Universitat Politècnica de València
Departamento de Sistemas Informáticos y Computación
Doctorado en Informática

Ph.D Thesis

Advanced Features in Protocol Verification: Theory, Properties, and Efficiency in Maude-NPA

Candidate:
Sonia Santiago Pinazo

Supervisor:
Dr. Santiago Escobar Román

Valencia, January 2015

Universitat Politècnica de València
Departamento de Sistemas Informáticos y Computación
Doctorado en Informática

Ph.D Thesis

Advanced Features in Protocol Verification: Theory, Properties, and Efficiency in Maude-NPA

A dissertation submitted by Sonia Santiago Pinazo in fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science with International Mention at the Universitat Politècnica de València.

Valencia, January 2015

Work partially supported by the EU (FEDER) and the Spanish MECD, under grants TIN 2007-68118-C02, TIN 2010-21062-C02-02, and FPU AP2009-4297; by the Generalitat Valenciana, under grant PROMETEO/2011/052; by the Universitat Politècnica de València under grant "Beca de Excelencia (2010)".

Title: Advanced Features in Protocol Verification: Theory, Properties, and Efficiency in Maude-NPA
Author: Sonia Santiago Pinazo
Address: Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València
Camí de Vera, s/n
46022 Valencia
España
E-mail: ssantiago@dsic.upv.es

Advanced Features in Protocol Verification: Theory, Properties, and Efficiency in Maude-NPA

Author:

Sonia Santiago Pinazo

Supervisor:

Dr. Santiago Escobar Román U. Politècnica de València

External Evaluators:

Prof. Narciso Martí Oliet U. Complutense de Madrid
Prof. Maribel Fernández King's College London
Prof. Luca Viganò King's College London

Jury:

Prof. Silvia Abrahão U. Politècnica de València
Prof. Gilles Barthe IMDEA Software
Prof. Luca Viganò King's College London

January 23rd, 2015

A mis padres, Juan y Rosa, por su apoyo incondicional. Su ejemplo de esfuerzo y trabajo duro ha sido siempre mi inspiración.

To my parents, Juan and Rosa, for their unconditional support. Their example of effort and hard work has always been my inspiration.

Abstract

The area of formal analysis of cryptographic protocols has been an active one since the mid 80's. The idea is to verify communication protocols that use encryption to guarantee secrecy and that use authentication of data to ensure security. Formal methods are used in protocol analysis to provide formal proofs of security, and to uncover bugs and security flaws that in some cases had remained unknown long after the original protocol publication, such as the case of the well known Needham-Schroeder Public Key (NSPK) protocol. In this thesis we tackle problems regarding the three main pillars of protocol verification: modelling capabilities, verifiable properties, and efficiency.

This thesis is devoted to investigate advanced features in the analysis of cryptographic protocols tailored to the Maude-NPA tool. This tool is a model-checker for cryptographic protocol analysis that allows for the incorporation of different equational theories and operates in the unbounded session model without the use of data or control abstraction.

An important contribution of this thesis is relative to theoretical aspects of protocol verification in Maude-NPA. First, we define a forwards operational semantics, using rewriting logic as the theoretical framework and the Maude programming language as tool support. This is the first time that a forwards rewriting-based semantics is given for Maude-NPA. Second, we also study the problem that arises in cryptographic protocol analysis when it is necessary to guarantee that certain terms generated during a state exploration are in normal form with respect to the protocol equational theory.

We also study techniques to extend Maude-NPA capabilities to support the verification of a wider class of protocols and security properties. First, we present a framework to specify and verify sequential protocol compositions in which one or more child protocols make use of infor-

mation obtained from running a parent protocol. Second, we present a theoretical framework to specify and verify protocol indistinguishability in Maude-NPA. This kind of properties aim to verify that an attacker cannot distinguish between two versions of a protocol: for example, one using one secret and one using another, as it happens in electronic voting protocols.

Finally, this thesis contributes to improve the efficiency of protocol verification in Maude-NPA. We define several techniques which drastically reduce the state space, and can often yield a finite state space, so that whether the desired security property holds or not can in fact be decided automatically, in spite of the general undecidability of such problems.

Resumen

El área de análisis formal de protocolos criptográficos ha experimentado una gran actividad desde mediados de los 80. El objetivo es verificar protocolos que utilizan un mecanismo de cifrado para garantizar la confidencialidad y la autenticación de los datos. Los métodos formales han sido utilizados en el análisis de protocolos para proporcionar pruebas formales de seguridad y para descubrir errores y flujos de seguridad que en algunos casos han permanecido ocultos durante mucho tiempo después de la publicación del protocolo original, como es el caso del conocido protocolo Needham-Schroeder Public Key (NSPK). En esta tesis abordamos problemas relacionados con los tres pilares principales de la verificación de protocolos: capacidades de modelado, propiedades verificables y eficiencia.

Esta tesis está dedicada a investigar características avanzadas del análisis de protocolos criptográficos, centrándose en la herramienta Maude-NPA. Esta herramienta es un comprobador de modelos (model-checker) para el análisis de protocolos criptográficos que permite la incorporación de distintas teorías ecuacionales y que opera en el modelo de número ilimitado de sesiones, sin realizar ningún tipo de abstracción de datos o de control.

Una contribución importante de esta tesis está relacionada con aspectos teóricos de verificación de protocolos en Maude-NPA. En primer lugar, definimos una semántica operacional hacia adelante, usando la lógica de reescritura como marco teórico y el lenguaje de programación Maude como herramienta de soporte. Esta es la primera vez que se define una semántica operacional hacia adelante basada en reescritura para Maude-NPA. En segundo lugar, estudiamos el problema que surge en el análisis de protocolos criptográficos cuando es necesario garantizar que determinados términos generados durante la exploración de estados están

en forma normal con respecto a la teoría ecuacional del protocolo.

También estudiamos técnicas para extender las capacidades de Maude-NPA para que se pueda verificar un abanico más amplio de protocolos y de propiedades de seguridad. En primer lugar, presentamos un marco para especificar y verificar composiciones secuenciales de protocolos en las que uno o más protocolos "hijo" hacen uso de información obtenida después de ejecutar un protocolo "padre". En segundo lugar, presentamos un marco teórico para especificar y verificar indistinguibilidad de protocolos en Maude-NPA. El objetivo de este tipo de propiedades es verificar que un atacante no puede distinguir dos versiones diferentes de un protocolo: por ejemplo, una en la que se utiliza un secreto y otra en la que se utiliza un secreto diferente, como ocurre en los protocolos de voto electrónico.

Por último, esta tesis contribuye a mejorar la eficiencia de la verificación de protocolos en Maude-NPA. Definimos varias técnicas que reducen drásticamente el espacio de búsqueda generado en el análisis de un protocolo, y que, a menudo, permite obtener un espacio de búsqueda finito de tal modo que se puede decidir automáticamente si la propiedad de seguridad deseada se satisface o no, a pesar de que tales problemas sean generalmente indecidibles.

Resum

L'àrea d'anàlisi formal de protocols criptogràfics ha experimentat una gran activitat des de mitjan 80. L'objectiu és verificar protocols que utilitzen un mecanisme de xifrat per a garantir la confidencialitat i l'autenticació de les dades. Els mètodes formals han sigut utilitzats en l'anàlisi de protocols per a proporcionar proves formals de seguretat i per a descobrir errors i fluxos de seguretat que en alguns casos han romàs ocults durant molt temps després de la publicació del protocol original, com és el cas del conegut protocol Needham-Schroeder Public Key (NSPK). En aquesta tesi abordem problemes relacionats amb els tres pilars principals de la verificació de protocols: capacitats de modelatge, propietats verificables i eficiència.

Aquesta tesi està dedicada a investigar característiques avançades de l'anàlisi de protocols criptogràfics, centrant-se en l'eina Maude-NPA. Aquesta eina és un comprobador de models (model-checker) per a l'anàlisi de protocols criptogràfics que permet la incorporació de diferents teories equationals i que opera en el model de nombre il·limitat de sessions sense realitzar cap tipus d'abstracció de dades o de control.

Una contribució important d'aquesta tesi està relacionada amb aspectes teòrics de verificació de protocols en Maude-NPA. En primer lloc, definim una semàntica operacional cap a avant, usant la lògica de reescriptura com a marc teòric i el llenguatge de programació Maude com a eina de suport. Aquesta és la primera vegada que es defineix una semàntica operacional cap a avant basada en reescriptura per a Maude-NPA. En segon lloc, estudiem el problema que sorgeix en l'anàlisi de protocols criptogràfics quan és necessari garantir que determinats termes generats durant l'exploració d'estats estan en forma normal respecte a la teoria equacional del protocol.

També estudiem tècniques per a estendre les capacitats de Maude-

NPA perquè es puga verificar un ventall més ampli de protocols i de propietats de seguretat. En primer lloc, presentem un marc per a especificar i verificar composicions seqüencials de protocols en les quals un o més protocols "fill" fan ús d'informació obtinguda després d'executar un protocol "pare". En segon lloc, presentem un marc teòric per a especificar i verificar indistinguibilitat de protocols en Maude-NPA. L'objectiu d'aquest tipus de propietats és verificar que un atacant no pot distingir dues versions diferents d'un protocol: per exemple, una en la qual s'utilitza un secret i una altra en la qual s'utilitza un secret diferent, com ocorre en els protocols de vot electrònic.

Finalment, aquesta tesi contribueix a millorar l'eficiència de la verificació de protocols en Maude-NPA. Definim diverses tècniques que redueixen dràsticament l'espai de cerca generat en l'anàlisi d'un protocol, i que, sovint, permet obtenir un espai de cerca finit de tal manera que es pot decidir automàticament si la propietat de seguretat desitjada es satisfà o no, a pesar que tals problemes siguen generalment indecidibles.

Acknowledgements

I still remember the moment when this thesis begun. I was walking thoughtfully, wondering whether I would be capable to complete a PhD and, at some point, I realized that the question was actually "why not?". This has been a long journey, but also a fulfilling and exciting experience that has changed my perspective of life in many aspects. Sometimes doing research might result too challenging but this is, indeed, its beauty since accomplishing those challenges is so rewarding that it makes our work worthwhile. As I was adviced once, the key is to always *keep working hard*, no matter what happens, specially in the "not so good" moments.

Many people have contributed to make my PhD student period one of the best times of my life and I would like to express my gratitude to them.

First of all I would like to thank my supervisor, Santiago Escobar, for his excellent guidance and dedication during these years. His passion for research is a source of inspiration for any young researcher and I feel fortunate to have been his PhD student. I am truly thankful to him for giving me the opportunity to do this thesis.

I am indebted to Prof. José Meseguer and Prof. Catherine Meadows, with whom I have had the pleasure to collaborate these years. I can only feel gratitude to them for allowing me to enjoy such an enriching experience.

Thanks also to Prof. Carolyn Talcott for offering me the opportunity to visit her at SRI. That was my first contact with research, which convinced me to begin the PhD.

I would also like to thank María Alpuente, the leader of the ELP research group and the person that taught me formal methods for the first time. Her enthusiasm for this research field was a key motivation to join the ELP group.

I am thankful to the members of the ELP and MiST research groups for providing a friendly atmosphere. I would specially like to thank Bea, Toni, Tama, Rafa, Michele and Pepe for their friendship in my first years of PhD. Their advices and experience have been very helpful during these years. I am also grateful to Marco, Nando, César, Fran, David, Javier, Javi, Laura, Antonio and Julia for those great moments we have spent together discussing about interesting ideas, sharing experiences, or simply making fun of stupid things. I would also like to thank Alicia and Raúl for their friendship from the very beginning, and for many interesting discussions during lunch times and coffee breaks.

Furthermore, I cannot forget all the people I met during my research stay at UIUC, from whom I have learnt so many things: Camilo, Kyungmin, Stephen, Nana, Lex, Edgar, Chelsea, Max, Antonio, and, specially, Raúl and Ralf, who were very helpful and kind with me. Thanks to them, that stay was fulfilling not only scientifically, but also personally.

Special mention goes to other friends here in the department: Lucía, Clara, the three Sergio's, Alejandro, Pablo and Víctor, which have played an important role in making these last years such a great time. We have shared our happiness, our worries, our doubts, our experiences, but specially, our sense of humor to overcome whatever difficulty we have found.

Many thanks also to my non-researchers friends for their support and interest. It is very pleasant being surrounded by people that cares about your work even though they have no connection to research.

I would like to thank my family for their unconditional support and love, and specially my parents, for their dedication since I was a child. I would never have reached this stage without them.

Finally, I owe a big thanks to Raúl Costa, the best possible companion in this journey. His always optimistic mood enlightens anyone who is at his side. Thanks for your patience and for always encouraging me to keep on going.

Sonia Santiago
Valencia, November 2014

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The area of formal analysis of cryptographic protocols has been an active one since the mid 80's. The idea is to verify communication protocols that use encryption to guarantee secrecy and that use authentication of data to ensure security. Whatever approach is taken, the use of formal methods has had a long history, not only for providing formal proofs of security, but also for uncovering bugs and security flaws that in some cases had remained unknown long after the original protocol's publication. This is the case, for example, of the Needham-Schroeder Public Key (NSPK) protocol [Needham and Schroeder, 1978], a cryptographic protocol that was intended to provide mutual authentication between two parties communicating on a network. However, this protocol was proved to be subject to a man-in-the-middle attack 20 years later in [Lowe, 1996], using the FDR model-checking tool. A fixed version of this protocol, called Needham-Schroeder-Lowe (NSL), was also presented in that paper.

In this thesis we tackle problems in the three main pillars of protocol verification: modelling capabilities, verifiable properties, and efficiency.

1.1 Formal Analysis of Cryptographic Protocols

Cryptographic protocols can be analyzed on two levels. On one level they can be modeled as communication protocols that must operate over

a hostile network that is controlled by an attacker who is trying to subvert the goals of the protocol. At this level the attacker is allowed to apply an unlimited number of operations taken from a finite set: e.g. encryption, decryption, digital signature generation, and so on. This class of models is usually referred to as the *Dolev-Yao* model, after the paper by Dolev and Yao [1983] that introduced this approach.

At another level cryptographic protocols may be thought of as cryptographic systems being attacked by a probabilistic polynomial-time adversary. This class of models is usually referred to as the *computational model*.

Both types of models have their advantages. The Dolev-Yao model supports the use of model checkers that can search exhaustively through different system traces. This makes it very good at finding attacks that involve interleaving of different protocol executions against each other. These types of attacks are generally nonintuitive and hard to identify manually. On the other hand, the computational model gives a much more fine-grained model of the attacker capabilities, and allows one to reduce the problem of breaking the protocol to breaking the cryptosystems used. Cryptographic protocol analysis tools based on model-checking discover attacks by generating and analyzing the search space of possible states arising from the execution of a given protocol, taking into consideration the functions of the honest principals as well as the capabilities of the intruder(s). The earliest protocol analysis tools, such as the Interrogator [Millen et al., 1987] and the NRL Protocol Analyzer (NPA) [Meadows, 1996b], while not strictly speaking of model checkers, relied on state exploration, and, in the case of NPA, could be used to verify security properties specified in a temporal logic language. Later, researchers used generic model checkers to analyze protocols, such as FDR [Lowe, 1996] and later Murphi [Mitchell et al., 1997].

More recently the focus has been on special-purpose model checkers developed specifically for cryptographic protocol analysis, such as ProVerif [Blanchet, 2001], the AVISPA tool [Armando et al., 2005], and Maude-NPA itself [Escobar et al., 2012a, 2009a].

There are a number of possible approaches to take in the modeling of cryptoalgorithms used. In the simplest case, the free algebra model, cryptosystems are assumed to behave like black boxes: an attacker knows nothing about encrypted data unless it has the appropriate key. This

is the approach taken, for example, by the above-cited use of Murphi and FDR to analyze cryptographic protocols, and current tools such as SATMC [Armando et al., 2014] and TA4SP [Boichut et al., 2004], both used in the AVISPA tool. However, such an approach, although it can work well for protocols based on generic shared key or public key cryptography, runs into problems with algorithms such as Diffie-Hellman exponentiation or algorithms employing exclusive-or, which rely upon various algebraic properties such as the law of exponentiation of products, associativity-commutativity and cancellation. Without the ability to specify these properties, one needs to rely on approximations of the algorithms that may result in formal proofs of secrecy invalidated by actual attacks that are missed by the analysis (see, e.g., [Paulson, 1998; Ryan and Schneider, 1998; Stubblebine and Meadows, 2000]). Thus there has been considerable interest in developing algorithms and tools for protocol analysis in the presence of algebraic theories [Abadi and Cortier, 2006; Chevalier and Rusinowitch, 2008; Bursuc and Comon-Lundh, 2009; Ştefan Ciobâcă et al., 2012; Baudet et al., 2013].

Another way in which tools can differ is in the number of sessions. A *session* is defined to be one execution of a protocol role by a single principal. A tool is said to use the *bounded session model* if the user must specify the maximum number of sessions that can be generated in a search. It is said to use the *unbounded session model* if no such restrictions are required.

Secrecy is known to be decidable in the free theory together with the bounded session model [Rusinowitch and Turuani, 2001], and undecidable in the free theory together with the unbounded session model [Durgin et al., 2004]. The same distinction between bounded and unbounded sessions is known to hold for a number of different equational theories of interest, as well as for some authentication-related properties; see for example [Bursuc and Comon-Lundh, 2009]. Thus, it is no surprise that most tools, whether or not they offer support for different algebraic theories, either operate in the bounded session model, or rely on abstractions that may result in reports of false attacks even when the protocol being analyzed is secure. Tools working on an unbounded session model making no data abstraction cannot guarantee the termination of the protocol analysis.

Tools can also differ in the "direction" of the search, that is, for-

wards or backwards search. Many of the earlier approaches [Lowe, 1996; Mitchell et al., 1997; Clarke et al., 2000] made use of explicit-state model-checking using forward search. More recently the emphasis has been on symbolic-state model-checking, in which states are represented by terms containing variables [Blanchet, 2001; Escobar et al., 2009a; Basin et al., 2005; Mödersheim and Viganò, 2009; Cremers, 2008a; Meier et al., 2013]. Here state transitions are computed using unification or constraint based techniques, and search is generally performed backwards from a symbolic specification of an insecure state. This approach has many advantages. In particular, the combination of symbolic states and goal-directed backwards search can result in a smaller search space.

