, proving that the search space is not empty a priori, however these proofs have not been included in this thesis, though are available online.

8.1 The NSL-DB protocol

We start with the attack for the NSL-DB protocol composition. The property (represented by an attack state) is one in which the honest principal `b` thinks that he has heard from a principal P (who may or may not be honest), but who has actually heard from an honest principal Q. This covers, for example, the case in which P is dishonest, and tries to pass on a honest principal's authenticated response as his own.

```
eq ATTACK-STATE(0)
  = :: r', r# ::
     [ nil | +(init-db(r#)),
             -({resp-nsl(r#) . P . b . n(Q,r)}),
             +(n(b,r'')), -(n(b,r'') * n(Q,r)), nil]
    || Q != P
    || nil
    || nil
    || nil .
```

However, the composition of the NSL and DB protocols does not guarantee this property. The analysis of this property for the NSL-DB transformed protocol using the Maude-NPA finds an initial state from which the attack state described above is reachable. The principal and intruder strands of such an initial state are as follows:

```
:: nil :: [nil | -(pk(i,n(Q,r') ; Q)), +(n(Q,r') ; Q) ] &
:: nil :: [nil | -(pk(i,n(Q,r') ; n(b,r'') ; b)),
                 +(n(Q,r') ; n(b,r'') ; b) ] &
:: nil :: [nil | -(n(b,r)), -(n(Q,r')), +(n(b,r) * n(Q,r')) ] &
:: nil :: [nil | -(n(b,r'')), +(pk(b,n(b,r''))) ] &
:: nil :: [nil | -(n(Q,r')), -(i), +(n(Q,r') ; i) ] &
:: nil :: [nil | -(n(b,r'') ; b), +(n(b,r'')) ] &
:: nil :: [nil | -(n(Q,r') ; i), +(pk(b,n(Q,r') ; i)) ] &
:: nil :: [nil | -(n(Q,r') ; Q), +(n(Q,r')) ] &
:: nil :: [nil | -(n(Q,r') ; n(b,r'') ; b), +(n(b,r'') ; b) ] &
:: r'' :: [nil | +(resp-nsl), -(pk(b,n(Q,r') ; i)),
                 +(pk(i,n(Q,r') ; n(b,r'') ; b)),  -(pk(b,n(b,r''))),
                 -(init-db(r#)), +({resp-nsl(r#) . i . b . n(Q,r')}) ] &
:: r' :: [nil | +(init-nsl), +(pk(i,n(Q,r') ; Q)) ] &
:: r , r# :: [nil | +(init-db(r#)), -({resp-nsl(r#) . i . b . n(Q,r')}),
                    +(n(b,r)), -(n(b,r) * n(Q,r')) ])
```

Figure 8.1 shows the graphical representation of the strands above given by the Maude-NPA GUI [Santiago 09]. In this attack state the actual message exchange between those principal and intruder strands is as follows:

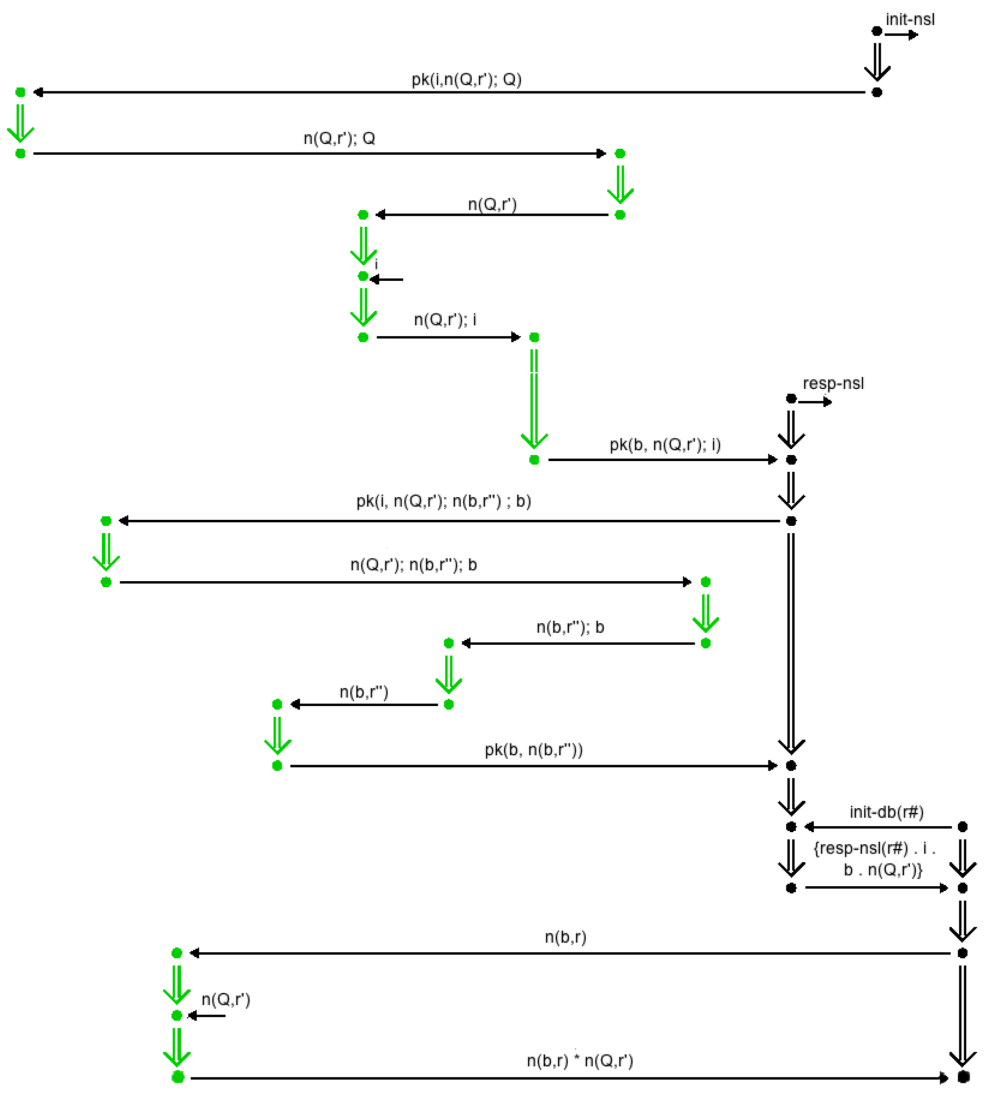

Figure 8.1: Graphical representation of the NSL-DB attack

```
+(init-nsl),                     +(n(b,r'') ; b),
+(pk(i,n(Q,r') ; Q)),            -(n(b,r'') ; b),
-(pk(i,n(Q,r') ; Q)),            +(n(b,r'')),
+(n(Q,r') ; Q),                  -(n(b,r'')),
-(n(Q,r') ; Q),                  +(pk(b,n(b,r''))),
+(n(Q,r')),                      +(init-db(r#)),
-(n(Q,r')),                      -(pk(b,n(b,r''))),
-(i),                            -(init-db(r#)),
+(n(Q,r') ; i),                  +({resp-nsl(r#) . i . b . n(Q,r')}),
-(n(Q,r') ; i),                  -({resp-nsl(r#) . i . b . n(Q,r')}),
+(pk(b,n(Q,r') ; i)),            +(n(b,r)),
+(resp-nsl),                     -(n(b,r)),
-(pk(b,n(Q,r') ; i)),            -(n(Q,r')),
+(pk(i,n(Q,r') ; n(b,r'') ; b)),+(n(b,r) * n(Q,r')),
-(pk(i,n(Q,r') ; n(b,r'') ; b)),-(n(b,r) * n(Q,r'))
+(n(Q,r') ; n(b,r'') ; b),
-(n(Q,r') ; n(b,r'') ; b),
```

However, as explained in Chapter 2, this attack is avoided using a hash function. The fixed version of the DB protocol is as follows:

```
prot DB-hash is
  --- Sort Information
  sorts Name Nonce .
  subsort Name Nonce < Msg .
  subsort Name < Public .
  --- Principals
  ops a b i : -> Name . --- Alice Bob Intruder
  --- Nonce operator
  op n : Name Fresh -> Nonce .
  --- Exclusive-or operator
  op _*_ : Msg  Msg  -> Msg [assoc comm gather (e E)] .
  op null : -> Msg .
  --- Hash operator
  op h :  Msg Msg -> Hash .
  --- Variables
  vars X Y : Msg . var r : Fresh .
   vars A B : Name . var N : Nonce .
  --- XOR equational rules
  eq X * X * Y = Y .
  eq X * X = null .
  eq X * null = X .
  --- Dolev-Yao Strands
  DYstrand :: nil :: [nil | -(X), -(Y), +(X * Y), nil] .
  --- Strands
  strand [init]
  :: r :: [{A,B,NA} | +(n(B,r)), -(n(B,r)*h(A,NA)), {A,B,NA,n(B,r)}] .
  strand [resp]
  :: nil :: [{A,B,NA} | -(NB), +(NB * h(A,NA)), {A,B,NA,NB}] .
endp
```

The transformed version of the protocol composition NSL-DB-hash is as follows:

```
prot NSL-DB-HASH is
  --- Synchronization for composition
  sort Params Msg$ Role .
  subsort Params < Msg .
  subsort Msg$ Role < Msg .
  subsort Role < Msg .
  --- Roles
  ops init-nsl resp-nsl : Fresh -> Role .
  ops init-db resp-db :  Fresh -> Role .
  --- Composition
  sort ParamSeq .
  subsort Msg$ Role < ParamSeq .
  op _._ : ParamSeq ParamSeq -> ParamSeq [gather (e E) frozen] .
  op {_} : ParamSeq -> Params .
  --- Sort Information
  sorts Name Nonce  Enc .
  subsort Name Nonce Enc < Msg$ .
  subsort Name < Public .
  --- Encoding operators for public/private encryption
  op pk : Name Msg$ -> Enc .
  op sk : Name Msg$ -> Enc .
  --- Principals
  ops a b i : -> Name . --- Alice Bob Intruder
  --- Nonce operator
  op n : Name Fresh -> Nonce .
  --- Concatenation operator
  op _;_ : Msg$  Msg$  -> Msg$ [gather (e E)] .
  --- Exclusive-or operator
  op _*_ : Msg$ Msg$ -> Msg$ [assoc comm] .
  op null : -> Msg$ .
  --- Hash operator
    op h(_,_) : Name Nonce -> Msg$.
  --- Variables
  vars X Y Z : Msg$ . vars A B : Name .
  --- Encryption/Decryption Cancellation
  eq pk(A,sk(A,Z)) = Z .
  eq sk(A,pk(A,Z)) = Z .
  --- XOR equational  rules
  eq X * X * Y = Y . eq X * X = null . eq X * null = X .
  DYstrand :: nil :: [ nil | -(X), -(Y), +(X * Y), nil ] .
  DYstrand :: nil :: [ nil | -(X), -(Y), +(X ; Y), nil ] .
  DYstrand :: nil :: [ nil | -(X ; Y), +(X), nil ] .
  DYstrand :: nil :: [ nil | -(X ; Y), +(Y), nil ] .
  DYstrand :: nil :: [ nil | -(X), +(sk(i,X)), nil ] .
  DYstrand :: nil :: [ nil | -(X), +(pk(A,X)), nil ] .
  strand [init-nsl] :: r ::  [ nil | +(pk(B,n(A,r) ; A)),
     -(pk(A, n(A,r) ; NB ; B )), +(pk(B, NB)),
     -(resp-db(r#)), +({init-nsl(r#) . A . B . n(A,r)}), nil ] .
  strand [resp-nsl] :: r :: [ nil | -(pk(B,NA ; A)),
     +(pk(A, NA ; n(B,r) ; B)), -(pk(B,n(B,r))),
      -(init-db(r#)), +({resp-nsl(r#) . A . B  . NA }), nil ] .
  strand [init-db] :: r', r# :: [ nil | +(init-db(r#)),
      -({resp-nsl(r#) . A . B . NA  }), +(n(B,r')), -(h(A,NA) * n(B,r')), nil] .
  strand [resp-db] :: r# :: [ nil | +(resp-db(r#)),
      -({init-nsl(r#) . A . B . NA }), -(N), +( h(A,NA) * N), nil ]
endp
```

And the previous property for the NSL-DB is specified in the new version of the protocol with the following attack pattern:

```
 eq ATTACK-STATE(0)
   = :: r', r# ::
     [ nil, +(init-db(r#)),
           -({resp-nsl(r#) . Q  . b . n(P,r) }),
           +(n(b,r')), -(n(b,r') * h(Q,n(P,r))) | nil ]
     || Q != P
     || nil
     || nil
     || nil .
```

In this case, the analysis terminates without finding any attack. Thus, the protocol is secure for such an attack pattern, since the search space was finite (i.e., after the application of all the optimizations provided by the tool, see [Escobar 08]).