1.2 Protocol Analysis modulo Equational Theories

The analysis of cryptographic protocols implies that the analysis must be performed *modulo* the algebraic properties of their underlying cryptographic functions. These range from the fact that decryption with a key cancels out encryption with the same key, expressible by the equation

$$dec(enc(m, k), k) = m \tag{1.1}$$

through the Abelian group properties of algorithms based on exponentiation and/or elliptic curves, all the way to the property of homomorphism over an Abelian group possessed by many of the algorithms used for privacy-preserving computation.

There are two main issues in protocol analysis modulo equational theories: (i) the procedure that is used for dealing with them, and (ii) the class of equational theories supported by that procedure. Those two issues greatly determine what kind of protocols can be analyzed by one technique or another. Regarding the first issue, there are three classes of procedures that have been developed for dealing with algebraic properties: (i) augmented intruder inference rules, (ii) deducibility algorithms, and (iii) equational unification. We explain each of these procedures below.

Augmented intruder inference rules. When specifying a cryptographic protocol, one normally specifies a set of inference rules that describe the operations that an intruder can perform. Thus, one would specify an inference rule that says that, if an intruder knows a message and a key, then he can construct the encryption of the message with the key. These inference rules can also be augmented to describe the consequences of equational properties. For example, the encryption-decryption equation (1.1) could be represented by the following inference rule:

$$\frac{enc(m,k) \in \mathcal{I},\ k \in \mathcal{I}}{m \in \mathcal{I}}$$

where $\mathcal{I}$ stands for the set of terms known to the intruder.

The problem is that this method is often incomplete. Consider the following protocol:

$$1.\ \ A \rightarrow B : M \qquad 2.\ \ B \rightarrow A : dec(M, key(B))$$

The inference rule fails to predict what happens when $M = enc(X, key(B))$. In this case, A would wind up sending a cleartext message X, but this is because of the action of A's decryption operation, not because of the application of an inference rule by the intruder. Thus, the inference rules are not complete, although in some cases there are subclasses of protocols for which given sets of inference rules are sound (see for example [Millen, 2003; Lynch and Meadows, 2005]),

However, if successful, augmented inference rules have the advantage that they can be used with tools that do not support the equational theory that the inference rules represent. This is the case, for example, of the works presented in [Küsters and Truderung, 2011, 2009], and [Kremer and Ryan, 2005], which provide augmented inference systems that can be used together with ProVerif to analyze protocols involving Diffie-Hellman exponentiation, a restricted version of exclusive-or, and cipher block chaining, respectively in a limited way. These kind of protocols are not directly supported by ProVerif.

Deducibility algorithms. The second class of procedures for dealing with equational theories, deduciblity algorithms, determine whether an intruder can deduce a term from a set of terms already in its possession, given that terms obey a given equational theory. In this case,

one starts with a set T of terms known to the intruder, a term t the intruder is trying to learn, a set of inference rules describing operations the intruder can perform, and an equational theory E. A number of algorithms have been developed for different classes of equational theories, including associative-commutative and homomorphic operators [Abadi and Cortier, 2006]. A survey of deducibility with respect to equational theories may be found in [Bursuc and Comon-Lundh, 2009].

In particular, algorithms for a class of theories known as subterm convergent (convergent theories for which the right-hand side is either an irreducible ground term or a subterm of the left-hand side) have been developed in [Ştefan Ciobâcă et al., 2012] and for a larger class of convergent theories that include encryption homomorphic over a free operator [Baudet et al., 2013] and have been developed and implemented in the tools KISS and YAPA, respectively. These tools when used by themselves can only prove security against a passive intruder, who only spies upon message traffic but does not further interact with the protocol. However they can also be interfaced with other tools that use deducibility to reason about security against an active attacker who reads, alters, redirects, and deletes traffic as well as creating its own messages. The *cap unification* approach of Anantharaman et al. [2010] also uses deducibility algorithms for classes of equational theories that extend the subterm convergent class. The tools OFMC [Basin et al., 2005; Mödersheim and Viganò, 2009], and CL-Atse [Turuani, 2006; Arora and Turuani, 2009] all make use of deducibility and provide some support for equational theories, in the case of OFMC and CL-Atse those governing Diffie-Hellman and exclusive-or.

A limitation of using deducibility is that it requires a complete description of the terms an intruder knows at a given state. This is fine for tools that generate states in a forward fashion, but it does not work as well for tools such as Maude-NPA, which generate states on the fly in a backwards manner and thus only are aware of some of the terms the intruder knows at any point in time.

Equational Unification Finally, equational unification based procedures consist in computing protocol execution paths by unifying messages received with messages sent *modulo* an equational theory E describing the protocol's algebraic properties. Unification of two terms s and t mod-

ulo an equational theory E is the process of finding substitutions σ to the variables in s and t making the two terms equal modulo E.

An advantage of E-unification over deducibility is that it can be applied even on incomplete information, since this incomplete information can be represented by variables. Thus, it applies not only to both forward and backwards search, but to constraint-based searches that can proceed from any direction, e.g. [Comon-Lundh and Shmatikov, 2003; Chevalier et al., 2007; Comon-Lundh et al., 2011], and to deducibility procedures for checking whether one term is deducible from a given set of terms, e.g. [Abadi and Cortier, 2006; Baudet et al., 2013; Ştefan Ciobâcă et al., 2012].

Any unification technique used in cryptographic protocol analysis must satisfy two properties. First of all, it must behave well with respect to composition, especially of disjoint theories, since cryptographic protocols often combine different algorithms described by different theories. Although methods for combining unification algorithms of disjoint theories are well-known [Schmidt-Schauß, 1989; Baader and Schulz, 1992], the solution in the general case is highly nondeterministic and inefficient, so more efficient special algorithms are desirable. The second property that must be satisfied is a little more subtle, and has to do with the fact that many of the state space reduction techniques used require that terms in the state be in some kind of normal form with respect to the theory E used. Generally this is expressed by writing E as a decomposition (E_0, B) where B is regular and has a finitary unification algorithm, and E_0 is a set of oriented equations being convergent modulo B. This ensures enough stability in normal form representations of terms so that syntactic state space reduction techniques can be applied.

Variant-based unification Both the first and second desiderata of unification-based cryptographic can be achieved, if the decomposition (B, E_0) has the finite variant (FV) property, via variant unification as described in Section 2.1, below. Since the first step of variant unification requires the computation of all the irreducible variants of each side of the unification problem, and a solution is discarded if a solution makes either side reducible, variant unification guarantees the irreducibility constraint required by state space reduction techniques. Moreover, variant unification behaves well under composition, at least in the area of cryptographic

protocol analysis. First of all, the axioms B are relatively few and well-understood. Moreover, if the combination of the two theories also has a finite variant decomposition, then the same finite variant algorithm can be applied as well.

Not surprisingly many tools have followed approaches similar, if not identical, to variant unification. Both Maude-NPA [Escobar et al., 2009a] and Tamarin [Meier et al., 2013] use variant-based unification explicitly. Indeed support for generic variant-based unification is already built into an upcoming version of Maude. Moreover, other tools have used approaches that have many features in common with variant-based unification. For example, ProVerif (see [Blanchet et al., 2008, Sec. 5]) and OFMC (see [Mödersheim, 2007, Sec. 10]) both compute the variants of protocol rules, modulo the free theory for ProVerif, and modulo the free theory or AC for OFMC. This has the effect of computing the variants of both sides of the unification problem. More recently, variants have been applied to expanding the capacity of ProVerif to deal with AC theories. Thus, in [Küsters and Truderung, 2011] the authors implement a special case of the exclusive-or theory in the ProVerif tool by expressing it as a rewrite theory with the finite variant property with respect to the free theory ($E = \varnothing$) and computing variants that are unified syntactically. This requires some restrictions on the syntax of the protocol, however. Similar approaches have been applied by Küsters and Truderung [2009] for modular exponentiation, and Arapinis et al. [2012] for commuting encryption and AC theories.

Variant-based unification does have some drawbacks, however. First of all, it can be inefficient for theories of high variant complexity. This can be mitigated by the use of *asymmetric unification* (see Chapter 8), in which only the variants of the right-hand side of a unification problem are computed, and irreducibility constraints are also enforced only on the right-hand sides. This requires the use of specialized asymmetric unification algorithms, so combination is no longer as straightforward, but it is still possible to apply the state space reduction techniques, and efficiency gains, as shown in Chapter 8, can be dramatic.

A more serious problem arises when a theory of interest fails to have a finite variant decomposition, at all, i.e. a decomposition of the equational theory that satisfies the FV property. Fortunately, most theories of interest to cryptographic protocol analysis have a decomposition (B, E_0)

satisfying the FV property in which B is either the empty theory or AC. However, there is one notable exception, the theory of encryption homomorphic over another operator, that is $e(X * Y, K) = e(X, K) * e(Y, K)$ where $*$ is an operator that may have some other equational properties, shown not to satisfy the finite variant property when the homomorphic equation is in R in [Comon-Lundh and Delaune, 2005]. This case has been successfully handled in Maude-NPA. The work presented in [Escobar et al., 2011] provided a dedicated unfication for the theory of homomorphic encryption shown above, and an under-approximation to support the AGH property, i.e. the case where $*$ is an Abelian group. More recently, Yang et al. [2014] developed a general strategy to approximate theories without the FV property, by using a combination of under-approximation and over-approximation, to theories having that property. This general strategy was applied to develop a hierarchy of theories approximating homomorphic encryption that are verified to have the finite variant property.

1.3 Reducing the State Search Space

Symbolic analysis of cryptographic protocol usually generates huge, and, even worse, infinite search state spaces. This is the case, for example, of state exploration tools, specially when they operate in the unbounded session model, or reason about equational theories. This issue can become even more critical when the tool does not rely on data abstractions or approximations, such as the Maude-NPA. In such general approaches, protocol security properties are well-known to be undecidable, as explained in Section 1.1. However, any protocol analyzer tool requires a finite state search space finding no attack to prove a protocol secure.

It is therefore very important, both for efficiency and to achieve full verification whenever possible, to use *state-space reduction techniques* that: (i) can drastically cut down the number of states to be explored; and (ii) have in practice a good chance to make the, generally infinite, search space finite without compromising the completeness of the analysis; that is, so that if a protocol is indeed secure, failure to find an attack in such a finite state space guarantees the protocol's security for that attack relative to the assumptions about the intruder actions and the

algebraic properties.

Optimization techniques are used in all protocol analysis tools but are not always well documented in the literature. However, even so, there are a number of exceptions in which particular techniques have been well documented.

One of the most effective techniques to detect unreachable states is the grammar generation technique of NPA [Meadows, 1996b], which is used with very little change in Maude-NPA. The grammar generation can be thought of as implementing something similar to a resolution technique, in which backwards narrowing steps are applied until saturation is achieved. This is probably closest in spirit to the use of resolution to generate a search space, for example, in ProVerif [Blanchet, 2001] or SPASS [Weidenbach, 1999]. In these tools actions of principals are represented as Horn clauses, and a form of resolution (resolution with free selection in ProVerif and ordered resolution in SPASS), until saturation is achieved. The application of the technique, of course, is very different, since ProVerif and SPASS use resolution to generate a search space, while in the generation of grammars the goal is to give a finite description of a set of words not learnable by the intruder. It is perhaps closer in intent to a heuristic used in the Scyther tool [Cremers, 2006, 2008b] to recognize cyclical dependencies among terms sent in messages, although Scyther uses a technique that is closer to Maude-NPA's "intruder-learns-only-once" rule, that allows the intruder to learn a term only once.

Partial order techniques are also well-known reduction techniques used in any state exploration tool. In the context of protocol analysis, the most promminent use of these kind of techniques is to give priority to some kind of messages. For example, NPA and Maude-NPA, because of their backwards search, give prioritiy to input messages, whereas the work of Shmatikov and Stern [1998] gives priority to output messages for forward search. Indeed, Basin et al. [2013] mentions partial order reduction as a natural optimization technique in the context of model-checking of security protocols.

The use of heuristics to identify unreachable states is probably the least well documented of optimization techniques. However we note that Cremers [2008b] describes a mechanism used in the Scyther tool for identifying the case in which a nonce is used before it has been generated. NPA also had a mechanism for doing this, which was very similar to the

one used in Maude-NPA. The main difference among the techniques used by NPA and Maude-NPA is that Maude-NPA's use of the strand space model often allows to identify these anomalous states before the event in question has actually been produced in the narrowing sequence.

The *lazy intruder*, first proposed by Huima [1999] [1], and later expanded upon by a number of others, for example [Amadio and Lopez, 2000; Millen and Shmatikov, 2001; Chevalier et al., 2008], is a technique used in connection with constraint-based protocol analysis. It arises from the fact that the actual value of a certain part of a message may be irrelevant to the receiver, for example, when the receiver will simply pass it on without interpreting it. Determining all the ways an intruder could construct a message would lead to a needless state space explosion. Thus, instead the decision is postponed by replacing the "irrelevant" part of the message with a variable and recording, as a constraint, the information on which knowledge the intruder used to generate the message. Because the relevant information is carried along with the variable, the solving of the constraint can be delayed until more information is known about how the lazy term will be used by a recipient of a message containing it.

The concept of lazy intruder used in constraint-based analysis has been incorporated in Maude-NPA's super-lazy intruder technique. More specifically, Maude-NPA delays finding how to reach certain 'super-lazy" terms in the intruder knowledge about which little is known, until one or more variables in the term are further instantiated. However, Maude-NPA's search method differs from that done in constraint-based systems in that the evaluation of a constraint can take place at any point in a search, while in Maude-NPA a search for a term must be executed at the point in the search tree corresponding to the time the term was learned by the intruder. This means that if a search for a term is delayed until it has been further instantiated, the state must be "rolled back" to the point in the search tree at which the term was learned in the intruder. The super-lazy intruder technique was also used by NPA, but the "roll back" process was performed in a different way as it is done in Maude-NPA.

The *lazy intruder* technique has been connected with partial order reductions. For example, *constraint differentiation* [Mödersheim et al., 2010] works by identifying overlaps between the constraints that arise in the application of the lazy intruder technique. If the constraints belong-

[1] although not under that name, which was coined by Basin et al. [2003]

ing to two different states overlap, then the constraints in the overlap are ultimately applied to only one of the states. Maude-NPA's subsumption partial order can be thought of as taking a similar approach, in which substitutions are being compared instead of constraints, and instead of identifying overlaps, one identifies cases in which one state subsumes another, which for constraint differentiation would correspond to the case in which one set of constraints contains another.

1.4 Protocol Composition

Traditionally, the analysis of cryptographic protocols has focused on individual protocols. But protocols do not always work alone, but together, one protocol relying on another to provide needed services. However it is well known that many problems in the security of cryptographic protocols arise when the protocols are composed. This is true whether the composition is parallel, in which two different protocols are executed in an interleaved fashion, or sequential, in which one or more child protocols use information from executing a parent protocols. 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. Security of parallel composition can generally be achieved by avoiding ambiguity about which protocol a message belongs to (as in, e.g. [Guttman and Thayer, 2000; Ştefan Ciobâcă and Cortier, 2010]). The necessary conditions for security of sequential composition are harder to pin down, since they depend on the guarantees offered and needed by the particular protocols being analyzed.

The importance of understanding composition has long been acknowledged, and there have been two approaches to this problem. One, called *nondestructive* composition in [Datta et al., 2003], 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 [1998], Guttman and Thayer [2000], Cortier and Delaune [2009a], Ştefan Ciobâcă and Cortier [2010], Groß and Mödersheim [2011], and, in the computational model, the Universal Composability of Canetti et al. [2002]. 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.

The other approach, called *additive* composition in [Datta et al., 2003] addresses the compositionality of the protocol properties themselves. This addresses the development of logical systems and tools in which inference rules are provided for deriving complex properties of a protocol from simpler ones. The Protocol Composition Logic (PCL) begun in [Durgin et al., 2001] is probably the first protocol logic to approach composition in a systematic way. Logics such as the Protocol Derivation Logic (PDL) [Cervesato et al., 2005], and tools such as the Protocol Derivation Assistant (PDA) [Anlauff et al., 2006] and the Cryptographic Protocol Shape Analyzer (CPSA) [Doghim et al., 2007] 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 et al., 2003] PCL and in [Guttman, 2001] 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 helps 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. For example, in the Internet Key Exchange Protocol [Harkins and Carrel, 1998] there are sixteen different one-to-many parent-child compositions of Phase One and Phase Two protocols. The ability to synthesize compositions auto-

matically would greatly simplify the specification and analysis of protocols like these. However, very little work has been done to address this problem. Indeed, to the best of our knowledge, most protocol analysis model-checking tools simply use concatenation of protocol specifications to express sequential composition.

We have defined a framework to support the specification and verification of sequential protocol compositions in Maude-NPA. First, in [Escobar et al., 2010] we presented a technique in which protocol compositions were handled via a protocol transformation, so that it was not necessary to modify Maude-NPA. More recently, we have provided a direct implementation of protocol composition in Maude-NPA by extending its operational semantics.

1.5 Security Properties

Traditionally, properties proved about Dolev-Yao specifications of protocols fall into two classes: secrecy and correspondence. *Correspondence* holds if, whenever certain actions have occurred, one can guarantee that certain other actions have occurred in a specified order; correspondence properties are thus used to reason about authentication. *Secrecy* (also called *simple secrecy*) holds for a term if the attacker never sees that term in the clear. This is a much weaker property than secrecy in the computational model, which usually relies on proving that an adversary cannot distinguish between two versions of the protocol: for example, one using one secret and one using another, or one using an encrypted secret and one using random data.