8.2 The NSL-KD protocol

Finally, the property that we may wish for the NSL-KD protocol is to guarantee that a dishonest principal is not able to learn the secret key of an honest principal. This property is represented by the following attack state:

```
  eq ATTACK-STATE(0)
   = :: r' ::
     [ nil , +(resp-key),
             -({resp-nsl(r#) . a . b . K }),
             -(e(K, KA )), +(e(K, KA ; n(b,r'))),
             -(e(K, n(b,r'))) | nil ]
     || KA inI
     || nil
     || nil .
```

In this case, that property is satisfied by the NSL-KD, since the analysis terminates after 7 backwards reachability steps without finding any initial state for the attack state described above. Below we include the output of the analysis of the NSL-KD given by the Maude-NPA.

```
==========================================
reduce in MAUDE-NPA : summary(1) .
rewrites: 3391205 in 27499ms cpu (27505ms real)
          (123317 rewrites/second)
result Summary: States>> 5 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(2) .
rewrites: 87328734 in 504410ms cpu (504500ms real)
          (173130 rewrites/second)
result Summary: States>> 8 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(3) .
rewrites: 218063153 in 1201493ms cpu (1201708ms real)
          (181493 rewrites/second)
result Summary: States>> 8 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(4) .
rewrites: 385741394 in 2053801ms cpu (2054168ms real)
          (187818 rewrites/second)
result Summary: States>> 5 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(5) .
rewrites: 252124790 in 1400027ms cpu (1400278ms real)
          (180085 rewrites/second)
result Summary: States>> 3 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(6) .
rewrites: 186089431 in 1032143ms cpu (1032327ms real)
          (180294 rewrites/second)
result Summary: States>> 1 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(7) .
rewrites: 80549705 in 400475ms cpu (400546ms real)
          (201135 rewrites/second)
result Summary: States>> 0 Solutions>> 0
==========================================
```

Chapter 9

Soundness and Completeness of the Protocol Transformation

In this chapter, we prove soundness and completeness of the protocol transformation with respect to the extended operational semantics.

9.1 Relating States from Protocol Composition and Protocol Transformation

First, we must relate protocol states using the protocol composition rewrite rules of Chapter 7 and protocol states in the transformed protocol composition.

Definition 1 *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two protocols and $\mathcal{P}_1;\mathcal{P}_2$ their composition. Let $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$ be the rewrite theory associated in Section 7.1 to the protocol composition $\mathcal{P}_1;\mathcal{P}_2$ and $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$ be the rewrite theory associated in Section 7.2 to the transformed protocol. Given a state St associated to the rewrite theory $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$ and a state St' associated to the rewrite theory $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$, we define the (non-symmetric) relation $St \,{}^{\circ}{\equiv}^{\Phi}_{\mathcal{P}_1;\mathcal{P}_2} St'$ (or just $St \,{}^{\circ}{\equiv}^{\Phi} St'$) as the maximal relation satisfying the conditions in Figure 9.1.*

The following auxiliary result becomes crucial and ensures that for each state corresponding to the rewrite theory associated to the transformed pro-

$$
St \mathrel{{}^{\circ}{\equiv}^{\Phi}_{\mathcal{P}_1;\mathcal{P}_2}} St' \begin{cases}
\text{whenever } & \{a\{\overrightarrow{O}\};\{\overrightarrow{I}\}b\}\,[1{-}1] \text{ in } \mathcal{P}_1;\mathcal{P}_2,\\
& \text{strand } [\{\overrightarrow{I_b}\},\overrightarrow{b_1} \mid \overrightarrow{b_2}] \text{ in } St,\\
& \text{and strand } [+(role_b(r)), -(role_a(r) \;.\; \dot{I}_b), \overrightarrow{b_1} \mid \overrightarrow{b_2}] \text{ in } St'.\\
\text{whenever } & \{a\{\overrightarrow{O}\};\{\overrightarrow{I}\}b\}\,[1{-}1] \text{ in } \mathcal{P}_1;\mathcal{P}_2,\\
& \text{strand } [\{\overrightarrow{I_b}\} \mid \overrightarrow{b}\,] \text{ in } St,\\
& \text{and strand } [+(role_b(r)) \mid -(role_a(r) \;.\; \dot{I}_b), \overrightarrow{b}\,] \text{ in } St'.\\
\text{whenever } & \{a\{\overrightarrow{O}\};\{\overrightarrow{I}\}b\}\,[1{-}1] \text{ in } \mathcal{P}_1;\mathcal{P}_2,\\
& \text{strand } [nil \mid \{\overrightarrow{I_b}\}, \overrightarrow{b}\,] \text{ in } St,\\
& \text{strand } [\overrightarrow{a_1} \mid \overrightarrow{a_2}, \{\overrightarrow{O_a}\}] \text{ in } St,\\
& \text{strand } [nil \mid +(role_b(r)), -(role_a(r) \;.\; \dot{I}_b), \overrightarrow{b}\,] \text{ in } St',\\
& \text{and strand } [\overrightarrow{a_1} \mid \overrightarrow{a_2}, -(role_b(r)), +(role_a(r) \;.\; \dot{I}_b)] \text{ in } St'.\\
\text{whenever } & \{a\{\overrightarrow{O}\};\{\overrightarrow{I}\}b\}\,[1{-}*] \text{ in } \mathcal{P}_1;\mathcal{P}_2,\\
& \text{strand } [\{\overrightarrow{I_b}\},\overrightarrow{b_1} \mid \overrightarrow{b_2}] \text{ in } St,\\
& \text{and strand } [-(role_a(r) \;.\; \dot{I}_b), \overrightarrow{b_1} \mid \overrightarrow{b_2}] \text{ in } St'.\\
\text{whenever } & \{a\{\overrightarrow{O}\};\{\overrightarrow{I}\}b\}\,[1{-}*] \text{ in } \mathcal{P}_1;\mathcal{P}_2,\\
& \text{strand } [\{\overrightarrow{I_b}\} \mid \overrightarrow{b}\,] \text{ in } St,\\
& \text{strand } [nil \mid -(role_a(r) \;.\; \dot{I}_b), \overrightarrow{b}\,] \text{ in } St',\\
& \text{and there is no strand } [\overrightarrow{a} \mid +(role_a(r) \;.\; \dot{I}_b)] \text{ in } St'.\\
\text{whenever } & \{a\{\overrightarrow{O}\};\{\overrightarrow{I}\}b\}\,[1{-}*] \text{ in } \mathcal{P}_1;\mathcal{P}_2,\\
& \text{strand } [nil \mid \{\overrightarrow{I_b}\}, \overrightarrow{b}\,] \text{ in } St,\\
& \text{strand } [\overrightarrow{a_1} \mid \overrightarrow{a_2}, \{\overrightarrow{O_a}\}] \text{ in } St,\\
& \text{strand } [nil \mid -(role_a(r) \;.\; \dot{I}_b), \overrightarrow{b}\,] \text{ in } St',\\
& \text{and strand } [\overrightarrow{a_1} \mid \overrightarrow{a_2}, +(role_a(r) \;.\; \dot{I}_b)] \text{ in } St'.
\end{cases}
$$

plus every element $m{\in}\mathcal{I}$ and $m{\notin}\mathcal{I}$ in the intruder knowledge of St appears in the intruder knowledge of St', and every element $m{\in}\mathcal{I}$ and $m{\notin}\mathcal{I}$ in the intruder knowledge of St' such that m is not of sort Param or Role appears in the intruder knowledge of St.