Recently the interest in formulating and applying indistinguishability properties for Dolev-Yao models has been growing. There are a number of reasons for this. The first is that cryptography has advanced to the point at which it is not only possible to provide computational proofs of security for algorithms, but for the protocols that use those algorithms as well. If Dolev-Yao tools can be extended to prove indistinguishability, this increases the likelihood that both approaches can be used together in an effective way to ensure protocol security. The second is that there is a growing class of privacy-protection protocols for which simple secrecy is clearly inadequate. Such protocols protect low-entropy data such as

votes, medical records, or network routes; even partial leakage of this information could be harmful. The third is the result of recent work on automatic generation of cryptographic algorithms. In this work, multiple possible algorithms are generated out of a library of components and then checked for security. This may involve the use of Dolev-Yao like tools to weed out insecure algorithms or even verify the security of correct ones, as in [Barthe et al., 2013].

Work in extending the Dolev-Yao model to support the definition and verification of indistinguishability properties goes as far back as the early eighties, when Michael Merritt developed a theory of *hidden automorphisms* [Merritt, 1984]. The first to apply a tool to analyze protocols for indistinguishability was Gavin Lowe [Lowe, 2004], who used the FDR model checker to analyze security of password-base protocols against off-line guessing attacks.

In Abadi and Fournet [2001] the authors gave the definition of two kinds of indistinguishability: static equivalence and observational equivalence, in terms of the applied π-calculus presented in that paper. Roughly speaking, static equivalence describes a passive observer's inability to distinguish between two protocols, while observational equivalence describes an active attacker's inability to distinguish between two protocols.

More recently Cortier and Delaune [2009b] have shown that *trace equivalence* implies observational equivalence for *determinate* applied π-calculus processes; roughly speaking, a process is determinate if it exhibits no non-deterministic choice points, and two processes are trace equivalent if for any trace produced by one process there is a trace produced by the other process indistinguishable from the first trace by the attacker.

Trace equivalence is decidable in the bounded session model, and a number of algorithms and tools have been developed, covering a wide class of equational theories [Baudet, 2005; Cheval et al., 2010, 2011; Chadha et al., 2012; Cheval et al., 2013]. However to our knowledge there has been very little work (if any) on trace equivalence involving AC theories.

Although trace equivalence is decidable for the bounded session model, it is not straightforward to implement in search-based tools that are typically used to evaluate cryptographic protocols. This is because trace equivalence is an example of a *hyperproperty* [Clarkson and Schneider,

2010]: it is not defined in terms of sets of traces, but sets of pairs of traces, and thus cannot be defined in terms of reachability or unreachability of particular classes of states. However, integration of indistinguishability into state exploration tools has a number of potential benefits, since one automatically obtains support for whatever feature the tool offers, e.g. support for the unbounded session model and equational theories involving AC.

Checking for hyperproperties such as trace or observational equivalence can be implemented in a search-based tool by specifying a stronger property that can be formulated in terms of state reachability. Such an approach was taken by Lowe [2004]; a protocol was secure against guessing attacks if the attacker could not generate certain types of terms. This was later shown in [Newcomb and Lowe, 2005] to imply a property similar to the observational equivalence of Abadi and Fournet [2001].

The most prominent application of this approach to cryptographic protocol verification has been in the ProVerif tool via the notion of *uniformity*, shown to imply observational equivalence in [Blanchet et al., 2008]. It is used to define the indistinguishability of two processes that differ only in certain terms. Uniformity requires that the two processes be executed in lock-step as a *bi-process* and projection of the bi-process to each of its components is a bisimulation. The authors prove that uniformity is equivalent to a state unreachability property, and thus can be evaluated using ProVerif. ProVerif can be used to verify uniformity for *subterm convergent* rewrite theories; Arapinis et al. [2012] have developed methods for extending this to some theories that include a restricted encoding of AC axioms (in particular, some terms must be ground).

We have recently proposed in [Santiago et al., 2014b] an intuitive notion of indistinguishability related to the notion of *uniformity* used in ProVerif [Blanchet et al., 2008]; we define a pairing between two protocols and define security in terms of reachability conditions on the protocol pairing. One major difference is in the support of AC theories without any encoding restrictions, as long as they have decompositions with the finite variant property. This is inherited from Maude-NPA. There are also differences in the approach we take to specification and implementation of security properties. In ProVerif, the intruder is given the ability to evaluate a predicate that outputs "bad" if there is a violation of uniformity. One then checks for uniformity by proving that no state containing

"bad" is reachable. This gives ProVerif the ability to reduce everything to just one property.

In our approach, we use an unmodified Dolev-Yao intruder with no ability to evaluate predicates. This is motivated by our preference to avoid increasing the complexity of Maude-NPA's Dolev-Yao model unless absolutely necessary, and thus to express our security requirements in the original Maude-NPA framework.

1.6 Contributions

In this thesis we present different techniques that contribute both to improve the performance and to extend the capabilities of the Maude-NPA tool, a model checker for cryptographic protocol analysis that both allows for the incorporation of different equational theories and operates in the unbounded session model without the use of data or control abstraction. We detail these contributions below, following the order in which they appear.

(i) In Chapter 4, we have defined several *state space reduction techniques* that we have implemented in Maude-NPA, and provide completeness proofs and experimental evaluations. Since Maude-NPA allows reasoning in the unbounded session model, and because it allows reasoning about different equational theories (which typically generate many more solutions to unification problems than syntactic unification, leading to bigger state spaces), it is necessary to find ways of pruning the search space in order to prevent infinite or overwhelmingly large search spaces. The combination of these techniques allows Maude-NPA to obtain a *finite* state space for all protocols in the experiments, whereas the state space will be infinite without our optimizations. This is very important for Maude-NPA, which can prove a protocol secure if it obtains a finite state space finding no attacks. All these state space reduction techniques have been implictly used in the experiments performed in Chapters 6, 7, and 8. These results have been published in [Escobar et al., 2014a].

(ii) In Chapter 5, we have defined a *rewriting-based forwards semantics* for Maude-NPA. Maude-NPA has a backwards operational semantics. Therefore, it already has an intuitive forwards semantics

obtained by reversing the rewrite rules defining this backwards semantics. However, such intuitive forwards semantics is not suitable for model-checking, and a better approach is taken. This rewriting-based forwards semantics can be applied for purposes not suitable for the backwards semantics, switching back and forth between both approaches. For example, this forwards semantics has become vital to provide a formal definition of protocol indistinguishability in Maude-NPA (see Chapter 7). The experimental evaluation performed validates the feasiblity of this forwards semantics. These results have been published in [Escobar et al., 2014b].

(iii) In Chapter 6 we provide a framework to support *dynamic sequential protocol compositions* in Maude-NPA, i.e., protocols are specified in a modular way and can be composed when desired during the verification process. More specifically, we present two different techniques to support sequential protocol compositions, one that does not require to modify the tool, and another one for which is necessary to extend Maude-NPA's operational semantics. We compare both techniques showing the advantages of each approach, and providing experimental results of their performance. This new framework clearly extends the capabilities of Maude-NPA, since it allows the verification of protocols, not supported before, or, at least, not in such a modular way. These results have been published in [Escobar et al., 2010; Santiago et al., 2014a].

(iv) In Chapter 7, we provide a theoretical framework to specify and verify *protocol indistinguishability* in Maude-NPA, thus allowing the tool to verify a class of properties not supported before. To the best of our knowledge, this is the first time that indistinguishability has been defined for the strand space model. Moreover, our framework supports the class of equational theories with a finite variant decomposition which contains a large number of theories of interest to cryptographic protocol analysis, including exclusive-or, Abelian groups, and modular exponentiation. We have performed a preliminar experimental evaluation of this technique obtaining encouraging results. Our experiments include protocols involving AC theories such as exclusive-or, which, as far was we know, is not supported by any other existing tool. These results have been

published in [Santiago et al., 2014b].

(v) In Chapter 8 we define *asymmetric unification*, a new unification paradigm with interesting properties for cryptographic protocol analysis, and provide a method to convert standard equational unification algorithms into asymmetric algorithms, illustrating it with an asymmetric version of an exclusive-or unification algorithm. Basically, an asymmetric unification problem is a standard unification problem in which the right hand side is irreducible. This is very interesting for most cryptographic protocols analysis tools, since their optimizations usually rely on the assumption that received messages are in normal form. For example, Maude-NPA's state reduction techniques of Chapter 4 require such irreducibility constraints. Asymmetric unification allows the development of a tool-independent symbolic state exploration algorithm that preservers irreducibility constraints. We illustrate the benefits of asymmetric unification for crytographic protocol analysis w.r.t. standard unification with the results obtained in the experimental evaluation of the asymmetric unification algorithm for exclusive-or. These results have been published in [Erbatur et al., 2012, 2013].

1.7 Plan of the Thesis

The chapters of this thesis are not organized following the three main pillars of protocol verification mentioned above. Instead, chapters are presented in an order that we believe may facilitate the reading and understanding of this thesis.

Therefore, this thesis is organized as follows. First, we recall some preliminaries necessary for the understanding of the thesis in Chapter 2. In Chapter 3 we provide detailed information on Maude-NPA, the protocol analyzer tool in which the techniques presented in this thesis have been implemented. Chapter 4 describes the techniques implemented to reduce the state search space generated by Maude-NPA. Chapter 5 is devoted to the rewriting-based forwards semantics defined for Maude-NPA. Chapter 6 explains two approaches that allow the specification and analysis of sequential protocol compositions in Maude-NPA. In Chapter 7 we provide a formal definition of protocol indistinguishability in Maude-NPA,

and explain how this kind of properties can be verified in the tool. Chapter 8 presents asymmetric unification, a new unification paradigm that allows to consider irreducibilty conditions necessary in protocol analysis tools. Each chapter includes some conclusions, but, in Chapter 9, we conclude this thesis, summarizing the contributions, and providing some perspectives on how to extend the work presented in this thesis.

The publications derived from this thesis are:

- [Escobar et al., 2010] Santiago Escobar, Catherine Meadows, José Meseguer, and Sonia Santiago. Sequential Protocol Composition in Maude-NPA. In Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou, editors, *ESORICS*, volume 6345 of *Lecture Notes in Computer Science*, pages 303–318. Springer, 2010. ISBN 978-3-642-15496-6.

  Acceptance rate: 42/201 (20%) - CORE 2013: A

- [Erbatur et al., 2012] Serdar Erbatur, Santiago Escobar, Deepak Kapur, Zhiqiang Liu, Christopher Lynch, Catherine Meadows, José Meseguer, Paliath Narendran, Sonia Santiago, and Ralf Sasse. Effective symbolic protocol analysis via equational irreducibility conditions. In Sara Foresti, Moti Yung, and Fabio Martinelli, editors, *ESORICS*, volume 7459 of *Lecture Notes in Computer Science*, pages 73–90. Springer, 2012. ISBN 978-3-642-33166-4.

  Acceptance rate: 50/248 (20%) - CORE 2013: A

- [Erbatur et al., 2013] Serdar Erbatur, Santiago Escobar, Deepak Kapur, Zhiqiang Liu, Christopher Lynch, Catherine Meadows, José Meseguer, Paliath Narendran, Sonia Santiago, and Ralf Sasse. Asymmetric unification: A new unification paradigm for cryptographic protocol analysis. In Maria Paola Bonacina, editor, *CADE*, volume 7898 of *Lecture Notes in Computer Science*, pages 231–248. Springer, 2013. ISBN 978-3-642-38573-5.

  Acceptance rate: 22/53 (41%) - CORE 2013: A

- [Escobar et al., 2014b] Santiago Escobar, Catherine Meadows, José Meseguer, and Sonia Santiago. A rewriting-based forwards semantics for Maude-NPA. In proceedings of *I Symposium and Bootcamp on the Science of Security (HotSoS 2014). IEEE digital library*, 2014. To appear.

- [Escobar et al., 2014a] Santiago Escobar, Catherine Meadows, José Meseguer, and Sonia Santiago. State space reduction in the Maude-NRL Protocol Analyzer. *Information and Computation*, 238:157–186, (2014).

  JCR Impact Factor 2012: 0.699 -
  JCR 5-Year Impact Factor 2012: 0.890 - ERA 2010 CORE B

- [Santiago et al., 2014b] Sonia Santiago, Santiago Escobar, Catherine Meadows, and José Meseguer. A formal definition of protocol indistinguishability and its verification using Maude-NPA. In Sjouke Mauw and Christian Damsgaard Jensen, editors, In proceedings of *Security and Trust Management - 10th International Workshop, STM 2014, Wroclaw, Poland, September 10-11, 2014.*, volume 8743 of *Lecture Notes in Computer Science*, pages 162–177. Springer, 2014. ISBN 978-3-319-11850-5.

  Acceptance rate (regular papers) : 11/29 (37%)

- [Santiago et al., 2014a] Sonia Santiago, Santiago Escobar, Catherine Meadows, and José Meseguer. Sequential protocol composition in Maude-NPA. Submitted to *Journal of Computer Security*, 2014.

Chapter 2

Preliminaries

In this chapter we provide some technical background necessary to understand this thesis. More specifically, Section 2.1 recalls standard notions and terminology of term rewriting. Section 2.2 gives an overview about narrowing modulo equations of Meseguer and Thati [2007] using topmost rewriting as a tool-independent semantic framework for symbolic reachability analysis of protocols under algebraic properties. Finally, Section 2.3 is devoted to Maude [Clavel et al., 2007], a very efficient implementation of *Rewriting Logic* [Meseguer, 1992], upon which Maude-NPA is implemented.

2.1 Rewriting Logic and Term Rewriting

In this section we recall the standard notions and terminology of term rewriting. We follow the classical notation and terminology from [TeReSe, 2003] for term rewriting and from [Meseguer, 1992, 1997] for rewriting logic and order-sorted notions.

Terms, sorts and positions

We assume an *order-sorted signature* $\mathbf{\Sigma} = (\mathsf{S}, \leqslant, \Sigma)$ with poset of sorts $(\mathsf{S}, \leqslant)$, and a finite number of function symbols. We also 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. $\mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ is the set of terms of sort s, and $\mathcal{T}_{\Sigma,\mathsf{s}}$ is the set of ground terms of sort s. We write $\mathcal{T}_\Sigma(\mathcal{X})$ and $\mathcal{T}_\Sigma$ for the corresponding order-

sorted 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.

Substitutions, matching, and unification

A *substitution* σ is a sort-preserving mapping from a finite subset of $\mathcal{X}$, written $\mathcal{D}om(\sigma)$, to $\mathcal{T}_\Sigma(\mathcal{X})$. Substitutions are written as $\sigma = \{X_1 \mapsto t_1, \ldots, X_n \mapsto t_n\}$ where the domain of σ is $Dom(\sigma) = \{X_1, \ldots, X_n\}$ and the set of variables introduced by terms $t_1, \ldots, t_n$ is written $Ran(\sigma)$. The identity substitution is *id*. Substitutions are homomorphically extended to $\mathcal{T}_\Sigma(\mathcal{X})$. The application of a substitution σ to a term t is denoted by $t\sigma$. For simplicity, we assume that every substitution is idempotent, i.e., σ satisfies $Dom(\sigma) \cap Ran(\sigma) = \varnothing$. Substitution idempotency ensures $t\sigma = (t\sigma)\sigma$. The restriction of σ to a set of variables V is $\sigma|_V$. Composition of two substitutions σ and σ' is denoted by $\sigma \circ \sigma'$.

A Σ-*equation* is an unoriented pair $t = t'$, where $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ for some sort $\mathsf{s} \in \mathsf{S}$. Given Σ and 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, 1997]). The E-equivalence class of a term t is denoted by $[t]_E$, and $\mathcal{T}_{\Sigma/E}(\mathcal{X})$ and $\mathcal{T}_{\Sigma/E}$ denote the corresponding order-sorted term algebras modulo E. Throughout this thesis we assume that $\mathcal{T}_{\Sigma,\mathsf{s}} \neq \varnothing$ for every sort s, because this affords a simpler deduction system. An *equational theory* (Σ, E) is a pair with Σ an order-sorted signature and E a set of Σ-equations.

The E-*subsumption* preorder $\sqsupseteq_E$ (or just $\sqsupseteq$ if E is understood) holds between $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})$, denoted $t \sqsupseteq_E t'$ (meaning that t is *more general* than t' modulo E), if there is a substitution σ such that $t\sigma =_E t'$; such a substitution σ is said to be an E-*match* from t' to t. The E-renaming equivalence $t \approx_E t'$, holds if there is a variable renaming θ such that $t\theta =_E t'$. We write $t \sqsupset_E t'$ if $t \sqsupseteq_E t'$ and $t \not\approx_E t'$. Relations $\approx_E$ and $\sqsupseteq_E$ are extended to substitutions in a similar way. For substitutions σ, ρ and a set of variables V we define $\sigma|_V =_E \rho|_V$ if $x\sigma =_E x\rho$ for all $x \in V$; $\sigma|_V \sqsupseteq_E \rho|_V$ if there is a substitution η such that $(\sigma \circ \eta)|_V =_E \rho|_V$; and $\sigma|_V \approx_E \rho|_V$ if there is a renaming η such that $(\sigma \circ \eta)|_V =_E \rho|_V$.

An E-*unifier* for a Σ-equation $t = t'$ is a substitution σ such that

$t\sigma =_E t'\sigma$. For $\mathcal{V}ar(t) \cup \mathcal{V}ar(t') \subseteq W$, a set of substitutions $CSU_E^W(t = t')$ is said to be a *complete* set of unifiers for the equality $t = t'$ modulo E away from W iff: (i) each $\sigma \in CSU_E^W(t = t')$ is an E-unifier of $t = t'$; (ii) for any E-unifier ρ of $t = t'$ there is a $\sigma \in CSU_E^W(t = t')$ such that $\sigma|_W \sqsupseteq_E \rho|_W$; (iii) for all $\sigma \in CSU_E^W(t = t')$, $Dom(\sigma) \subseteq (\mathcal{V}ar(t) \cup \mathcal{V}ar(t'))$ and $Ran(\sigma) \cap W = \varnothing$. If the set of variables W is irrelevant or is understood from the context, we write $CSU_E(t = t')$ instead of $CSU_E^W(t = t')$. An E-unification algorithm is *complete* if for any equation $t = t'$ it generates a complete set of E-unifiers. We say that $CSU_E(t = t')$ is *the set of most general unifiers* if each unifier $\tau \in CSU_E(t = t')$ is minimal among all unifiers of $t = t'$ w.r.t. $\sqsupseteq_E$. A unification algorithm is said to be *finitary* and complete if it always terminates after generating a finite and complete set of solutions.

Example 2.1 For example, consider an infix symbol $\_*\_ : \mathsf{Msg} \times \mathsf{Msg} \to \mathsf{Msg}$ satisfying the following associativity and commutativity (AC) equational properties (where X, Y, Z are variables of sort Msg):

$$X * Y = Y * X \qquad X * (Y * Z) = (X * Y) * Z$$

A complete set of most general AC-unifiers of the two terms $t = X * Y$ and $s = U * V$ (where X, Y, U, V are variables of sort Msg) is

$$\begin{array}{lllll}
\sigma_1 = \{ & X \mapsto X', & Y \mapsto Y', & U \mapsto X', & V \mapsto Y' \} \\
\sigma_2 = \{ & X \mapsto X', & Y \mapsto Y', & U \mapsto Y', & V \mapsto X' \} \\
\sigma_3 = \{ & X \mapsto X', & Y \mapsto Y' * Y'', & U \mapsto X' * Y'', & V \mapsto Y' \} \\
\sigma_4 = \{ & X \mapsto X', & Y \mapsto Y' * Y'', & U \mapsto Y'', & V \mapsto X' * Y' \} \\
\sigma_5 = \{ & X \mapsto X' * X'', & Y \mapsto Y', & U \mapsto X'', & V \mapsto X' * Y' \} \\
\sigma_6 = \{ & X \mapsto X' * X'', & Y \mapsto Y', & U \mapsto X'' * Y', & V \mapsto X' \} \\
\sigma_7 = \{ & X \mapsto X' * X'', & Y \mapsto Y' * Y'', & U \mapsto X'' * Y'', & V \mapsto X' * Y' \}
\end{array}$$

■

Example 2.2 Consider now the exclusive-or symbol $\_\oplus\_ : \mathsf{Msg} \times \mathsf{Msg} \to \mathsf{Msg}$ and the constant $0 : \mathsf{Msg}$ satisfying the following exclusive-or properties (where X, Y, Z are variables of sort Msg):

$$\begin{array}{ll}
X \oplus Y = Y \oplus X & X \oplus X = 0 \\
X \oplus (Y \oplus Z) = (X \oplus Y) \oplus Z & X \oplus 0 = X
\end{array}$$

A complete set of most general exclusive-or unifiers of the two terms $t = X \oplus Y$ and $s = U \oplus V$ (where X, Y, U, V are variables of sort Msg) is the unique unifier $\theta_1 = \{X \mapsto Y' \oplus U' \oplus V', Y \mapsto Y', U \mapsto U', V \mapsto V'\}$. ■

Term Rewriting

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 $p \in Pos_\Sigma(t)$, $l \to r \in R$, $t|_p = l\sigma$, and $t' = t[r\sigma]_p$ 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. $t =_E u_1$, $u_1 \to_R u_2$, and $u_2 =_E s$. 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 $t \to_{R/E} t'$. The transitive (resp. transitive and reflexive) closure of $\to_{R/E}$ is denoted $\to^+_{R/E}$ (resp. $\to^*_{R/E}$). A term t is called $\to_{R/E}$-irreducible (or just R/E-irreducible) if there is no term t' such that $t \to_{R/E} t'$.

For a rewrite rule $l \to r$, we say that it is *sort-decreasing* if for each substitution σ, we have $r\sigma \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ implies $l\sigma \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$. A rewrite theory (Σ, E, R) is sort-decreasing if all rules in R are. For a Σ-equation $t = t'$, we say that it is *regular* if $\mathcal{V}ar(t) = \mathcal{V}ar(t')$, and it is *sort-preserving* if for each substitution σ, we have $t\sigma \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ implies $t'\sigma \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$ and vice versa.

For substitutions σ, ρ and a set of variables V we define $\sigma|_V \to_{R/E} \rho|_V$ if there is $x \in V$ such that $x\sigma \to_{R/E} x\rho$ and for all other $y \in V$ we have $y\sigma =_E y\rho$. A substitution σ is called *R/E-normalized* (or normalized) if $x\sigma$ is R/E-irreducible for all $x \in V$.

The relation $\to_{R/E}$ is called *terminating* if there is no infinite sequence $t_1 \to_{R/E} t_2 \to_{R/E} \cdots t_n \to_{R/E} t_{n+1} \cdots$. Further, the relation $\to_{R/E}$ is *confluent* if whenever $t \to^*_{R/E} t'$ and $t \to^*_{R/E} t''$, there exists a term t''' such that $t' \to^*_{R/E} t'''$ and $t'' \to^*_{R/E} t'''$. An order-sorted rewrite theory (Σ, E, R) is confluent (resp. terminating) if the relation $\to_{R/E}$ is confluent (resp. terminating). In a confluent, terminating, sort-decreasing, order-

sorted rewrite theory, for each term $t \in \mathcal{T}_\Sigma(\mathcal{X})$, there is a unique (up to E-equivalence) R/E-irreducible term t' obtained from t by rewriting to canonical form, which is denoted by $t \to^!_{R/E} t'$, or $t{\downarrow_{R/E}}$ when t' is not relevant.

The relation $\to_{R/E}$-reducibility is undecidable in general since E-congruence classes can be arbitrarily large. Therefore, R/E-rewriting is usually implemented [Jouannaud and Kirchner, 1986] by R, E-rewriting, thank to the notion of coherence. A relation $\to_{R,E}$ on $\mathcal{T}_\Sigma(\mathcal{X})$ is defined as: $t \to_{p,R,E} t'$ (or just $t \to_{R,E} t'$) iff there is a non-variable position $p \in Pos_\Sigma(t)$, a rule $l \to r$ in R, and a substitution σ such that $t|_p =_E l\sigma$ and $t' = t[r\sigma]_p$. Note that, assuming E-matching is decidable, $\to_{R,E}$ is decidable. Notions such as confluence, termination, irreducible terms, and normalized substitution, are defined in a straightforward manner for $\to_{R,E}$ [Jouannaud and Kirchner, 1986]. Note that since R is sort-decreasing, confluent, and terminating, i.e., the relation $\to_{R/E}$ is confluent and terminating, and since $\to_{R,E} \subseteq \to_{R/E}$, the relation $\to^!_{R,E}$ is decidable.

Variants

Let (Σ, E) be an order-sorted equational theory, we call (Σ, B, E_0) a *decomposition* of (Σ, E) if $E = E_0 \cup B$ and (Σ, B, E_0) is an order-sorted rewrite theory satisfying the following properties:

1. B is regular and sort-preserving; furthermore, for each equation $t = t'$ in B, all variables in $\mathcal{V}ar(t)$ have a top sort.

2. B has a finitary and complete unification algorithm.

3. $\to_{E_0,B}$ is sort-decreasing, confluent, and terminating.

4. $\to_{E_0,B}$ is locally B-*coherent* [Jouannaud and Kirchner, 1986], i.e., $\forall t_1, t_2, t_3$ we have $t_1 \to_{E_0,B} t_2$ and $t_1 =_B t_3$ implies $\exists t_4, t_5$ such that $t_2 \to^*_{E_0,B} t_4$, $t_3 \to^+_{E_0,B} t_5$, and $t_4 =_B t_5$.

Given a decomposition (Σ, B, E_0) of an equational theory, (t', θ) is an E_0, B-*variant* [Escobar et al., 2012b] (or just a variant) of term t if $t\theta{\downarrow_{E_0,B}} =_B t'$ and $\theta{\downarrow_{E_0,B}} =_B \theta$. A *complete set of* E_0, B-*variants* [Escobar et al., 2012b] (up to renaming) of a term t is a subset, denoted by $[\![t]\!]^\star_{E_0,B}$,

of the set of all E_0, B-variants of t such that, for each E_0, B-variant (t', σ) of t, there is an E_0, B-variant $(t'', \theta) \in [\![t]\!]^{\star}_{E_0,B}$ such that $(t'', \theta) \sqsupseteq_{E_0,B} (t', \sigma)$, i.e., there is a substitution ρ such that $t' =_B t''\rho$ and $\sigma|_{\mathcal{V}ar(t)} =_B (\theta \circ \rho)|_{\mathcal{V}ar(t)}$.

Given two variants $(t_1, \theta_1), (t_2, \theta_2) \in [\![t]\!]^{\star}_{E_0,B}$, we write $(t_2, \theta_2) \sqsupseteq_{E_0,B} (t_1, \theta_1)$, meaning (t_2, θ_2) is more general than (t_1, θ_1), iff there is substitution ρ such that $t_1 =_B t_2\rho$ and $(\theta_1{\downarrow_{E_0,B}})|_{\mathcal{V}ar(t)} =_B (\theta_2 \circ \rho)|_{\mathcal{V}ar(t)}$. We write $(t_2, \theta_2) \sqsupset_{E_0,B} (t_1, \theta_1)$ iff $(t_2, \theta_2) \sqsupseteq_{E_0,B} (t_1, \theta_1)$ and for every substitution ρ such that $t_1 =_B t_2\rho$ and $(\theta_1{\downarrow_{E_0,B}})|_{\mathcal{V}ar(t)} =_B (\theta_2 \circ \rho)|_{\mathcal{V}ar(t)}$, ρ is not a renaming. Given a decomposition (Σ, E_0, B) of an equational theory and t be a Σ-term the set of *most general variants* of t, denoted $[\![t]\!]_{E_0,B}$, is a subset $[\![t]\!]_{E_0,B} \subseteq [\![t]\!]^{\star}_{E_0,B}$ such that: (i) $[\![t]\!]_{E_0,B} \sqsupseteq_{E_0,B} [\![t]\!]^{\star}_{E_0,B}$, and (ii) for each $(t_1, \theta_1) \in [\![t]\!]_{E_0,B}$, there is no $(t_2, \theta_2) \in [\![t]\!]_{E_0,B} \backslash \{(t_1, \theta_1)\} s.t. (t_2, \theta_2) \sqsupseteq (t_1, \theta_1)$. That is, for any term t $[\![t]\!]_{E_0,B}$ characterizes the set of *maximal elements* of the preorder $([\![t]\!]^{\star}_{E_0,B}, \sqsupseteq_{E_0,B})$. Note that even though the set of E_0, B-variants of a term t may be infinite, the set of *most general variants* may be finite. The variant preorder $\sqsupseteq_{E_0,B}$ takes into account not only the instantiation relation between the two substitutions θ_1 and θ_2 and the two normal forms t_1 and t_2 of a term t, but also whether θ_2 is already an E_0, B-normalized substitution, since, for a substitution θ, the less E_0, B rewrite steps, the better.

A decomposition (Σ, E_0, B) has the *finite variant (FV) property* [Escobar et al., 2012b] (also called a *finite variant decomposition*) iff for each Σ-term t, a complete set $[\![t]\!]^{\star}_{E_0,B}$ of its most general variants is finite.

Example 2.3 Let us consider the following equational theory (Σ, B, E_0) for the exclusive-or theory, where E_0 consists of the rules shown below, [1] and B contains the associativity and commutativity (AC) axioms for $\oplus$:

$$X \oplus 0 \to X \qquad X \oplus X \to 0 \qquad X \oplus X \oplus Y \to Y$$

For term $t = M \oplus M$, $(0, id)$ is the only variant. For term $s = X \oplus Y$,

[1] Note that the first two rules are not locally AC-coherent, but adding the third rule (with variable Y) is sufficient to recover that property (see [Viry, 2002; Durán and Meseguer, 2010]).

the set of its most general variants is

$$\{\ (X \oplus Y, id),\\ (Z, \{X \mapsto 0, Y \mapsto Z\}), \quad (Z, \{X \mapsto Z, Y \mapsto 0\}),\\ (Z, \{X \mapsto Z \oplus U, Y \mapsto U\}), \quad (Z, \{X \mapsto U, Y \mapsto Z \oplus U\}),\\ (0, \{X \mapsto U, Y \mapsto U\}), \quad (Z_1 \oplus Z_2, \{X \mapsto U \oplus Z_1, Y \mapsto U \oplus Z_2\})\}$$

since any possible variant of s is an instance of one of the terms according to the substitution. ■

Variants are connected with equational unification as follows. Let $\mathcal{R} = (\Sigma, B, E_0)$ be a finite variant decomposition of an equational theory (Σ, E), and t_1, t_2 be two Σ-terms. Then, intuitively ρ is an E-unifier of t_1 and t_2 iff (t'_1, ρ_1) and (t'_2, ρ_2) are variants of t_1, t_2, respectively, and there exists a substitution σ such that $t'_1\sigma =_B t'_2\sigma$, and $\rho =_B \rho_1 \circ \sigma =_B \rho_2 \circ \sigma$ (see [Escobar et al., 2012b] for more details).

Transition Systems

A transition system is written $\mathcal{A} = (A, \rightarrow)$, where A is a set of states, and $\rightarrow$ is a transition relation between states, i.e., $\rightarrow \subseteq A \times A$. A rewrite theory $\mathcal{R} = (\Sigma, E, R)$ specifies a transition system $\mathcal{T}_\mathcal{R}$ whose states are elements of the initial algebra $\mathcal{T}_{\Sigma/E}$, and whose transitions are specified by the set of rewrite rules R. Given two transition systems $\mathcal{A} = (A, \rightarrow_\mathcal{A})$ and $\mathcal{B} = (B, \rightarrow_\mathcal{B})$, a *simulation* from $\mathcal{A}$ to $\mathcal{B}$, written $\mathcal{A}\, H\, \mathcal{B}$, is a relation $H \subseteq A \times B$ such that $a\, H\, b$ and $a \rightarrow_\mathcal{A} a'$ implies that there exists $b' \in B$ such that $a'\, H\, b'$ and $b \rightarrow_\mathcal{B} b'$. A simulation H from $(A, \rightarrow_\mathcal{A})$ to $(B, \rightarrow_\mathcal{B})$ is a *bisimulation* if H^{-1} is a simulation from $(B, \rightarrow_\mathcal{B})$ to $(A, \rightarrow_\mathcal{A})$.

2.2 Symbolic Reachability Analysis by Narrowing

In this section we recall basic facts about narrowing modulo equations of Meseguer and Thati [2007] using topmost rewriting as a tool-independent semantic framework for symbolic reachability analysis of protocols under algebraic properties. We first define reachability goals.

Definition 2.4 (Reachability goal) *Given an order-sorted rewrite theory* (Σ, E, R)*, a* reachability goal *is defined as a pair* $t \stackrel{?}{\rightarrow}^*_{R/E} t'$*, where* $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$*. It is abbreviated as* $t \stackrel{?}{\rightarrow}^* t'$ *when the theory is clear from the context;* t *is the* source *of the goal and* t' *is the* target*. A substitution* σ *is a* R/E-solution *of the reachability goal (or just a solution for short) iff there is a sequence* $t\sigma \rightarrow_{R/E} u_1\sigma \rightarrow_{R/E} \cdots \rightarrow_{R/E} (u_{k-1})\sigma \rightarrow_{R/E} t'\sigma$*.*

A set Γ *of substitutions is said to be a* complete set of solutions *of* $t \stackrel{?}{\rightarrow}^*_{R/E} t'$ *iff (i) every substitution* $\sigma \in \Gamma$ *is a solution of* $t \stackrel{?}{\rightarrow}^*_{R/E} t'$*, and (ii) for any solution* ρ *of* $t \stackrel{?}{\rightarrow}^*_{R/E} t'$*, there is a substitution* $\sigma \in \Gamma$ *more general than* ρ *modulo* E*, i.e.,* $\sigma|_{\mathcal{V}ar(t)\cup\mathcal{V}ar(t')} \sqsupseteq_E \rho|_{\mathcal{V}ar(t)\cup\mathcal{V}ar(t')}$*.*

If in a goal $t \stackrel{?}{\rightarrow}^*_{R/E} t'$, terms t and t' are ground, then goal solving becomes a standard rewriting reachability problem. However, since we allow terms t, t' with variables, we need a mechanism more general than standard rewriting to find solutions of reachability goals.

Narrowing generalizes term rewriting by allowing free variables in terms (as in logic programming) and by performing unification instead of matching in order to (non–deterministically) reduce a term. Intuitively, the difference between a rewriting step and a narrowing step is that in both cases we use a rewrite rule $l \rightarrow r$ to rewrite t at a position p in t, but narrowing finds values for the variables in the chosen subject term $t|_p$ before actually performing the rewriting step. The narrowing relation $\leadsto_R$ on $\mathcal{T}_\Sigma(\mathcal{X})$ is $t \stackrel{p}{\leadsto}_{\sigma,R} t'$ (or $\leadsto_{\sigma,R}$, $\leadsto_R$) if $p \in Pos_\Sigma(t)$, $l \rightarrow r \in R$, $\sigma \in CSU_\emptyset(t|_p = l)$, and $t' = (t[r]_p)\sigma$. Given a *topmost* rewrite theory $\mathcal{R} = (\Sigma, E, R)$, if the left-hand side of the rules R are non-variable terms and a finitary E-unification procedure is available, each rule $l \rightarrow r$ specifies a *topmost narrowing step* $t \leadsto_{\sigma,R,E} t'$ (or $t \leadsto_{R,E} t', t \leadsto_R t'$) iff there exists an E-unifier $\sigma \in CSU_E(t = l)$ such that $t' = r\sigma$.

The use of topmost rewrite theories is entirely natural for communication protocols, since all state transitions can be viewed as changes of the global distributed state. It also provides several advantages (see [Thati and Meseguer, 2007]): (i) as pointed out above the relation $\rightarrow_{R,E}$ achieves the same effect as the relation $\rightarrow_{R/E}$, and (ii) we obtain a completeness result between narrowing ($\leadsto_{R,E}$) and rewriting ($\rightarrow_{R/E}$).