Figure 9.1: Relation ${}^{\circ}{\equiv}^{\Phi}_{\mathcal{P}_1;\mathcal{P}_2}$ from the states associated to the rewrite theory $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$ onto the states associated to the rewrite theory $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$

tocol, there is always a state corresponding to the rewrite theory associated to the protocol composition.

Lemma 1 *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two protocols and $\mathcal{P}_1;\mathcal{P}_2$ their composition. Let $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$ be the rewrite theory associated in Section 7.1 to the protocol composition $\mathcal{P}_1;\mathcal{P}_2$ and $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$ be the rewrite theory associated in Section 7.2 to the transformed protocol. If St' is a protocol state associated to the rewrite theory $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$, then there is a protocol state St associated to the rewrite theory $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$ such that $St\,{}^{\circ}{\equiv}^{\Phi}_{\mathcal{P}_1;\mathcal{P}_2}\,St'$.*

Proof. Immediate by definition of the transformation Φ in Figure 7.3 and the binary relation $St\,{}^{\circ}{\equiv}^{\Phi}_{\mathcal{P}_1;\mathcal{P}_2}\,St'$ of Definition 1. □

Another relevant property is ensuring that one-to-one protocol compositions are indeed one-to-one in the transformed protocol composition. We define this result in terms of one strand composition $a\{\overrightarrow{O}\};\{\overrightarrow{I}\}b$ [1−1], which is easily extensible to the whole protocol composition $\mathcal{P}_1;\mathcal{P}_2$.

Lemma 2 (Unique One-to-one Composition) *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two protocols and $\mathcal{P}_1;\mathcal{P}_2$ be a one-to-one composition. Let $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$ be the rewrite theory associated in Section 7.2 to the transformed protocol and $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$ be the rewrite theory associated in Section 7.1 to the protocol composition $\mathcal{P}_1;\mathcal{P}_2$. If St' is a protocol state associated to the rewrite theory $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$, then, for each parent strand, there is only one child composed with it.*

Proof. Recall that a fresh variable `r#` is uniquely generated by each child strand, sent to the prospective parent strand, and sent back to the child strand by the input/output exchange messages. Therefore, each parent uses only one fresh variable `r#`. Now, the Maude-NPA restriction that two strands cannot generate a same fresh variable ensures that there are no two children generating the same `r#`. □

9.2 Soundness and Completeness for One Narrowing Step

We introduce our main results for soundness and completeness. First, we consider soundness of one backwards narrowing step using the rewrite theory associated to the transformed protocol (i.e., $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$) w.r.t. backwards narrowing using the rewrite theory associated to protocol composition (i.e., $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$). We write R^{-1} to denote the reverse form of all the rules in R. The composition of two substitutions σ_1 and σ_2 is defined as $\sigma_1 \circ \sigma_2(X) = \sigma_2(\sigma_1(X))$ for any variable X. A substitution ρ is called *E-compatible* with another substitution σ if there is a substitution τ s.t. $\sigma =_E \rho \circ \tau$.

Theorem 1 (One-step Soundness) *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two protocols and $\mathcal{P}_1;\mathcal{P}_2$ their composition. Let $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$ be the rewrite theory associated in Section 7.1 to the protocol composition $\mathcal{P}_1;\mathcal{P}_2$ and $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$ be the rewrite theory associated in Section 7.2 to the transformed protocol. Let St'_1 and St'_2 be two protocol states associated to the rewrite theory $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$. If $St'_1 \leadsto_{\sigma,\mathcal{R}^{-1}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)},E_{\mathcal{P}_1;\mathcal{P}_2}} St'_2$, then either*

1. *there is a protocol state St_1 associated to the rewrite theory $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$ and a substitution ρ such that $St_1 \,{}^{\circ}{\equiv}^{\Phi}\, St'_1$, $\rho(St_1) \,{}^{\circ}{\equiv}^{\Phi}\, St'_2$, and σ and ρ are $E_{\mathcal{P}_1;\mathcal{P}_2}$-compatible; or*

2. *there are two protocol states St_1 and St_2 associated to the rewrite theory $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$ and two substitutions ρ and ρ' such that $\rho(St_1) \,{}^{\circ}{\equiv}^{\Phi}\, St'_1$, $St_2 \,{}^{\circ}{\equiv}^{\Phi}\, St'_2$, $St_1 \leadsto_{\rho',\mathcal{R}^{\circ,-1}_{\mathcal{P}_1;\mathcal{P}_2},E_{\mathcal{P}_1;\mathcal{P}_2}} St_2$, and $\rho' = \rho \circ \sigma$.*

Proof. We prove the result by case analysis of the rewrite rules applicable to term St'_1.

Let us consider first the second case of the theorem statement where one rewrite step in $\mathcal{R}^{-1}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$ corresponds to one rewrite step in $\mathcal{R}^{\circ,-1}_{\mathcal{P}_1;\mathcal{P}_2}$. This case corresponds to two situations: (i) the standard uses of the rule-base semantics (i.e. Rules (4.1),(4.2), (4.3), and (4.4)) for dealing with regular incoming and outcoming messages in the strand; and (ii) the following transitions dealing with the messages associated to the input and output parameters:

1. In the case in which Rule (4.4) in $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$ is applied in order to produce St'_2 by adding a strand $[\overrightarrow{a} \mid +(role_a(r).\dot{I}_b)]$ to St'_1 by unification with a term $(role_a(r)\ .\ \dot{I}_b){\in}\mathcal{I}$ in the intruder knowledge of St'_1, this corresponds to the application of Rule (7.2) in $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$, in which the state St_2 is created by adding the strand $[\overrightarrow{a} \mid \{\overrightarrow{O_a}\}]$ to a state St_1 by unification with the input parameters of a child strand. In this case, the substitution ρ above is the identity, since both rule applications produce the same unifier.

2. In the case in which the state St'_2 is created by applying the rule Rule (4.3) in $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$ to unify the term $+(role_a(r')\ .\ \dot{I}_b)$ of a strand $[\overrightarrow{a}, +(role_a(r')\ .\ \dot{I}_b) \mid nil]$ in St'_1 with a term $(role_a(r)\ .\ \dot{I}_b){\in}\mathcal{I}$ in the intruder knowledge of St'_1, this corresponds to the creation of St_2 from St_1 via the application of Rule (7.1) in $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$, in which the output parameters $\{\overrightarrow{O_a}\}$ of the strand associated to protocol a are synchronized with the input parameters $\{\overrightarrow{I_b}\}$ of the strand associated to protocol b. Similarly, in this case the substitution ρ above is the identity, since both rule applications produce the same unifier.

3. In the case in which the state St'_2 is created from St'_1 via application of Rule (4.3) in $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$ to unify the term $+(role_b(r))$ of a strand $[+(role_b(r)) \mid -(role_a(r)\ .\ \dot{I}_b), \overrightarrow{b}]$ in St'_1 with a term $role_b(r){\in}\mathcal{I}$ in the intruder knowledge of St'_1, this corresponds to the application of Rule (7.1) in $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$ to create St_2 from St_1, in which the output parameters $\{\overrightarrow{O_a}\}$ of the strand associated to protocol a are synchronized with the input parameters $\{\overrightarrow{I_b}\}$ of the strand associated to protocol b. In this case, the substitution ρ above may be different from the identity, since the concrete unifier computed by Rule (7.1) was computed before, when the term $role_a(r)\ .\ \dddot{I}_b \in\mathcal{I}$ was unified with a concrete parent strand.

Now, let us consider the first case of the theorem statement where one rewrite step in $\mathcal{R}^{-1}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$ does not correspond to one rewrite step in $\mathcal{R}^{\circ,-1}_{\mathcal{P}_1;\mathcal{P}_2}$ and an instantiation is just computed for St_1. This case corresponds to the rule applications in $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$ manipulating the input and output messages associated to the protocol composition such that the same state is associated to $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$ by the equivalence $^{\circ}{\equiv}^{\Phi}$. Such rule applications in $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$ are as follows:

1. In a one-to-one composition, the child transition from strand $[+(role_b(r)), -(role_a(r)\ .\ \dot{I}_b)\ \mid\ \overrightarrow{b}\,]$ in St'_1 to strand $[+(role_b(r))\ \mid\ -(role_a(r)\ .\ \dot{I}_b), \overrightarrow{b}\,]$ in St'_2. Here, states St'_1 and St'_2 have the same state St_1 given by $^{\circ}{\equiv}^{\Phi}$.

2. In a one-to-one composition, the parent transition from strand $[\overrightarrow{a}, -(role_b(r)), +(role_a(r)\ .\ \dot{I}_b) \mid nil]$ in St'_1 to strand $[\overrightarrow{a}, -(role_b(r)) \mid +(role_a(r)\ .\ \dot{I}_b)]$ in St'_2. Here, state St'_1 has a state St_1 given by $^{\circ}{\equiv}^{\Phi}$, St'_2 has a state St_2 given by $^{\circ}{\equiv}^{\Phi}$ and states St_1 and St_2 differ only in that St_2 is an instance of St_1.

3. In a one-to-one composition, the addition of parent strand $[\overrightarrow{a}, -(role_b(r)) \mid +(role_a(r)\ .\ \dot{I}_b)]$ to St'_1, yields St'_2. Here, state St'_1 has a state St_1 given by $^{\circ}{\equiv}^{\Phi}$, St'_2 has a state St_2 given by $^{\circ}{\equiv}^{\Phi}$ and states St_1 and St_2 differ only in that St_2 is an instance of St_1.

4. In a one-to-one composition, the parent transition from strand $[\overrightarrow{a}, -(role_b(r))\ \mid\ +(role_a(r)\ .\ \dot{I}_b)]$ in St'_1 to strand $[\overrightarrow{a}\ \mid\ -(role_b(r)), +(role_a(r)\ .\ \dot{I}_b)]$ in St'_2. Here, states St'_1 and St'_2 have the same state St_1 given by $^{\circ}{\equiv}^{\Phi}$.

5. In a one-to-many composition, the child transition from strand $[-(role_a(r)\ .\ \dot{I}_b)\ \mid\ \overrightarrow{b}\,]$ in St'_1 to strand $[nil\ \mid\ -(role_a(r)\ .\ \dot{I}_b), \overrightarrow{b}\,]$ in St'_2. Here, states St'_1 and St'_2 have the same state St_1 given by $^{\circ}{\equiv}^{\Phi}$.

This concludes the proof. □

Now, we consider completeness of one backwards narrowing step using the rewrite theory associated to protocol composition (i.e., $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$) w.r.t. backwards narrowing using the rewrite theory associated to the transformed protocol (i.e., $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$) w.r.t. We write $St_1 \rightarrow^{1,2,4}_{R} St_2$ to denote that either $St_1 \rightarrow_R St_2$, or there is a term St' such that $St_1 \rightarrow_R St' \rightarrow_R St_2$, or there are terms St', St'', St''' such that $St_1 \rightarrow_R St' \rightarrow_R St'' \rightarrow_R St''' \rightarrow_R St_2$.

Theorem 2 (One-step Completeness) *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two protocols and $\mathcal{P}_1;\mathcal{P}_2$ their composition. Let $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$ be the rewrite theory associated in Section 7.1 to the protocol composition $\mathcal{P}_1;\mathcal{P}_2$ and $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$ be the rewrite theory associated in Section 7.2 to the transformed protocol. Let St_1 and St_2 be two protocol states associated to the rewrite theory $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$. If $St_1 \leadsto_{\sigma,\mathcal{R}^{\circ,-1}_{\mathcal{P}_1;\mathcal{P}_2},E_{\mathcal{P}_1;\mathcal{P}_2}} St_2$, then there are two protocol states St'_1 and St'_2 associated to the rewrite theory $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$ such that $St_1 \,{}^{\circ}\!\!\equiv^{\Phi} St'_1$, $St_2 \,{}^{\circ}\!\!\equiv^{\Phi} St'_2$, and $St'_1 \leadsto^{1,2,4}_{\sigma,\mathcal{R}^{-1}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)},E_{\mathcal{P}_1;\mathcal{P}_2}} St'_2$.*

Proof. We prove the result by case analysis of the rewrite rules applicable to term St_1. Since the proof of this theorem is very similar to the proof of Theorem 1, we present the cases only in broad outline.

When the Rules (4.1),(4.2), (4.3), and (4.4) are applied to regular incoming and outcoming messages in the strands, we have just one narrowing step from the associated state St'_1.

When we have a one-to-many composition and Rules (7.1), (7.2), or (7.3) are applied, we have two narrowing steps from the associated state St'_1 as follows:

1. Rule (7.1) corresponds to an application of Rule (4.1) (accepting the input parameters of the child strand) followed by an application of Rule (4.3) (synchronizing the input of the child strand with the output parameters of the parent strand);

2. Rule (7.2) correspond to an application of Rule (4.1) followed by an application of Rule (4.4) (introducing a new strand); and

3. Rule (7.3) correspond to an application of Rule (4.3) (synchronizing the output parameters of the parent strand with the already accepted input parameters of the child strand).

When we have a one-to-one composition and Rules (7.1) or (7.2) are applied, we have four narrowing steps from the associated state St'_1 as follows:

1. Rule (7.1) corresponds to an application of Rule (4.1) (accepting the input parameters of the child strand) followed by an application of Rule (4.3) (synchronizing the input of the child strand with the output parameters of the parent strand), and two further applications of Rules (4.1) and (4.3) for the $role_b(r)$ message; and

2. Rule (7.2) correspond to an application of Rule (4.1) followed by an application of Rule (4.4) (introducing a new strand), and two further applications of Rules (4.1) and (4.3) for the $role_b(r)$ message.