Theorem 2.5 (Topmost Completeness) [Thati and Meseguer, 2007] *Let* $\mathcal{R} = (\Sigma, E, R)$ *be a topmost rewrite theory,* $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})$*, and let* σ

be a substitution such that $t\sigma \rightarrow^*_{R,E} t'$. *Then, there are substitutions* θ, τ *and a term* t'' *such that* $t \rightsquigarrow^*_{\theta,R,E} t''$, $t\sigma =_E (t\theta)\tau$, *and* $t' =_E t''\tau$.

2.3 Maude

Maude [Clavel et al., 2007] is a very efficient implementation of *Rewriting Logic* [Meseguer, 1992], publicly available at [2]. As it is presented in this section, Maude is a high-performance programming language that uses rewriting rules, similarly to the so-called functional languages like Haskell, ML, Scheme, or Lisp. In the following, we briefly present some of the features of this language that have been used in our work.

A Maude program is made up of different *modules*. Each module can include:

- *sort* (or type) declarations;
- *variable* declarations;
- *operator* declarations;
- *rules* and/or *equations* describing the behavior of the system operators, i.e., the *functions*.

Maude mainly distinguishes two kinds of modules depending on the constructions they define and on their expected behavior. Functional modules do not contain rules and the behavior of their equations is expected to be confluent and terminating. On the contrary, system modules can contain both equations and rules and, though the behavior of their equations is also expected to be confluent and terminating, the behavior of its rules may be non-confluent and non-terminating. A functional module is limited by the reserved keywords `fmod` and `endfm`, whereas a system module is defined in between `mod` and `endm`.

Sorts. A sort declaration is as follows:

```
sort T .
```

[2] At `http://maude.cs.uiuc.edu`

where `T` is the identifier of the newly introduced sort `T`. Maude *identifiers* are sequences of ASCII characters without white spaces, nor the special characters '{', '}', '(', ')', '[', and ']', unless they are escaped with the back-quote character '`'. If we want to introduce many sorts `T1 T2 ...Tn` at the same time, we write:

```
sorts T1 T2 ... Tn .
```

After having declared the sorts, we define operators on them.

Operators. Operators are declared as follows:

```
op C : T1 T2 ... Tn -> T .
```

where `T1 T2 ... Tn` are the sorts of the arguments for operator `C`, and `T` is the resulting sort for the operator. We can also declare at the same time many operators `C1 C2 ... Cn` with the same signature (i.e., sort of arguments and resulting sort):

```
ops C1 C2 ... Cn : T1 T2 ... Tn -> T .
```

Operators can represent two kinds of objects: *constructors* and *defined* symbols. Constructors constitute the *ground terms* or *data* associated to a sort, whereas defined symbols represent functions whose behavior will be specified by means of equations or rules. The rewriting engine of Maude does not distinguish between constructors or defined symbols, so there is no real syntactic difference between them. However, for documentation (and debugging) purposes, operators that are used as constructors can be labeled with the attribute `ctor`.

Operator attributes. Operator attributes are labels that can be associated to an operator in order to provide additional information (either syntactic or semantic) about the operator. All such attributes are declared within a single pair of enclosing square brackets "`[`" and "`]`":

```
ops C1 C2 ... Cn : T1 T2 ... Tm -> T  [A1 ... Ak] .
```

where the `Ai` are attribute identifiers. The set of operator attributes includes among others: `ctor`, `assoc`, `comm`, `id`, `ditto`, etc., that are described below.

Mix-fix notation. Another interesting feature of operators in Maude is *mix-fix* notation. Every operator defined as above is declared in *prefix* notation, that is, its arguments are separated by commas, and enclosed in parenthesis, following the operator symbol, as in:

```
C(t1, t2, ... , tn)
```

where `C` is an operator symbol, and `t1, t2,..., tn` are, respectively, terms of sorts `T1, T2,..., Tn`. Nevertheless, Maude provides a powerful and tunable syntax analyzer that allows us to declare operators composed of different identifiers separated by its arguments. Arguments can be set in any position, in any order, and even separated by white spaces. Mix-fix operators are identified by the sequence of its component identifiers, with characters '_' inserted in the place each argument is expected to be, as in:

```
op if_then_else_fi : Bool Exp Exp -> Exp .
op __ : Element List -> List .
```

The first line above defines an if-then-else operator, while the second one defines `Lists` of juxtaposed (i.e., separated by white spaces) `Element`s. A term built with the `if_then_else_fi` operator looks like:

```
if b1 then e1 else e2 fi
```

where the tokens `if`, `then`, `else` and `fi` represent the mixfix operator, `b1` represents a term of sort `Bool`, and finally `e1` and `e2` represent terms of sort `Exp`. A term built with the operator __ looks like:

```
e1 e2
```

where `e1` is a term of sort `Element`, `e2` is a term of sort `List`, and the space separating them represents the juxtaposition operator __ .

Sorts orders. Sorts can be organized into *hierarchies* with `subsort` declarations. In the following declaration:

```
subsort T1 < T2 .
```

we state that each element in `T1` is also in `T2`. For example, we can define natural numbers by considering their classification as positives or as the zero number in this way:

```
sorts Nat Zero NonZeroNat .
subsort Zero < Nat .
subsort NonZeroNat < Nat .
op 0 : -> Zero [ctor] .
op s : Nat -> NonZeroNat [ctor].
```

Maude also provides operator *overloading*. For example, if we add:

```
sort Binary .
op 0 : -> Binary [ctor] .
op 1 : -> Binary [ctor] .
```

to the previous declarations, the operator 0 is used to construct values both for the `Nat` and for the `Binary` sorts.

Structural Axioms. The language allows the specification of structural axioms over operators, i.e., certain algebraic properties like *Associativity*, *Commutativity* and *Identity* element that operators may satisfy. In the following, we write ACU to refer to the three previous algebraic properties. Structural axioms serve to perform the computation on equivalence classes of expressions, instead of on simple expressions. In order to carry out computations on equivalence classes, Maude chooses an irreducible representative of each class and uses it for the computation. Thanks to the structural information given as operator attributes, Maude can also choose specific data structures for an efficient low-level representation of expressions.

For example, let us define a list of natural numbers separated by colons:

```
sorts NatList EmptyNatList NonEmptyNatList .
subsort EmptyNatList < NatList .
subsort Nat < NonEmptyNatList < NatList  .
op nil : -> EmptyNatList [ctor] .
op _:_  : NatList NatList -> NonEmptyNatList [assoc] .
```

The operator "_:_" is declared as associative by means of its attribute `assoc`. Associativity means that the value of an expression is not dependent on the subexpression grouping considered, that is, the places where the parenthesis are inserted. Thus, if "_:_" is associative Maude considers the following expressions as equivalent:

```
s(0) : s(s(0)) : nil
(s(0) : s(s(0))) : nil
s(0) : (s(s(0)) : nil)
```

As another example, let us define an associative list with `nil` as its identity element:

```
sort NatList .
subsort Nat < NatList .
op nil : -> NatList [ctor] .
op _;_ : NatList NatList -> NatList [assoc id: nil] .
```

The operator "_;_" is declared as having `nil` as its *identity element* by means of its attribute `id: nil`. Having an identity element e means that the value of an expression is not dependent on the presence of e's as subexpressions, that is, it is possible to insert e's without changing the meaning of the expression. Thus, in our example Maude considers the following expressions (and an infinite number of similar ones) as equivalent:

```
s(0) ; s(s(0))
nil ; s(0) ; s(s(0))
s(0) ; nil ; s(s(0))
s(0) ; s(s(0)) ; nil
nil ; s(0) ; nil ; s(s(0)) : nil
```

⋮

For that reason, Maude omits `nil` in the irreducible representative, unless it appears alone as an expression. Now, let us introduce how we define a multi-set, that is, an associative and commutative list with `nil` as its identity element:

```
sort NatMultiSet .
subsort Nat < NatMultiSet .
op nil : -> NatMultiSet .
op _:_ : NatMultiSet NatMultiSet -> NatMultiSet
         [assoc comm id: nil] .
```

In this example, the operator "_:_" is declared to be *commutative* by means of the attribute `comm`. Commutativity means that the value of an expression is not dependent on the order of its subexpressions, that is, it is possible to change the order of subexpressions without changing the meaning of the expression. Thus, if "_:_" is a commutative and associative operator, Maude considers the following expressions equivalent:

```
s(0) : s(s(0)) : s(s(0))
s(s(0)) : s(0) : s(s(0))
s(s(0)) : s(s(0)) : s(0)
```

The structural properties are efficiently built in Maude. Additional structural properties can be defined by means of equations, as we discuss below.

Rules and equations. In Maude, *rules* or *equations* characterize the behavior of the *defined symbols*. Both language constructions have a similar structure:

```
rl l => r .
eq l = r .
```

where `l` and `r` are `terms`, i.e., expressions recursively built by nesting correctly typed operators and variables. `l` is called the left-hand side of a rule or equation, whereas `r` is its right-hand side. Variables can be declared when they are used in an expression by using the structure *name:sort*, or also in a general variable declaration:

```
var N1 N2 ... Nm : S .
```

where `N1, N2, ..., Nm` are variable names, and `S` is a sort. In Maude-NPA protocol specifications equations are labelled with the `variant` attribute, to denote that these equations must be used for variant-based unification.

The `search` command. The `search` command allows one to explore (following a breadth-first strategy) the reachable state space in different ways. Throughout this thesis, we will use `search` commands of either one of the following schemes:

```
search in ⟨ModId⟩ :  ⟨Term-1⟩ =>*  ⟨Term-2⟩.
search in ⟨ModId⟩ :  ⟨Term-1⟩ =>*  ⟨Term-2⟩
   such that ⟨Condition⟩.
```

where:

- `⟨ModId⟩` is the module where the search takes place. This parameter can be ommited.
- `⟨Term-1⟩` is the starting term.
- `⟨Term-2⟩` is the pattern that has to be reached.
- `=>*` indicates that the rewriting proof from `⟨Term-1⟩` to `⟨Term-2⟩` can consist of zero or more steps.
- `⟨Condition⟩` states an optional property that has to be satisfied by the reached state.

Example 2.6 Finally, let us show the complete specification of a Maude program as an example. In the following, we consider an alternating-bit-protocol, a simple but effective protocol that avoids duplication or loss of messages. In this protocol there are two principals, Alice and Bob, and a communication channel connecting both of them. Each exchange message consists in a protocol bit, i.e., either 0 or 1.

This protocol works as follows. When Alice wants to send a message to Bob, she repeats it indefinetely through the channel, using the

same protocol bit until she receives an acknowledgement from B with the same protocol bit she sent. When Alice receives the acknowledgement, she sends the next message with the opposite protocol bit, that is, if the previous sent message was 0, the next message is 1, and viceversa. When Bob receives a message, he accepts it and sends to Alice the acknowledgement, sending to her the same protocol bit that he received. If he receives several messages with the same protocol bit, he sends the same acknoledgement with the same bit protocol.

In order to represent the loss of messages, it is assumed that the channel chooses in a non-deterministic way whether it copies the received message in the channels output, or discards it.

This program is specified in Maude as follows:

```
mod Alternating-bit-protocol is
 sorts Bit Msg Channel State .
 subsort Msg < Channel .

 ops 0 1 : -> Bit .
 ops _>> <<_ : Bit -> Msg .
 op lost : -> Channel .
 op A{_}_B{_} : Bit Channel Bit -> State .

 var M : Msg . vars X Y Z : Bit .

 --- equations to compute inverse of a bit
 op not : Bit -> Bit .
 eq not(0) = 1 .
 eq not(1) = 0 .

 --- The message in the channel is lost
 rl [loss] : A{X} M B{Y} => A{X} lost B{Y} .

 --- Bob receives the message sent by Alice
 rl [B-receives] : A{X} Z >> B{Y} => A{X} Z >> B{Z} .
 --- Bob sends the acknowledgement to Alice
 rl [B-repeats] : A{X} M B{Y} => A{X} << Y B{Y} .

 --- Alice changes the bit
 rl [A-changesBit] : A{X} << X B{X} => A{not(X)} << X B{X} .
```

```
  --- Alice sends the bit
  rl [A-sends] : A{X} << Z B{Z} => A{X} X >>   B{Z} .
endm
```

This protocol must satisfy that if Alice sends to Bob a protocol bit, e.g. 0, and Bob acknowledges, the received protocol bit must be the same that she sent before. This can be verified in Maude by executing the following `search` command:

```
search A{0} << 1  B{1} =>* A{0} << 0 B{0} .
```

which produces the output shown below:

```
==========================================
search in Alternating-bit-protocol :
A{1}<< 0 B{0} =>* A{0}<< 1 B{1} .

Solution 1 (state 6)
states: 7  rewrites: 12 in 0ms cpu (0ms real)
(200000 rewrites/second)
empty substitution

No more solutions.
states: 12  rewrites: 26 in 0ms cpu (0ms real)
(156626 rewrites/second)
```

■

Chapter 3

Maude-NPA

In this chapter we present Maude-NPA, the protocol analyzer tool in which the techniques explained in the thesis have been implemented. We refer the reader to [Escobar et al., 2009a, 2012a] for futher detailed information on Maude-NPA.

First, Section 3.1 provides an overview of the Maue-NPA tool. Section 3.2 explains Maude-NPA's use of the strand space model to specify protocols and states. In Section 3.3 we show how backwards analysis works in Maude-NPA, and in Section 3.4 we specify the rewrite rules governing the backwards semantics. Section 3.5 explains in detail the most general class of equational theories for which the Maude-NPA can currently support variant-based unification, with the important requirement of the number of unifier solutions being finite. These are called *admissible theories*. In Section 3.6 we briefly recall how to specify a protocol in Maude-NPA, and, finally, in Section 3.7 we describe the tool's commands for attack search.

3.1 Overview

The Maude-NPA is a tool and inference system for reasoning about the security of cryptographic protocols in which the cryptosystems satisfy different equational properties. The tool handles searches in the unbounded session model, and thus can be used to provide proofs of security as well as to search for attacks. It is the next generation of the NRL Protocol Analyzer [Meadows, 1996a], a tool that supported limited equational

reasoning and was successfully applied to the analysis of many different protocols.

Maude-NPA is a model-checker for cryptographic protocol analysis that both allows for the incorporation of different equational theories and operates in the unbounded session model without the use of abstraction. This means that the analysis is *exact*. That is, (i) if an attack exists using the specified algebraic properties, it will be found; (ii) no false attacks will be reported; and (iii) if the tool terminates without finding an attack, this provides a *formal proof* that the protocol is secure for that attack modulo the specified properties. However, it is always possible that the tool will not terminate; although, as explained in Chapter 4, a number of heuristics are included to drastically reduce the search space and make nontermination less likely. In order to have a steady, incremental approximation of the analysis, the user is also given the option of restricting the number of steps executed by Maude-NPA.

Maude-NPA is a backwards search tool, i.e., it searches backwards from a final insecure state to determine whether or not it is reachable from an initial state. This backwards search is *symbolic*, i.e., it does not start with a *concrete* attack state, but uses instead a symbolic *attack pattern*, i.e., a term with logical variables describing a general attack situation. The backwards search is then performed by *backwards narrowing*. Each backwards narrowing step denotes a state transition, such as a principal sending or receiving a message or the intruder manipulating a message, all in a backwards sense. Each backwards narrowing step takes a symbolic state (i.e., a term with logical variables) and returns a previous symbolic state in the protocol (again a term with logical variables). In performing a backwards narrowing step, the variables of the input term are appropriately instantiated in order to apply the concrete state transition, and the new previous state may contain new variables that are differentiated from any previously used variable to avoid confusion. To appropriately instantiate the input term, narrowing uses *equational unification* (see Section 2.1). As it is well-known from logic programming and automated deduction (see e.g., [Baader and Snyder, 2001]), unification is the process of solving equations $t = t'$. Standard unification solves these equations in a term algebra. Instead, *equational unification* (w.r.t. an equational theory E) solves an equation $t = t'$ *modulo* the equational theory E. In the Maude-NPA, the equational theory E used

depends on the protocol, and corresponds to the algebraic properties of the cryptographic functions (e.g. cancellation of encryption and decryption, Diffie-Hellman exponentiation, or exclusive-or).

3.2 Maude-NPA's Strand Space Model

In this section we give an overview of Maude-NPA's use of the strand space model to specify protocols and states.

Given a protocol $\mathcal{P}$, states are modeled as elements of an initial algebra $\mathcal{T}_{\Sigma_\mathcal{P}/E_\mathcal{P}}$, 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 $[t]_{E_\mathcal{P}} \in \mathcal{T}_{\Sigma_\mathcal{P}/E_\mathcal{P}}$ with t a ground $\Sigma_\mathcal{P}$-term. However, we explore *symbolic state patterns* $[t(x_1, \ldots, x_n)]_{E_\mathcal{P}} \in \mathcal{T}_{\Sigma_\mathcal{P}/E_\mathcal{P}}(\mathcal{X})$ (see Chapter 2) on the free $(\Sigma_\mathcal{P}, E_\mathcal{P})$-algebra over a set of sorted variables $\mathcal{X}$. There are three relevant sorts in Maude-NPA, namely State, Msg, and Fresh, which are described below. Also, due to the symbolic representation, we use uppercase names for variables (we omit the sort of a variable when it is easy to deduce from the context) and lowercase names for terms (with or without variables). Indeed, we will make explicit when a term does not contain variables.

In Maude-NPA a *state pattern* in a protocol execution is a term t of sort State (i.e., $t \in \mathcal{T}_{\Sigma_\mathcal{P}/E_\mathcal{P}}(\mathcal{X})_{\mathsf{State}}$) $\{S_1 \,\&\, \cdots \,\&\, S_n \,\&\, \{IK\}\}$ where & is an associative-commutative union[1] operator with identity symbol $\varnothing$. Each element in the set is either a *strand* S_i or the *intruder knowledge* $\{IK\}$ at that state.

The *intruder knowledge* $\{IK\}$ is represented as a set of facts using the comma as an associative-commutative union[2] operator with identity operator *empty*. There are two kinds of intruder facts: positive knowledge facts (the intruder knows m, i.e., $m{\in}\mathcal{I}$), and negative knowledge facts

[1] As described in [Escobar et al., 2009a], & can also be treated as an associative-commutative-idempotent union operator with an identity symbol because the combination of fresh variables and the learn-only-once rule allows for that.

[2] Again, the comma for the intruder's knowledge is described in [Escobar et al., 2009a] as an associative-commutative union operator with an identity symbol but it can be understood as being idempotent, though only for the positive intruder 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.

A *strand* [Fabrega et al., 1999] specifies the sequence of messages sent and received by a principal executing the protocol and is represented as a sequence of messages $[msg_1^-, msg_2^+, msg_3^-, \ldots, msg_{k-1}^-, msg_k^+]$ such that msg_i^- (also written $-msg_i$) represents an input message, msg_i^+ (also written $+msg_i$) represents an output message, and each msg_i is a term of sort Msg (i.e., $msg_i \in \mathcal{T}_{\Sigma_\mathcal{P}/E_\mathcal{P}}(\mathcal{X})_{\mathsf{Msg}}$). For each positive message msg_i^+ in a strand, the variables occurring in message msg_i must appear[3] in previous messages $msg_1, \ldots, msg_{i-1}$, except for variables denoting principal names (they are considered as initial knowledge available to all participants) and variables of sort Fresh. Variables of sort Fresh are unique for each instance of a strand schemata, i.e., if we compare two strands for Alice or a strand for Alice and a strand for Bob, they will have different, unique, fresh variables associated with them.

Strands are used to represent both the actions of honest principals (with a strand specified for each protocol role) and the actions of an intruder (with a strand for each operator an intruder is able to perform on terms). In Maude-NPA, strands evolve over time; the symbol | is used to divide past and future. That is, given a strand $[\ {m_1}^{\pm},\ \ldots,\ {m_i}^{\pm} \mid {m_{i+1}}^{\pm},\ \ldots,\ {m_k}^{\pm}\ ]$, messages $m_1^{\pm}, \ldots, m_i^{\pm}$ are the *past messages*, and messages $m_{i+1}^{\pm}, \ldots, m_k^{\pm}$ are the *future messages* ($m_{i+1}^{\pm}$ is the immediate future message). 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. A strand $[msg_1^{\pm}, \ldots, msg_k^{\pm}]$ is a shorthand for $[nil \mid msg_1^{\pm}, \ldots, msg_k^{\pm}, nil]$. An initial state is a state where the bar is at the beginning for all strands in the state, and 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. A final state is a state where the bar is at the end for all strands in the state and there is no intruder fact of the form $m \notin \mathcal{I}$.

Variables of sort Fresh represent 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 nonces are represented using variables of sort

[3] This restriction is common in protocol analysis using constraint systems and corresponds to *deterministic constraint systems*, see [Chevalier and Rusinowitch, 2008].

Fresh, they will never be merged and no approximation for nonces is necessary. We make the Fresh variables generated by a strand explicit by writing $(r_1, \ldots, r_k : \mathsf{Fresh})\ [msg_1^{\pm}, \ldots, msg_n^{\pm}]$, where $r_1, \ldots, r_k$ are all the variables of sort Fresh generated by $msg_1^{\pm}, \ldots, msg_n^{\pm}$. Each r_i first appears in an output message $m_{j_i}^{+}$ and can later be used in any input and output message of $m_{j_i+1}^{\pm}, \ldots, m_n^{\pm}$. If it does not generate any Fresh variable, we write $:: nil :: [m_1^{\pm}, \ldots, m_n^{\pm}]$.

Example 3.1 Let us consider the well-known Diffie-Hellman protocol, used without authentication. This protocol uses exponentiation to share a secret between two parties, Alice and Bob. The protocol involves an initiator, Alice, and a responder, Bob. We use the common notation $A \rightarrow B : M$ to stand for "A sends message M to B". Encryption of message M using a key K is denoted by $\{M\}_K$. Decryption is done when the principal knows the appropriate key. Concatenation of two messages M_1 and M_2 is denoted by $M_1; M_2$. Raising message M to the power of exponent X is denoted by $(M)^X$. There is a public term denoted by g, which will be the base of our exponentiations. We represent the product of exponents by using the symbol $*$, which is an associative-commutative symbol. Nonces are represented by N_X, denoting a nonce created by principal X. The protocol description is as follows.

1. $A \rightarrow B : A; B; g^{N_A}$
   Alice creates a new nonce N_A and sends her name, Bob's name, and g^{N_A} to Bob.

2. $B \rightarrow A : B; A; g^{N_B}$
   Bob creates a new nonce N_B and sends his name, Alice's name, and g^{N_B} to Alice.

3. $A \rightarrow B : \{secret\}_{g^{N_B N_A}}$
   Alice computes $g^{N_B N_A}$ and encrypts the secret data *secret*. The key $g^{N_B N_A}$ is equal to $g^{N_B * N_A}$ using the algebraic property $X^{YZ} = X^{ZY} = X^{Y*Z}$. Bob computes $g^{N_A N_B}$ and obtains the secret data *secret*.

This protocol is described using strands as follows. Here encryption $\{M\}_K$ is denoted by $e(K, M)$ and exponentiation X^Y is denoted by

$exp(X, Y)$. Nonces and secret data are denoted by terms of the form $n(A, r)$ and *sec(A,r)*, respectively where r is a fresh variable that ensures uniqueness and A is a variable used to identify which principal generated the nonce or the secret data.

$$\begin{aligned}
&:: r, r' ::: [+(A; B; exp(g, n(A, r))),\\
&\qquad -(B; A; X),\\
&\qquad +(e(exp(X, n(A, r)), sec(A, r')))] \;\&\\
&:: r'' :: [-(A; B; Y),\\
&\qquad +(B; A; exp(g, n(B, r''))),\\
&\qquad -(e(exp(Y, n(B, r'')), Sr))]
\end{aligned}$$

Intruder strands are also included for each function. For example, encryption and decryption by the intruder are described by the strands shown below, respectively:

$$[-(K), -(M), +(e(K, M))]$$
$$[-(K), -(M), +(d(K, M))]$$

together with the equational property $d(K, e(K, M)) = M$. The intruder can perform Diffie-Hellman exponentiations via the strand below:

$$:: nil :: [-(Y), -(X), +(exp(X, Y))]$$

together with the equational property $exp(exp(X, Y1), Y2) = exp(X, Y1 * Y2)$, where $*$ is an AC operator denoting the product of exponents. The intruder's capability to perform the product of exponents is denoted by the strand below:

$$:: nil :: [-(NS1), -(NS2), +(NS1 * NS2)]$$

Finally, the intruder can also generate nonces via the strand below:

$$:: r :: [+(n(i, r))]$$

■

3.3 Backwards Reachability Analysis

Since the number of states $\mathcal{T}_{\Sigma_\mathcal{P}/E_\mathcal{P}}$ is in general infinite, rather than exploring concrete protocol states $[t]_{E_\mathcal{P}} \in \mathcal{T}_{\Sigma_\mathcal{P}/E_\mathcal{P}}$ we explore *symbolic state patterns* $[t(x_1, \ldots, x_n)]_{E_\mathcal{P}} \in \mathcal{T}_{\Sigma_\mathcal{P}/E_\mathcal{P}}(X)$ on the free $(\Sigma_\mathcal{P}, E_\mathcal{P})$-algebra over a set of variables X. In this way, a state pattern $[t(x_1, \ldots, x_n)]_{E_\mathcal{P}}$ represents not a single concrete state but a possibly infinite set of such states, namely all the instances of the pattern $[t(x_1, \ldots, x_n)]_{E_\mathcal{P}}$ where the variables $x_1, \ldots, x_n$ have been instantiated by concrete ground terms.

The protocol analysis methodology of Maude-NPA is then based on the idea of *symbolic 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.

Given a protocol $\mathcal{P}$ and the rewrite theory $(\Sigma_\mathcal{P}, E_\mathcal{P}, R_{B\mathcal{P}})$ associated to it, in a backwards execution of the protocol we start from an attack pattern, i.e., a term with variables, containing: (i) some of the strands of the protocol to be executed backwards, (ii) a variable SS denoting a set of additional strands, (iii) some terms the intruder knows at the attack state, i.e., of the form $t{\in}\mathcal{I}$, and (iv) a variable IK denoting a set of additional intruder facts. We then symbolically run the protocol "in reverse" by narrowing modulo the equations $E_\mathcal{P}$. This can be achieved by using a set of rules $R^{-1}_{B\mathcal{P}}$ described in Section 3.4 (where $v \longrightarrow u$ is in $R^{-1}_{B\mathcal{P}}$ iff $u \longrightarrow v$ is in $R_{B\mathcal{P}}$), and performing backwards *narrowing* (see Chapter 2) until either we find an initial state, or cannot perform any other useful backwards narrowing steps. Note that variables SS and IK will be instantiated by backwards narrowing to additional strands and additional intruder facts, respectively, in order to find an initial state. Indeed, these variables SS and IK are not required for the specification of the attack state as explained in Section 3.6.1.

Example 3.2 Given the protocol of Example 3.1, the final state pattern associated to Bob receiving some secret data from a communication with Alice and the intruder learning the secret is as follows (where Y, SR, SS, and IK are variables and we use lowercase a and b to represent the actual names of Alice and Bob instead of variable names A and B):

$$\{:: r'' :: [-(a; b; Y),\\ +(b; a; exp(g, n(b, r''))),\\ -(e(exp(Y, n(b, r'')), SR)) \mid nil] \ \& \ SS \ \& \ \{SR{\in}\mathcal{I}\}\}$$

The strands of the initial state found by the tool correspond to a very general man-in-the-middle attack, with two sessions and variables B', NS and NS'. The principal strands are as follows, where Alice (principal name a) is talking to some principal name B' and Bob (principal name b) believes is talking to Alice:

$$:: r_1, r_2 :: [+(a; B'; exp(g, n(a, r_2))),\\ -(B'; a; exp(g, NS)),\\ +(e(exp(g, NS * n(a, r_2)), sec(a, r_1)))]) \ \&$$

$$:: r_3 :: [-(a; b; exp(g, NS')),\\ +(b; a; exp(g, n(b, r_3))),\\ -(e(exp(g, NS' * n(b, r_3)), sec(a, r_1)))]$$

The instantiated Dolev-Yao intruder strands of the initial state found by the tool are as follows, where variables NS and NS' correspond to sets of nonces generated by the intruder but the tool does not actually try to find instances of those variables:

$$[-(a; B'; exp(g, n(a, r_2))), +(B'; exp(g, n(a, r_2)))] \ \&$$

$$[-(B'; exp(g, n(a, r_2))), +(exp(g, n(a, r_2)))] \ \&$$

$$[-(exp(g, n(a, r_2))), -(NS), +(exp(g, NS * n(a, r_2)))] \ \&$$

$$[-(exp(g, NS * n(a, r_2))), -(e(exp(g, NS * n(a, r_2)), sec(a, r_1))),\\ +(sec(a, r_1))] \ \&$$

$$[-(exp(g, n(b, r_3))), -(NS'), +(exp(g, NS' * n(b, r_3)))] \ \&$$

$$[-(exp(g, NS' * n(b, r_3))), -(sec(a, r_1)),\\ +(e(exp(g, NS' * n(b, r_3)), sec(a, r_1)))] \ \&$$

$$[-(b; a; exp(g, n(b, r_3))), +(a; exp(g, n(b, r_3)))] \ \&$$

$$[-(a; exp(g, n(b, r_3))), +(exp(g, n(b, r_3)))]$$

Note that Maude-NPA does not display the initial knowledge of the intruder, since it corresponds to all the input and output messages appearing in the initial strands above.

Maude-NPA also allows verification of authentication properties by using *never patterns*, i.e., the reachability analysis succeeds when none of the states in the reachability sequence is an instance of the never pattern. Never patterns can share variables with the attack pattern in order to have more specific patterns and the vertical bar is not included in strands of never patterns, since all the combinations of the vertical bar are taken into account. For instance, we can specify the following authentication attack pattern for Diffie-Hellman by including Bob's strand and adding never patterns for Alice's strand

$$\begin{aligned}&\{:: r'' :: [-(a;b;Y), +(b;a;exp(g,n(b,r''))),\\ &\qquad -(e(exp(Y,n(b,r'')),SR)) \mid nil] \ \ \&\ SS\ \&\ \{IK\}\} \wedge\\ &never(:: r, r' :: [+(a;b;exp(g,n(a,r))), -(b;a;X),\\ &\qquad +(e(exp(X,n(a,r)),sec(a,r')))])\end{aligned}$$

The initial state above is also a solution of this attack pattern with never patterns, since Alice was talking to a different participant in a different session. ■

Because (as we shall see in Section 3.4) terms available to the intruder are not always explicitly represented in the intruder knowledge, we assume that never patterns as implemented in Maude-NPA consist only of strands, and do not describe intruder knowledge terms. This is generally the case for authentication patterns. However, if we do wish to specify a never pattern in which the intruder knows a particular message, this can be represented as a set of never patterns, each one containing one of the possible strands having that message as a positive term. Note that explicitly specifying the message as part of the intruder knowledge in the never pattern would *not* rule out all states in which the message is produced, since the intruder knowledge is only guaranteed to contain the messages the intruder uses to get to the (main) final state; it is not guaranteed to contain all the messages produced in the protocol execution.

3.4 Backwards Operational Semantics

In the backwards reachability analysis performed by Maude-NPA sketched in Section 3.3, state changes are described by means of a set $R_{B\mathcal{P}}$ of

rewrite rules, so that the rewrite theory $(\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, R_{B\mathcal{P}})$ characterizes the behavior of protocol $\mathcal{P}$ modulo the equations $E_{\mathcal{P}}$ for backwards execution. In this section we use $R_{B\mathcal{P}}$ to denote the rewrite rules associated to protocol $\mathcal{P}$ for backwards execution. The rules $R_{B\mathcal{P}}$ are defined in two blocks below: (i) *generic rules* (3.1), (3.2), and (3.3), and (ii) *protocol-specific* rules (3.4) generated for each principal and intruder strand in the given protocol.

The following rewrite rules, though written in a forward sense, are used in a backwards sense (by reversing the direction of the arrow) and describe[4] the general state transitions:

$$\{SS \,\&\, [L \mid M^-, L'] \,\&\, \{M{\in}\mathcal{I}, IK\}\} \rightarrow \{SS \,\&\, [L, M^- \mid L'] \,\&\, \{M{\in}\mathcal{I}, IK\}\} \quad (3.1)$$

$$\{SS \,\&\, [L \mid M^+, L'] \,\&\, \{IK\}\} \rightarrow \{SS \,\&\, [L, M^+ \mid L'] \,\&\, \{IK\}\} \quad (3.2)$$

$$\{SS \,\&\, [L \mid M^+, L'] \,\&\, \{M{\notin}\mathcal{I}, IK\}\} \rightarrow \{SS \,\&\, [L, M^+ \mid L'] \,\&\, \{M{\in}\mathcal{I}, IK\}\} \quad (3.3)$$

Variables L and L' denote lists of input and output messages of the form m^+ or m^- within a strand, IK denotes a set of intruder facts ($m{\in}\mathcal{I}$ or $m{\notin}\mathcal{I}$), and SS denotes a set of strands.

Rule (3.1) used in a forwards sense means that the intruder knows the message that a strand is waiting to receive, but when executed backwards by narrowing on a symbolic state with variables *SS'* and *IK'* it may either unify the input message with some term already in the intruder knowledge or unify *SS'* or *IK'* with parts of the rule, thus adding new information to the symbolic state. Rule (3.2) used in a forwards sense, means that the intruder did not learn a message generated by a strand, but when executed backwards by narrowing on a symbolic state with variables *SS'* and *IK'*, it either moves the bar to the left or unifies variable *SS'* with parts of the rule. Rule (3.3) used in a forwards sense means that the intruder learns a message M generated by a strand that it did not know before (expressed by $M{\notin}\mathcal{I}$), but when executed backwards by narrowing on a symbolic state with variables *SS'* and *IK'*, it either detects the instant where the intruder is learning a message and, thus, transforms a fact $m{\in}\mathcal{I}$ into $m{\notin}\mathcal{I}$ to identify the transition in which the fact $M{\in}\mathcal{I}$ was learned, or unifies variables *SS'* or *IK'* with parts of the rule, thus adding new information to the symbolic state.

[4]We do not include the fresh variables in rules (3.1), (3.2), and (3.3) for simplicity, but a expression $:: r_1, \ldots, r_k ::$ should always appear before each strand.