This concludes the proof. □

9.3 Soundness and Completeness for Reachability Analysis

Now, we extend the previous results of soundness and completeness of one rewrite step to backwards reachability analysis.

Theorem 3 (Reachability Soundness) *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two protocols and $\mathcal{P}_1;\mathcal{P}_2$ their composition. Let $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$ be the rewrite theory associated in Section 7.2 to the transformed protocol and $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$ be the rewrite theory associated in Section 7.1 to the protocol composition $\mathcal{P}_1;\mathcal{P}_2$. If St'_1 and St'_2 are two protocol states associated to the rewrite theory $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$ such that St'_2 is an initial state, and $St'_1 \leadsto^{*}_{\sigma,\mathcal{R}^{-1}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)},E_{\mathcal{P}_1;\mathcal{P}_2}} St'_2$, then there are two protocol states St_1 and St_2 associated to the rewrite theory $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$ and two substitutions ρ and ρ' such that St_2 is an initial state, $\rho(St_1) \,{}^{\circ}\!\equiv^{\Phi} St'_1$, $St_2 \,{}^{\circ}\!\equiv^{\Phi} St'_2$, $St_1 \leadsto^{*}_{\rho',\mathcal{R}^{\circ,-1}_{\mathcal{P}_1;\mathcal{P}_2},E_{\mathcal{P}_1;\mathcal{P}_2}} St_2$, and $\rho' = \rho \circ \sigma$.*

Proof. By successive application of Theorem 1. Let us consider

$$St'_1 \leadsto^{n}_{\sigma,\mathcal{R}^{-1}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)},E_{\mathcal{P}_1;\mathcal{P}_2}} St'_2.$$

If $n = 0$, then the conclusion is immediate. If $n > 0$, then there is a state St' s.t.

$$St'_1 \leadsto_{\sigma_1,\mathcal{R}^{-1}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)},E_{\mathcal{P}_1;\mathcal{P}_2}} St' \leadsto^{n-1}_{\sigma',\mathcal{R}^{-1}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)},E_{\mathcal{P}_1;\mathcal{P}_2}} St'_2.$$

Then, by Theorem 2, either

1. there is a protocol state St_1 associated to the rewrite theory $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$ and a substitution ρ such that $St_1 \,{}^{\circ}{\equiv}^{\Phi}\, St'_1$, $\rho(St_1) \,{}^{\circ}{\equiv}^{\Phi}\, St'$ and σ_1 and ρ are $E_{\mathcal{P}_1;\mathcal{P}_2}$-compatible; or

2. there are two protocol states St_1 and St associated to the rewrite theory $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$ and two substitutions ρ_1 and ρ'_1 such that $\rho_1(St_1) \,{}^{\circ}{\equiv}^{\Phi}\, St'_1$, $St \,{}^{\circ}{\equiv}^{\Phi}\, St'$, $St_1 \leadsto_{\rho'_1,\mathcal{R}^{\circ,-1}_{\mathcal{P}_1;\mathcal{P}_2},E_{\mathcal{P}_1;\mathcal{P}_2}} St$ and $\rho'_1 = \rho_1 \circ \sigma_1$.

In the first case, we can apply the induction hypothesis saying that there are two protocol states $\widehat{St}$ and St_2 associated to the rewrite theory $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$ and two substitutions $\widehat{\rho}$ and $\widehat{\rho'}$ such that St_2 is an initial state, $\widehat{\rho}(\widehat{St}) \,{}^{\circ}{\equiv}^{\Phi}\, St'$, $St_2 \,{}^{\circ}{\equiv}^{\Phi}\, St'_2$, $\widehat{St} \leadsto^{*}_{\widehat{\rho'},\mathcal{R}^{\circ,-1}_{\mathcal{P}_1;\mathcal{P}_2},E_{\mathcal{P}_1;\mathcal{P}_2}} St_2$, and $\widehat{\rho'} = \widehat{\rho} \circ \sigma'$. Since $\rho(St_1) \,{}^{\circ}{\equiv}^{\Phi}\, St'$ and $\widehat{\rho}(\widehat{St}) \,{}^{\circ}{\equiv}^{\Phi}\, St'$, we can conclude by definition of the relation ${}^{\circ}{\equiv}^{\Phi}$ that there is a substitution σ'' s.t $St_1 \leadsto^{*}_{\sigma'',\mathcal{R}^{\circ,-1}_{\mathcal{P}_1;\mathcal{P}_2},E_{\mathcal{P}_1;\mathcal{P}_2}} St_2$ and $\sigma'' = \rho \circ \widehat{\rho'}$. This concludes this part of the proof.

In the second case, we can apply the induction hypothesis saying that there are there are two protocol states $\widehat{St}$ and St_2 associated to the rewrite theory $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$ and two substitutions $\widehat{\rho}$ and $\widehat{\rho'}$ such that St_2 is an initial state, $\widehat{\rho}(\widehat{St}) \,{}^{\circ}{\equiv}^{\Phi}\, St'$, $St_2 \,{}^{\circ}{\equiv}^{\Phi}\, St'_2$, $\widehat{St} \leadsto^{*}_{\widehat{\rho'},\mathcal{R}^{\circ,-1}_{\mathcal{P}_1;\mathcal{P}_2},E_{\mathcal{P}_1;\mathcal{P}_2}} St_2$, and $\widehat{\rho'} = \widehat{\rho} \circ \sigma'$. Here the conclusion follows because we can build the sequence

$$St_1 \leadsto_{\rho'_1,\mathcal{R}^{\circ,-1}_{\mathcal{P}_1;\mathcal{P}_2},E_{\mathcal{P}_1;\mathcal{P}_2}} St \leadsto^{*}_{\widehat{\rho'},\mathcal{R}^{\circ,-1}_{\mathcal{P}_1;\mathcal{P}_2},E_{\mathcal{P}_1;\mathcal{P}_2}} St_2.$$

such that $\rho(St_1) \,{}^{\circ}{\equiv}^{\Phi}\, St'_1$ $St_2 \,{}^{\circ}{\equiv}^{\Phi}\, St'_2$, and $\rho'_1 \circ \widehat{\rho'} = \rho_1 \circ \sigma_1 \circ \widehat{\rho} \circ \sigma'$. This concludes this part of the proof. □

Theorem 4 (Reachability Completeness) *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two protocols and $\mathcal{P}_1;\mathcal{P}_2$ their composition. Let $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$ be the rewrite theory associated in Section 7.2 to the transformed protocol, and $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$ be the rewrite theory associated in Section 7.1 to the protocol composition $\mathcal{P}_1;\mathcal{P}_2$. If St_1 and St_2 are two protocol states associated to the rewrite theory $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$ such that St_2 is an initial state, and $St_1 \rightsquigarrow^{*}_{\sigma,\mathcal{R}^{\circ,-1}_{\mathcal{P}_1;\mathcal{P}_2},E_{\mathcal{P}_1;\mathcal{P}_2}} St_2$, then there are two protocol states St'_1 and St'_2 associated to the rewrite theory $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$ such that St'_2 is an initial state, $St_1 \,{}^{\circ}\!\!\equiv^{\Phi} St'_1$, $St_2 \,{}^{\circ}\!\!\equiv^{\Phi} St'_2$, and $St'_1 \rightsquigarrow^{*}_{\sigma,\mathcal{R}^{-1}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)},E_{\mathcal{P}_1;\mathcal{P}_2}} St'_2$.*

Proof. By successive application of Theorem 2. Let us consider

$$St_1 \rightsquigarrow^{n}_{\sigma,\mathcal{R}^{\circ,-1}_{\mathcal{P}_1;\mathcal{P}_2},E_{\mathcal{P}_1;\mathcal{P}_2}} St_2.$$

If $n = 0$, then the conclusion follows. If $n > 0$, then there is a state St such that

$$St_1 \rightsquigarrow_{\sigma_1,\mathcal{R}^{\circ,-1}_{\mathcal{P}_1;\mathcal{P}_2},E_{\mathcal{P}_1;\mathcal{P}_2}} St \rightsquigarrow^{n}_{\sigma',\mathcal{R}^{\circ,-1}_{\mathcal{P}_1;\mathcal{P}_2},E_{\mathcal{P}_1;\mathcal{P}_2}} St_2.$$

By Theorem 2, there are two protocol states St'_1 and St' associated to the rewrite theory $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$ such that $St_1 \,{}^{\circ}\!\!\equiv^{\Phi} St'_1$, $St \,{}^{\circ}\!\!\equiv^{\Phi} St'$, and $St'_1 \rightsquigarrow^{1,2,4}_{\sigma_1,\mathcal{R}^{-1}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)},E_{\mathcal{P}_1;\mathcal{P}_2}} St'$. Then, by induction hypothesis, there are two protocol states $\widehat{St'}$ and St'_2 associated to the rewrite theory $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$ such that St'_2 is an initial state, $St \,{}^{\circ}\!\!\equiv^{\Phi} \widehat{St'}$, $St_2 \,{}^{\circ}\!\!\equiv^{\Phi} St'_2$, and $\widehat{St'} \rightsquigarrow^{*}_{\sigma',\mathcal{R}^{-1}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)},E_{\mathcal{P}_1;\mathcal{P}_2}} St'_2$. Since $St \,{}^{\circ}\!\!\equiv^{\Phi} \widehat{St'}$ and $St \,{}^{\circ}\!\!\equiv^{\Phi} St'$, by definition of the relation ${}^{\circ}\!\!\equiv^{\Phi}$ we can assume that $St' = \widehat{St'}$. The conclusion follows because we can build the sequence

$$St'_1 \rightsquigarrow^{1,2,4}_{\sigma_1,\mathcal{R}^{-1}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)},E_{\mathcal{P}_1;\mathcal{P}_2}} St' \rightsquigarrow^{*}_{\sigma',\mathcal{R}^{-1}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)},E_{\mathcal{P}_1;\mathcal{P}_2}} St'_2.$$

□

Finally, we put everything together into one result.

Theorem 5 (Soundness and Completeness) *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two protocols and $\mathcal{P}_1;\mathcal{P}_2$ their composition. Let $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$ be the rewrite theory associated in Section 7.1 to the protocol composition $\mathcal{P}_1;\mathcal{P}_2$ and $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$ be the rewrite theory associated in Section 7.2 to the transformed protocol. Given St and St' associated to $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$ and $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$, respectively, then $St \,{}^{\circ}{\equiv}^{\Phi}\, St'$ implies that there is an initial state In reachable from St by backwards narrowing in $\mathcal{R}_{\Phi(\mathcal{P}_1;\mathcal{P}_2)}$ iff there is an initial state In' such that $In \,{}^{\circ}{\equiv}^{\Phi}\, In'$ and In' is reachable from St' by backwards narrowing in $\mathcal{R}^{\circ}_{\mathcal{P}_1;\mathcal{P}_2}$.*

Proof. By Theorems 3 and 4. □

Chapter 10

Related work and conclusions

Our work addresses a somewhat different problem than most existing work on cryptographic protocol composition, which generally does not address model-checking. Indeed, to the best of our knowledge, most protocol analysis model-checking tools simply use concatenation of protocol specifications to express sequential composition. However, we believe that the problem we are addressing is an important one that tackles a widely acknowledged source of protocol complexity. For example, in the Internet Key Exchange Protocol [Harkins 98] there are sixteen different one-to-many parent-child compositions of Phase One and Phase Two protocols. The ability to synthesize compositions automatically would greatly simplify the specification and analysis of protocols like these.

Future work related with the subject presented in this thesis includes experiments with more complex protocol compositions and optimizations to speed up the analysis. We also plan to take advantage of the Maude-NPA GUI to give syntax support and to mechanize the protocol transformation explained in Section 7.2, which is not supported by the current version of the Maude-NPA yet.

Now that we have a mechanism for synthesizing compositions, we are ready to revisit existing research on composing protocols and their properties and determine how we could best make use of it in our framework. There have been two approaches to this problem. One (called *nondestructive* composition in [Datta 03]) is to concentrate on properties of protocols and conditions on them that guarantee that properties satisfied separately are not violated by the composition. This is, for example, the approach taken by Gong and Syverson [Gong 98], Guttman and Thayer [Guttman 00], Cortier and Delaune [Cortier 09] and, in the computational model, Canetti's Universal Composability [Canetti 02]. The conditions in this case are usually ones that can be verified syntactically, so Maude-NPA, or any other model checker, would not be of much assistance here.

Of more interest to us is the research that addresses the compositionality of the protocol properties themselves (called *additive* composition in [Datta 03]). This addresses the development of logical systems and tools such as PCL, PDL, and CPSA cited earlier in this paper, in which inference rules are provided for deriving complex properties of a protocol from simpler ones. Since these are pure logical systems, they necessarily start from very basic statements concerning, for example, what a principal can derive when it receives a message. But there is no reason why the properties of the component protocols could not be derived using model checking, and then composed using the logic. This would give us the benefits of both model checking (for finding errors and debugging), and logical derivations (for building complex systems out of simple components), allowing to switch between one and the other as needed. Indeed, Maude-NPA is well positioned in that respect. For example, the notion of state in strand spaces that it uses is very similar to that used by PDL [Cervesato 05], and we have already developed a simple property language that allows us to translate the "shapes" produced by CPSA into Maude-NPA attack states. The next step in our research will be to investigate the connection more closely from the point of view of compositionality.

Chapter 11

Publications Associated to this Thesis

The publications associated to this Master Thesis are listed in the following:

- S. Santiago, C. Talcott, S. Escobar, C. Meadows, and J. Meseguer
  A Graphical User Interface for Maude-NPA.
  XIV Jornadas sobre Programacin y Lenguajes (PROLE09),
  Electronic Notes in Theoretical Computer Science, 258(1): 3-20 (2009)

- S. Escobar, C. Meadows, J. Meseguer and S. Santiago
  Sequential Protocol Composition in Maude-NPA
  Technical Report DSIC-II/06/10, Universidad Politécnica de Valencia, June 2010.