For an unbounded number of sessions, we have extra rewrite rules (one for each positive message in a protocol or intruder strand) that dynamically introduce additional strands into a state:

$$\begin{array}{l} \forall\ [l_1, u^+, l_2] \in \mathcal{P} : \\ \{\,\{SS \,\&\, [\,l_1 | u^+, l_2\,] \,\&\, \{u \notin \mathcal{I}, IK\}\} \rightarrow \{SS \,\&\, \{u \in \mathcal{I}, IK\}\}\} \end{array} \tag{3.4}$$

Note that these rules are essential in a backwards sense, since they will dynamically introduce new strands guided by existing terms in the intruder knowledge. For example, the intruder decryption capability $[-(K), -(M), +(d(K, M)]$ produces the following extra rewrite rule adding a new strand (when the rules are executed backwards) if a message of the form $d(K, M)$ appears in the intruder knowledge:

$$\begin{array}{l} \{SS \,\&\, [-(K), -(M) \mid +(d(K, M)] \,\&\, \{(d(K, M) \notin \mathcal{I}, IK\}\} \\ \rightarrow \{SS \,\&\, \{d(K, M) \in \mathcal{I}, IK\}\} \end{array} \tag{3.5}$$

Definition 3.3 *Let $\mathcal{P}$ be a protocol with signature $\Sigma_{\mathcal{P}}$ and equational theory $E_{\mathcal{P}}$. We define the* backwards rewrite theory characterizing $\mathcal{P}$ *to be $(\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, R^{-1}_{B\mathcal{P}})$ where $R^{-1}_{B\mathcal{P}}$ is the result of reversing the rewrite rules in the set $\{(3.1), (3.2), (3.3)\} \cup \{(3.4)\}$.*

Example 3.4 For example, consider the Diffie-Hellman attack state shown in Example 3.2 (Page 48):

$$\begin{array}{l} \{:: r'' :: [-(a; b; Y), \\ \qquad +(b; a; exp(g, n(b, r''))), \\ \qquad -(e(exp(Y, n(b, r'')), SR)) \mid nil] \ \ \& \ SS \ \& \ \{SR \in \mathcal{I}\}\} \end{array}$$

and the Rule (3.5). We can apply this rule to our attack state modulo the equational theory for Diffie-Hellman. That is, the attack term above unifies with the left-hand side of the rule modulo the following cancellation equational rule $E_{\mathcal{P}}$:

$$d(Ke, e(Ke, M)) = M$$

where Ke denotes a key and M denotes a message. To be more concrete, the term SR unifies with the term $d(Ke, M)$ modulo the cancellation equational rules yielding the unifier $\theta = \{M \mapsto e(Ke, SR)\}$. Therefore,

we obtain the following predecessor state using the narrowing relation $\leadsto_{\theta,R_{\mathcal{P}}^{-1},E_{\mathcal{P}}}$ from the Diffie-Hellman attack state shown above:

$$\{ :: nil :: [\, nil, -(Ke), -(e(Ke, SR)) \mid +(SR) \,] \;\&$$
$$:: r'' :: [-(a; b; Y), +(b; a; exp(g, n(b, r''))), -(e(exp(Y, n(b, r'')), SR)) \mid nil \,]$$
$$\& \; SS \; \& \; \{SR \notin \mathcal{I}\}\}$$

■

3.5 General Requirements for Algebraic Theories

The Maude-NPA's unification technique is based on the computation of variants. In order to provide a *finite* set of unifiers, five specific requirements must be met by any algebraic theory specifying cryptographic functions that the user provides. If these requirements are not satisfied, Maude-NPA may exhibit non-terminating and/or incomplete behavior, and any completeness claims about the results of the analysis cannot be guaranteed (these conditions are defined in Section 2.1). We call theories that satisfy these criteria *admissible* theories.

In the following, let T be an algebraic theory with a decomposition (Σ, B, E_0) (see Section 2.1), such that E_0 are equations oriented as rules, and B is a set of equations denoting axioms. In the Maude-NPA we call such algebraic theory T specified by the user for the cryptographic functions of the protocol *admissible* if it satisfies the following requirements: (i) axioms B are either commutativity (C), or associativity-commutativity (AC), or associativity-commutativity-unit element (ACU), (ii) the rules E_0 are *confluent modulo* B, (iii) the rules E_0 are *terminating modulo* B, (iv) the rules E_0 are *coherent modulo* B, (v) (Σ, B, E_0) is a finite variant decomposition. Below we explain in more detail each of these requirements.

Axioms The axioms B can declare some binary operators in Σ to be commutative (with the `comm` attribute), associative-commutative (with

the `assoc` and `comm` attributes), or associative-communtative and identity (with the `assoc`,`comm`, and `id` attributes). No other combinations of axioms are allowed; that is, a function symbol has either no attributes, or only the `comm` attribute, or only the `assoc` and `comm` attributes, or only the `assoc`, `comm` and `id` attributes.

Confluence. The equations E_0 are called *confluent* modulo B if and only if for each term t in the theory $T = (\Sigma, E_0 \cup B)$, if we can rewrite t with E_0 modulo B in two different ways as: $t \longrightarrow^*_{E_0,B} u$ and $t \longrightarrow^*_{E_0,B} v$, then we can always further rewrite u and v to a common term up to identity modulo B. That is, u and v are essentially the same term, in the sense that they are equal modulo the axioms B.

Termination. The equations E_0 are called *terminating* modulo B if and only if all rewrite sequences terminate; that is, if and only if we never have an infinite sequence of rewrites

$$t_0 \rightarrow_{E_0,B} t_1 \rightarrow_{E_0,B} t_2 \ldots t_n \rightarrow_{E_0,B} t_{n+1} \ldots$$

Coherence. Rather than explaining the *coherence modulo* B notion in general (already explained in Section 2.1 in Page 27), we explain in detail its meaning in the case where it is needed for the Maude-NPA, namely, the case of associative-commutative (AC) symbols. The best way to illustrate the meaning of coherence is by its *absence*. Consider, for example, an exclusive-or operator $\oplus$ which has been declared AC. Now consider the equation $X \oplus X = 0$. This equation, if not completed by another equation, is *not* coherent modulo AC. What this means is that there will be term *contexts* in which the equation *should* be applied, but it cannot be applied. Consider, for example, the term $b \oplus (a \oplus b)$. Intuitively, we should be able to apply to it the above equation to simplify it to the term $a \oplus 0$ in one step. However, since we are using the weaker rewrite relation $\rightarrow_{E_0,B}$ instead of the stronger but much harder to implement relation $\rightarrow_{E_0/B}$, we cannot! The problem is that the equation cannot be applied (even if we match modulo AC) to either the top term $b \oplus (a \oplus b)$ or the subterm $a \oplus b$. We can however make our equation *coherent* modulo

AC by adding the extra equation $X \oplus X \oplus Y = 0 \oplus Y$, which, using also the equation $X \oplus 0 = X$, we can slightly simplify to the equation $X \oplus X \oplus Y = Y$. This second variant of our equation will now apply to the term $b \oplus (a \oplus b)$, giving the simplification $b \oplus (a \oplus b) \longrightarrow_{E_0,B} a$. Technically, what coherence means is that the weaker relation $\rightarrow_{E_0,B}$ becomes semantically equivalent to the stronger relation $\rightarrow_{E_0/B}$.

For the Maude-NPA, coherence is only an issue for AC and ACU symbols. And there is always an easy way, given a set E_0 of equations, to make them AC-coherent. The method is as follows. For any symbol f which is AC, and for any equation of the form $f(u, v) = w$ in E_0, we add also the equation $f(f(u, v), X) = f(w, X)$, where X is a new variable not appearing in u, v, w. In an order-sorted setting, we should give to X *the biggest sort possible*, so that it will apply in all generality. As an additional optimization, note that some equations may already be coherent modulo AC, so that we need not add the extra equation. For example, if the variable X has the biggest possible sort it could have, then the equation $X \oplus 0 = X$ is already coherent, since X will match "the rest of the $\oplus$-expression," regardless of how big or complex that expression might be, and of where in the expression a constant 0 occurs. For example, this equation will apply modulo AC to the term $(a \oplus (b \oplus (0 \oplus c))) \oplus (c \oplus a)$, with x matching the term $(a \oplus (b \oplus c)) \oplus (c \oplus a)$, so that we indeed get a rewrite $(a \oplus (b \oplus (0 \oplus c))) \oplus (c \oplus a) \rightarrow_{E_0,B} (a \oplus (b \oplus c)) \oplus (c \oplus a)$.

Finite Variant Decomposition. A decomposition (Σ, B, E_0) of an equational theory (Σ, E) is a *finite variant decomposition* or has the *finite variant (FV) property* iff for each term t, a complete set of its most general variants is finite. Again, rather than explaining this notion in detail, we illustrate its meaning by a theory that does not have the FV property.

There is one notable theory that does not have the FV property, the theory of encryption homomorphic over another operator, that is $e(X * Y, K) = e(X, K) * e(Y, K)$, where $*$ is an operator that may have some other equational properties, shown not to satisfy the FV property when the homomorphic equation is in E_0 in [Comon-Lundh and Delaune, 2005]. In reality, Common and Delaune consider the case in which e has only one argument and $*$ is exclusive-or, but their case can be considered

as corresponding to a degenerate case of e with two arguments, in which only one key is used. Moreover, their counterexamples apply to any subtheory of the theory they use for $*$, including the abelian group theory, AC, and the free theory.

Comon and Delaune prove their result by producing counterexamples to a property that they show to be equivalent to FV property. However, it is also easy to produce direct counterexamples. Assuming that the distributive equation is oriented as the rewrite rule $e(X * Y) \rightarrow e(X) * e(Y)$, there is an infinite number of most general variants for the term $e(Z)$:

$$(e(Z), id), (e(Z_1) * e(Z_2), \{Z \mapsto Z_1 * Z_2\}), (e(Z_1) * e(Z_2) * e(Z_3), \{Z \mapsto Z_1 * Z_2 * Z_3\}), \ldots$$

If the equation is oriented as $e(X) * e(Y) \rightarrow e(X * Y)$, then there is an infinite number of most general variants of the term $e(Z * W)$:

$$(e(Z * W), id), (e(e(Z_1 * W_1))\{Z \mapsto e(Z_1), W \mapsto e(W_1)\}), (e(e(e(Z_1 * W_1)))\{Z \mapsto e(e(Z_1)), W \mapsto e(e(W_1))\}), \ldots$$

3.6 Protocol Specification in Maude-NPA

In this section we briefly recall how to specify a protocol and all its relevant items in the current version of the Maude-NPA. Note that, since we are using Maude also as the specification language, each declaration has to be ended by a space and a period.

The beginning of the protocol specification is used to define the different types of data and function symbols of the protocol, as well as the algebraic properties of those protocol functions. There are two types of algebraic properties in Maude-NPA: (i) *equational axioms*, such as commutativity, or associativity-commutativity, called *axioms*, (ii) *equational rules*, called *equations*. Axioms are specified within the typed function symbol declarations, whereas equations are specified separately with the symbol `eq` and attribute `variant`.

We use the keyword `STRANDS-PROTOCOL` for storing the strands denoting the actions of the protocol honest principals, and the keyword `STRANDS-DOLEVYAO` for storing the strands denoting the operations an intruder can perform, or Dolev-Yao rules [Dolev and Yao, 1983]. Each

such action can be specified by an intruder strand consisting of a (possibly empty) sequence of negative nodes, followed by a single positive node.

As an example, we provide the specification of the Diffie-Hellman protocol explained in Example 3.1 (in Page 45). This protocol specification includes sorts to represent names, nonces and secret information, namely, sorts Name, Nonce, and Secret, respectively. Sort Key is used to represent symmetric encryption keys. Sorts Gen, Exp, GenvExp, and NeNonceSet, are used to represent the different data necessary to perform the Diffie-Hellman exponentiation and modular multiplication.

The Diffie-Hellman *generator* of the multiplicative group is denoted by constant g, which has sort Gen. Operator exp denotes exponentiations. Note that, since both sorts Gen and Exp are subsorts of GenvExp, operator exp can be used to represent messages of the form g^X as terms of the form exp(g,X) and messages of the form g^{XY} as terms of the form exp(exp(g,X),Y). Operator _*_ represents the associative-commutative multiplication operation on nonces and products of such nonces (note that sort Nonce is subsort of NeNonceSet). The remaining operators are as explained in Example 3.1.

```
*** Sorts and operators
sorts Name Nonce NeNonceSet Gen Exp Key GenvExp Secret .
subsort Gen Exp < GenvExp .
subsort Name NeNonceSet GenvExp Secret Key < Msg .
subsort Exp < Key .
subsort Name < Public .
subsort Gen < Public .

--- Secret
op sec : Name Fresh -> Secret [frozen] .

--- Nonce operator
op n : Name Fresh -> Nonce [frozen] .

--- Encryption
op e : Key Msg -> Msg [frozen] .
op d : Key Msg -> Msg [frozen] .
```

```
--- Exp
op exp : GenvExp NeNonceSet -> Exp [frozen] .

--- Gen
op g : -> Gen .

--- NeNonceSet
subsort Nonce < NeNonceSet .
op _*_ : NeNonceSet NeNonceSet -> NeNonceSet
         [frozen assoc comm] .

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

--- names
ops a b i : -> Name .

*** Algebraic properties
var W : Gen .
var K : Key .
vars Y Z : NeNonceSet .
var M : Msg .

eq exp(exp(W,Y),Z)  = exp(W, Y * Z) [variant] .
eq e(K,d(K,M)) = M [variant] .
eq d(K,e(K,M)) = M [variant] .

*** Strands specification
vars A B : Name .
vars  r r' : Fresh .
vars M M1 M2 : Msg .
vars Ke : Key .
var GE : GenvExp .
vars NS1 NS2 : NeNonceSet .
var Sr : Secret .

eq STRANDS-DOLEVYAO =
   :: nil :: [ nil | -(M1 ; M2), +(M1), nil ] &
   :: nil :: [ nil | -(M1 ; M2), +(M2), nil ] &
```

```
   :: nil :: [ nil | -(M1), -(M2), +(M1 ; M2), nil ] &
   :: nil :: [ nil | -(Ke), -(M), +(e(Ke,M)), nil ] &
   :: nil :: [ nil | -(Ke), -(M), +(d(Ke,M)), nil ] &
   :: nil :: [ nil | -(NS1), -(NS2), +(NS1 * NS2), nil ] &
   :: nil :: [ nil | -(GE), -(NS), +(exp(GE,NS)), nil ] &
   :: r ::   [ nil | +(n(i,r)), nil ] &
   :: nil :: [ nil | +(g), nil ] &
   :: nil :: [ nil | +(A), nil ]
[nonexec] .

eq STRANDS-PROTOCOL =
   :: r,r' ::
   [nil | +(A ; B ; exp(g,n(A,r))),
          -(A ; B ; XE),
          +(e(exp(XE,n(A,r)),sec(A,r'))), nil] &
   :: r ::
   [nil | -(A ; B ; XE),
          +(A ; B ; exp(g,n(B,r))),
          -(e(exp(XE,n(B,r)),Sr)), nil]
[nonexec] .
```

Finally, let us explain the use of the `frozen` and `nonexec` attributes in the example protocol specification shown above. The `frozen` attribute is technically necessary to tell Maude not to attempt to apply rewrites in arguments of those symbols. This attribute must be included in all operator declarations in Maude-NPA specifications, excluding constants. The `nonexec` attribute is technically necessary to tell Maude not to use an equation or rule within its standard execution, since it will be used only at the Maude-NPA level rather than the Maude level. The `nonexec` attribute must be included at the end of the equation declarations used to specify the protocol and intruder strands.

3.6.1 Protocol States

In Maude-NPA, each state associated to the protocol execution (i.e., a backwards search) is represented by a term with five 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 component, the sequence of messages, is `nil` for any attack state at the beginning of the backwards search and records the actual sequence of messages exchanged so far in the backwards search from the attack state. This sequence grows as the backwards search continues and some variables may be instantiated in the backwards search. It gives a complete description of an attack when an initial state is reached but this component is intended for the benefit of the user, and is not actually used in the backward search itself, except just by recording it.

The fourth component contains information about the search space that the tool creates to help manage[5] its search. It does not provide any information about the attack itself, and is currently only displayed by the tool to help in debugging.

Finally, the last component allows the user to verify *authentication* properties or to specify conditions that should not happen during the analysis (see Section 3.3). By doing so, the Maude-NPA will discard any generated state that matches any of the conditions expressed by this "never pattern" and, thus, it will reduce the search space. In previous implementations of the Maude-NPA, the "never patterns" were not part of the state and were used only in the specification of the attack state. The idea of including the "never patterns" into a state is that it allows the user to specify attack states in a more expressive way, since now the specification of the state itself and the states within the "never pattern" can share variables.

[5]Indeed, we use the fourth component of a protocol state to store the data related to the Super Lazy Intruder optimization (see Section 4.5).

3.6.2 Attack States

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 intruder knowledge can also contain inequalities (the condition that a term is not equal to some other term) and, thus we can also include them in the intruder knowledge component of an attack state. The message sequence, auxiliary data components should have just the empty symbol `nil`. In the following we provide the specification of an attack state as an illustrating example.

Example 3.5 Let us consider the specification in Maude-NPA's syntax of the Diffie-Hellman protocol shown in Page 56. The attack state below denotes an attack in which the intruder is able to learn the secret message sent by Alice, denoted by the intruder knowledge fact `sec(a,r') inI`. This attack state includes the strand with the last received message of the protocol, i.e. Bob's strand with the vertical bar at the end.

```
eq ATTACK-STATE(1)
 = :: r ::
   [nil, -(a ; b ; XE),
         +(a ; b ; exp(g,n(b,r))),
         -(e(exp(XE,n(b,r)),sec(a,r'))) | nil]
     || sec(a,r') inI
     || nil
     || nil
     || nil
  [nonexec] .
```

■

Regarding the last component, the never pattern, it can be either the empty symbol `nil` or either the word `never` followed by a set of pairs of the form `(StrandSet || IntruderKnowledge)`, where `StrandSet` and `IntruderKnowledge` denote, respectively, the set of strands and the set of intruder knowledge terms of the states that should be discarded by the Maude-NPA during the backwards reachability analysis.

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.