- S. Escobar, C. Meadows, J. Meseguer and S. Santiago
  Sequential Protocol Composition in Maude-NPA
  15th European Symposium on Research in Computer Security (ESORICS 2010).
  Lecture Notes in Computer Science, 6345:303-318 (2010)

Bibliography

[Anlauff 06] M. Anlauff, D. Pavlovic, R. Waldinger & S. Westfold. *Proving Authentication Properties in the Protocol Derivation Assistant.* In Proc. of Joint Workshop on Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis, 2006.

[Canetti 02] R. Canetti, Y. Lindell, R. Ostrovsky & A.Sahai. *Universally composable two-party and multi-party secure computation.* In STOC, pages 494–503, 2002.

[Capkun 06] S. Capkun & J. P. Hubaux. *Secure Positioning in Wireless Networks.* IEEE Journal on Selected Areas in Communication, vol. 24, no. 2, February 2006.

[Cervesato 05] I. Cervesato, C. Meadows & D. Pavlovic. *An Encapsulated Authentication Logic for Reasoning About Key Establishment Protocols.* In IEEE Computer Security Foundations Workshop, 2005, 2005.

[Cortier 09] V. Cortier & S. Delaune. *Safely composing security protocols.* Formal Methods in System Design, vol. 34, no. 1, pages 1–36, 2009.

[Datta 03] A. Datta, A. Derek, J. C. Mitchell & D. Pavlovic. *Secure Protocol Composition.* In Proc. Mathematical Foundations of Programming Semantics, volume 83 of *Electronic Notes in Theoretical Computer Science*, 2003.

[Desmedt 88] Y. Desmedt. *Major security problems with the "unforgeable" (Feige-)Fiat-Shamir proofs of identity and how to overcome them.* In Securicom 88, 6th worldwide congress on computer and communications security and protection, pages 147–159, Paris France, March 1988.

[Doghim 07] S. Doghim, J. Guttman & F. J. Thayer. *Searching for Shapes in Cryptographic Protocols.* In TACAS 2007. Springer LNCS 4424, March 2007.

[Dolev 83] D. Dolev & A. Yao. *On the security of public key protocols.* IEEE Transaction on Information Theory, vol. 29, no. 2, pages 198–208, 1983.

[Durgin 01] N. Durgin, J. Mitchell & D. Pavlovic. *A Compositional Logic for Program Correctness.* In Fifteenth Computer Security Foundations Workshop — CSFW-14, Cape Breton, NS, Canada, 11–13 June 2001. IEEE Computer Society Press.

[Escobar 06] S. Escobar, C. Meadows & J. Meseguer. *A rewriting-based inference system for the NRL Protocol Analyzer and its meta-logical properties.* Theor. Comput. Sci., vol. 367, no. 1-2, pages 162–202, 2006.

[Escobar 08] Santiago Escobar, Catherine Meadows & José Meseguer. *State Space Reduction in the Maude-NRL Protocol Analyzer.* In Sushil Jajodia & Javier López, editeurs, Computer Security - ESORICS 2008, 13th European Symposium on Research in Computer Security, Málaga, Spain, October 6-8, 2008. Proceedings, volume 5283 of *Lecture Notes in Computer Science*, pages 548–562. Springer, 2008.

[Escobar 09a] S. Escobar, C. Meadows & J. Meseguer. *Maude-NPA: Cryptographic Protocol Analysis Modulo Equational Properties.* In A. Aldini, G. Barthe & R. Gorrieri, editeurs, FOSAD

2008/2009 Tutorial Lectures, volume 5705 of *LNCS*, pages 1–50. Springer, 2009.

[Escobar 09b] Santiago Escobar, Catherine Meadows & José Meseguer. *Maude-NPA, version 1.0.* University of Illinois at Urbana-Champaign, March 2009. Available at `http://maude.cs.uiuc.edu/tools/Maude-NPA`.

[Fabrega 99] F. J. Thayer Fabrega, J. Herzog & J. Guttman. *Strand Spaces: What Makes a Security Protocol Correct?* Journal of Computer Security, vol. 7, pages 191–230, 1999.

[Gong 98] L. Gong & P. Syverson. *Fail-stop protocols: An approach to designing secure protocols.* In R. K. Iyer, M. Morganti, W. K. Fuchs & V. Gligor, editeurs, Proc. of the 5th IFIP International Working Conference on Dependable Computing for Critical Applications (Urbana-Champaign, IL, Sept. 1995), pages 79–99. IEEE Computer Society Press, Los Alamitos, CA, 1998.

[Guttman 00] J. D. Guttman & F. J. Thayer. *Protocol Independence through Disjoint Encryption.* In CSFW, pages 24–34, 2000.

[Guttman 01] J. Guttman. *Security Protocol Design via Authentication Tests.* In Proc. Computer Security Foundations Workshop. IEEE Computer Society Press, 2001.

[Guttman 08] J. D. Guttman, J. C. Herzog, V. Swarup & F. J. Thayer. *Strand spaces: From Key Exchange to Secure Location.* In Carolyn Talcott, editor, Workshop on Event-Based Semantics, 2008. Position papers available at `http://blackforest.stanford.edu/eventsemantics/`.

[Harkins 98] D. Harkins & D. Carrel. *The Internet Key Exchange (IKE)*, November 1998. IETF RFC 2409.

[Lowe 96] G. Lowe. *Breaking and fixing the Needham-Schroeder public key protocol using FDR.* In Tools and Algorithms for the Construction and Analysis of Systems (TACAS '96), volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer-Verlag, 1996.

[Meseguer 92] J. Meseguer. *Conditional Rewriting Logic as a Unified Model of Concurrency.* Theoretical Computer Science, vol. 96, no. 1, pages 73–155, 1992.

[Meseguer 98] J. Meseguer. *Membership algebra as a logical framework for equational specification.* In F. Parisi-Presicce, editor, Proc. WADT'97, pages 18–61. Springer LNCS 1376, 1998.

[Santiago 09] Sonia Santiago, Carolyn L. Talcott, Santiago Escobar, Catherine Meadows & José Meseguer. *A Graphical User Interface for Maude-NPA. Electronic Notes in Theorethical Computer Science,* vol. 258, no. 1, pages 3–20, 2009.

[TeReSe 03] TeReSe, editor. Term rewriting systems. Cambridge University Press, Cambridge, 2003.

[Thati 07] P. Thati & J. Meseguer. *Symbolic reachability analysis using narrowing and its application verification of cryptographic protocols.* J. Higher-Order and Symbolic Computation, vol. 20, no. 1–2, pages 123–160, 2007.