Example 3.6 Let us consider again the specification in Maude-NPA's syntax of the Diffie-Hellman protocol shown in Page 56. The authentication attack pattern for Diffie-Hellman of Example 3.2 is specified in Maude-NPA's syntax as follows:

```
eq ATTACK-STATE(0)
 = :: r ::
   [nil, -(a ; b ; Y),
         +(b ; a ; exp(g,n(b,r))),
         -(e(exp(Y,n(b,r)),sec(a,r'))) | nil]
   || empty
   || nil
   || nil
   || never
   *** Pattern for authentication
   (:: R:FreshSet ::
   [nil | +(a ; b ; Y),
          -(b ; a;  exp(g,n(b,r))),
          +(e(X,sec(a,r'))), nil]
    & S:StrandSet || K:IntruderKnowledge)
[nonexec] .
```

■

3.7 Maude-NPA Commands

The commands `run`, `summary`, and `initials` are the tool's commands for attack search. They are invoked by reducing them in Maude, that is, by typing the Maude `red` command followed by the corresponding Maude-NPA command, followed by a space and a period. To use them we must specify the attack state we are searching for and the number of backwards reachability steps we want to compute. For example, the Maude-NPA command

```
run(0,10)
```

tells Maude-NPA to construct the backwards reachability tree up to depth 10 for the attack state designated with natural number 0. The Maude-NPA `run` command yields the set of states found at the leaves of the backwards reachability tree of the specified depth that has been generated. When the user is not interested in the current states of the reachability tree, he/she can use the Maude-NPA `summary` command, which outputs just the number of states found at the leaves of the reachability tree and how many of those are initial states, i.e., solutions for the attack. For instance, when we give the reduce command in Maude with the Maude-NPA command `summary(0,2)` as shown below for the Diffie-Hellman example, the tool returns:

```
red summary(0,2) .
result Summary: States>> 6 Solutions>> 0
```

The initial state representing the authentication Diffie-Hellman attack is found in twelve steps. That is, if we type:

```
red summary(0,12) .
```

the tool outputs:

```
red summary(0,12) .
result Summary: States>> 2 Solutions>> 2
```

A slightly different version of the `run` command, called `initials`, outputs only the initial states, instead of all the states at the leaves of the backwards reachability tree. Thus, if we type:

```
red initials(0,12) .
```

for the Diffie-Hellman example protocol, our tool outputs two initial states, which implies that the attack state has been proved reachable and the protocol is insecure. One of these two initial states is the one explained in Example 3.2 in Page 47. The sequence of exchanged messages of this initial state is as shown below:

```
generatedByIntruder(#1:NeNonceSet),
generatedByIntruder(a ; b ; exp(g, #1:NeNonceSet)),
-(a ; b ; exp(g, #1:NeNonceSet)),
+(a ; b ; exp(g, n(b, #0:Fresh))),
-(a ; b ; exp(g, n(b, #0:Fresh))),
+(b ; exp(g, n(b, #0:Fresh))),
-(b ; exp(g, n(b, #0:Fresh))),
+(exp(g, n(b, #0:Fresh))),
-(exp(g, n(b, #0:Fresh))),
-(#1:NeNonceSet),
+(exp(g, #1:NeNonceSet * n(b, #0:Fresh))),
generatedByIntruder(a ; #5:Name ; exp(g, #4:NeNonceSet)),
+(a ; #5:Name ; exp(g, n(a, #3:Fresh))),
-(a ; #5:Name ; exp(g, n(a, #3:Fresh))),
+(#5:Name ; exp(g, n(a, #3:Fresh))),
-(#5:Name ; exp(g, n(a, #3:Fresh))),
+(exp(g, n(a, #3:Fresh))),
generatedByIntruder(#4:NeNonceSet),
-(exp(g, n(a, #3:Fresh))),
-(#4:NeNonceSet),
+(exp(g, #4:NeNonceSet * n(a, #3:Fresh))),
-(a ; #5:Name ; exp(g, #4:NeNonceSet)),
+(e(exp(g, #4:NeNonceSet * n(a, #3:Fresh)), sec(a, #2:Fresh))),
resuscitated(exp(g, #4:NeNonceSet * n(a, #3:Fresh))),
-(exp(g, #4:NeNonceSet * n(a, #3:Fresh))),
-(e(exp(g, #4:NeNonceSet * n(a, #3:Fresh)), sec(a, #2:Fresh))),
+(sec(a, #2:Fresh)),
-(exp(g, #1:NeNonceSet * n(b, #0:Fresh))),
-(sec(a, #2:Fresh)),
+(e(exp(g, #1:NeNonceSet * n(b, #0:Fresh)), sec(a, #2:Fresh))),
-(e(exp(g, #1:NeNonceSet * n(b, #0:Fresh)), sec(a, #2:Fresh)))
```

where the terms enclosed by the expression `generatedByIntruder` denote terms trivially learnt by the intruder. These expressions are generated due to a state space reduction technique called the "Super-Lazy intruder" (see Section 4.5.2 for further details).

This corresponds to the following textbook version of the attack:

1. $A \rightarrow I : A\,;\, I\,;\, g^{N_A}$
2. $I \rightarrow A : I\,;\, A\,;\, g^{N_I}$
3. $A \rightarrow I : \{sec(A,r)\}_{g^{N_A N_I}}$
4. I decrypts $\{sec(A,r)\}_{g^{N_A N_I}}$ and obtains $sec(A,r)$
5. $I \rightarrow B : I\,;\, B\,;\, g^{N'_I}$
6. $B \rightarrow I : B\,;\, I\,;\, g^{N_B}$
7. $I \rightarrow B : \{sec(A,r)\}_{g^{N_B N'_I}}$

Maude-NPA applies by default the optimization techniques explained in Chapter 4 in the protocol analysis it performs. However, these optimizations can be disabled when executing the `summary`, `run`, and `initials` commands, as explained in [Escobar et al., 2009b]. Indeed, the experiments described in Section 4.6 have been performed disabling certain optimizations.

Chapter 4

State Space Reduction in the Maude-NPA

In this chapter we describe some of the major state space reduction techniques that we have implemented in Maude-NPA, and provide completeness proofs and experimental evaluations.

First, Section 4.1 motivates the work presented in this chapter. In Section 4.2, we give an overview of the various state space reduction techniques that have been introduced to control state explosion. In Sections 4.3, 4.4, and 4.5 we describe the state space reduction techniques and give proofs of their completeness as well as showing their relations to other optimization techniques in the literature. In Section 4.3, we first briefly describe how automatically generated grammars provide the main reduction that cuts down the search space. In this section, we also describe the early detection of inconsistent states (that will never reach an initial state). In Section 4.4, we obtain a second important state-space reduction by detecting redundant states using several optimizations: (i) reducing the number of logical variables present in a state, (ii) giving priority to input messages in strands, and (iii) a relation of transition subsumption (to discard transitions and states already being processed in another part of the search space). In Section 4.5, we obtain a third important state-space reduction by defining the super-lazy intruder, which delays the generation of substitution instances as much as possible. In Section 4.6 we describe our experimental evaluation of these state-space reduction techniques. Finally, Section 4.7 concludes the chapter.

These results have been published in [Escobar et al., 2014a].

4.1 Motivation

One technique for preventing infinite searches is the generation of formal grammars describing terms unreachable by the intruder (see [Meadows, 1996a; Escobar et al., 2006] and Section 4.3.1). However, grammars do not prune out all infinite searches, since unbounded session security is undecidable, and there is a need for other techniques. Moreover, even when a search space is finite it may still be necessary to reduce it to a manageable size, and state space reduction techniques for doing that will be necessary.

The results of our experimental evaluation demonstrate an average state-space size reduction of 95% (i.e., the average size of the reduced state space is 5% of that of the original one) in the examples we have evaluated; counting states with and without optimizations up to a depth bound of 5. Furthermore, we show our combined techniques effective in obtaining a *finite* state space for all protocols in our experiments, whereas the state space will be infinite without our optimizations. In our experimental evaluation we have considered a depth bound of 5 because the number of states generated in the case without optimizations often grows so large that it becomes unfeasible to count it even for moderate depth sizes, while the optimized state space can still be counted. Indeed, in most of the experiments without optimizations we could not count the number of generated states for a depth greater than 5. Of course, due to the undecidability of the unbounded session case, termination cannot be guaranteed and we have encountered cases where it does fail to terminate (see Table 4.2 in Page 108).

The optimizations we describe in this chapter were designed specifically for Maude-NPA, and work within the context of Maude-NPA search techniques. However, although different tools use different models and search algorithms, they all have a commonality in their syntax and semantics that means that, with some adaptations, optimization techniques developed for one tool or type of tools can be applied to different tools as well. Indeed, in Section 1.3 we have already seen such common techniques arise, for example the technique of giving priority to input or

output messages respectively when backwards or forwards search is used (used by us and by Shmatikov and Stern [1998]), the use of lazy evaluation techniques (used in constraint-evaluation based approaches, and by us in a somewhat different form), and the identification of premature use of nonces (used by us and Scyther [Cremers, 2008b]).

4.2 Overview of State Space Reduction Techniques

In this section we present Maude-NPA's state space reduction techniques. They are applied when a state is generated in the backwards narrowing search. A number of tests are applied. If a state is identified as *unproductive*, that is, such that its removal does not affect reachability of the final state one way or the other, it is also removed. Techniques for removing unproductive states, which are later refined into unreachable and redundant states, are described in Sections 4.3 and 4.4. A state can also be identified as potentially leading to a state space explosion, in which case it may be delayed. Such a delay will be complete, but not necessarily sound; in this case, the delay may have to be reversed as the narrowing tree is generated in order to maintain soundness. We have developed one technique that falls into this class: the super-lazy intruder, which is described in Section 4.5.

In the remainder of this chapter, we make use of a very general completeness result satisfied by Maude-NPA. This result, stated by the following corollary, can be deduced from Theorem 2.5 (see Page 30).

Corollary 4.1 (Completeness) [Escobar et al., 2006] *Given a topmost rewrite theory* $\mathcal{R}_{\mathcal{P}} = (\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, R_{B\mathcal{P}})$ *representing protocol* $\mathcal{P}$*, and a non-initial state* St *(with logical variables), if there is a substitution* σ *and an initial state* St_{ini} *such that* $St\sigma \rightarrow^*_{R^{-1}_{B\mathcal{P}}, E_{\mathcal{P}}} St_{ini}$*, then there are substitutions* σ', ρ *and an initial state* St'_{ini} *such that* $St \rightsquigarrow^*_{\sigma', R^{-1}_{B\mathcal{P}}, E_{\mathcal{P}}} St'_{ini}$*,* $\sigma =_{E_{\mathcal{P}}} \sigma' \circ \rho$*, and* $St_{ini} =_{E_{\mathcal{P}}} (St'_{ini})\rho$*.*

We have developed means of detecting two kinds of unproductive states: *unreachable* and *redundant* states. These are defined below.

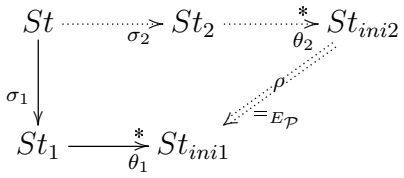

Figure 4.1: St_1 is a redundant state

Definition 4.2 (Unreachable States) *Given a topmost rewrite theory $\mathcal{R}_{\mathcal{P}} = (\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, R_{B\mathcal{P}})$ representing protocol $\mathcal{P}$, a non-initial state St (with logical variables) is* unreachable *if there is no sequence $St \leadsto^*_{\sigma, R^{-1}_{B\mathcal{P}}, E_{\mathcal{P}}} St_{ini}$ leading to an initial state St_{ini}.*

Definition 4.3 (Redundant States) *Given a topmost rewrite theory $\mathcal{R}_{\mathcal{P}} = (\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, R_{B\mathcal{P}})$ representing protocol $\mathcal{P}$ and a non-initial state St (with logical variables), a backwards narrowing step $St \leadsto_{\sigma_1, R^{-1}_{B\mathcal{P}}, E_{\mathcal{P}}} St_1$ such that St_1 is a non-initial state is called* redundant *(or just state St_1 is identified as* redundant*) if for any initial state St_{ini1} reachable from St_1, i.e., $St_1 \leadsto^*_{\theta_1, R^{-1}_{B\mathcal{P}}, E_{\mathcal{P}}} St_{ini1}$, there are states $St_2 \neq_{E_{\mathcal{P}}} St_1$ and St_{ini2}, a narrowing step $St \leadsto_{\sigma_2, R^{-1}_{B\mathcal{P}}, E_{\mathcal{P}}} St_2$, a narrowing sequence $St_2 \leadsto^*_{\theta_2, R^{-1}_{B\mathcal{P}}, E_{\mathcal{P}}} St_{ini2}$, and a substitution ρ such that $\sigma_1 \circ \theta_1 =_{E_{\mathcal{P}}} \sigma_2 \circ \theta_2 \circ \rho$ and $St_{ini1} =_{E_{\mathcal{P}}} (St_{ini2})\rho$.*

Redundant states are represented graphically in Figure 4.1, where plain arrows are quantified universally, dotted arrows are quantified existentially, and a double-dotted arrow means equational matching using the direction of the arrow from a more general term to a less general term.

There are three reasons for wanting to detect and remove unproductive states. One is to reduce, if possible, the initially infinite search space to a finite one, as it is sometimes possible to do with the use of grammars, by removing unreachable states. Another is to reduce the size of a (possibly finite) search space by eliminating unreachable states early, i.e., before they are eliminated by exhaustive search. This elimination of unreachable states can have an effect far beyond eliminating a single node in the search space, since a single unreachable state may appear multiple times and/or have multiple descendants. Finally, if there are several steps leading to the same initial state, as for redundant states,

then it is also possible to use various partial order reduction techniques that can further shrink the number of states that need to be explored.

4.3 Identifying Unreachable States

In this section we describe the various techniques Maude-NPA uses to identify unreachable states. These techniques have been adapted from those used by its ancestor, NPA.

There are two ways in which Maude-NPA identifies unreachable states. One is the use of inductive techniques to define grammars that characterize terms that can never be learned by the intruder. This has been described in detail in previous work, e.g. [Escobar et al., 2006] and [Meadows, 1996b], so we give only a brief overview here in Section 4.3.1. The other is the identification of states that describe impossible events, e.g. states in which an intruder learns a term containing a nonce that has not yet been created. This is described in Section 4.3.2.

4.3.1 Grammars

The Maude-NPA's ability to reason effectively about a protocol's algebraic properties is a result of its combination of symbolic reachability analysis using narrowing modulo equational properties (see Section 2.1), together with its grammar-based techniques for reducing the size of the search space. The key idea of grammars is to detect terms t in positive facts $t \in \mathcal{I}$ of the intruder's knowledge of a state St that will never be transformed into a negative fact $t\theta \notin \mathcal{I}$ in any initial state St' backwards reachable from St. This means that St can never reach an initial state and therefore it can be safely discarded. Here we briefly explain how grammars work as a state space reduction technique and refer the reader to [Meadows, 1996b; Escobar et al., 2006] for further details.

Automatically generated grammars $\langle G_1, \ldots, G_m \rangle$ represent unreachability information (or co-invariants), i.e., typically infinite sets of states unreachable from an initial state. These automatically generated grammars are very important in our framework, since in the best case they can reduce the infinite search space to a finite one, or, at least, can drastically reduce the search space.

Maude-NPA generates grammars completely automatically, inferring initial grammars from the protocol specification, and using built-in inference rules to generate new grammars.

As an example of how grammars work, let us consider the following attack pattern of the Diffie-Hellman protocol explained in Example 3.1 (see Page 45), in which Bob completes the protocol and the intruder is able to learn the secret that Bob received:

$$:: r' :: [-(A; B; E'), +(B; A; exp(g, n(B, r'))),\\ -(e(exp(E', n(b, r')), SR)) \mid nil] \ \& \ SS \ \& \ \{SR{\in}\mathcal{I}\} \quad (\dagger)$$

This pattern contains the intruder knowledge fact $SR{\in}\mathcal{I}$. If we run Maude-NPA without any optimizations, it will generate a state containing the facts $(M; SR){\in}\mathcal{I}$ and $SR{\notin}\mathcal{I}$, then a state containing the facts $(M'; M; SR){\in}\mathcal{I}$, $(M; SR){\notin}\mathcal{I}$, and $SR{\notin}\mathcal{I}$, then a state containing the facts $(M''; M'; M; SR){\in}\mathcal{I}$, $(M'; M; SR){\notin}\mathcal{I}$, $(M; SR){\notin}\mathcal{I}$, and $SR{\notin}\mathcal{I}$, and so on, producing an infinite sequence of states. We can describe the sequence beginning with the state containing the facts $(M; SR){\in}\mathcal{I}$ and $SR{\notin}\mathcal{I}$ using the following grammar:

```
grl M inL => (M' ; M) inL . ;
grl M notInI  => (M' ; M) inL .
```

where the first production describes the concatenation of two terms, the second of which is in the language L, and the second production gives the concatenation of two terms, the second of which is not yet known by the intruder.

We now want to see if all members of the language characterized by this grammar are unlearnable by the intruder. In order to do this, we attempt to show that, if the intruder learns a member of the language, it must have already known a member of the language. This is done by giving each production of the grammar to Maude-NPA as a goal, and using it to determine that in each preceding state the intruder knows a member of the language. Thus, if we give the state $(M'; M){\in}\mathcal{I}$ to Maude-NPA, keeping in mind that the M is a member of L, then we can see that one of the preceding states it finds contains $M{\in}\mathcal{I}, M'{\in}\mathcal{I}$. Since M is a member of L, this state requires that the intruder knows a member of the language.

It is unlikely that initially all preceding states will require that the intruder knows a member of the language. Whenever that is the case, Maude-NPA employs heuristics to add or modify a production (by adding constraints of the form `M notLeq Pattern`) so that some term known by the intruder is in the language defined by the new grammar. This process is iterated until it either reaches a fixed point, or no more heuristics can be applied. These heuristics and a proof of correctness are given in [Escobar et al., 2006].

For example, the initial grammar described above terminates in the following:

```
grl M inL => e(E, M) inL . ;
grl M inL => d(E, M) inL . ;
grl M inL => (M ; M') inL . ;
grl M inL => (M' ; M) inL . ;
grl M notInI,
    M notLeq exp(g, n(A, r)),
    M notLeq B ; exp(g, n(A, r')) => (M' ; M) inL .
```

where all the productions and exceptions refer to normal forms of messages w.r.t. the equational theory $E_{\mathcal{P}}$.

Intuitively, the last production rule in the grammar above says that any term with normal form $(M'; M)$ cannot be learned by the intruder if the subterm M is different from $exp(g, n(A, r))$ and $B; exp(g, n(A, r'))$ (i.e., it does not match such patterns) and the constraint $M \notin \mathcal{I}$ appears explicitly in the intruder's knowledge of the current state being checked for unreachability (described by constraints of the form `M notInI`). Moreover, any term of any of the following normal forms: $e(E, M)$, $d(E, M)$, $(M'; M)$, or $(M; M')$ cannot be learned by the intruder if subterm M is a member of the language described by the above grammar.

The interested reader can determine that the term we were originally interested in, i.e., the term $(M; SR)$, where SR is not yet known by the intruder, is indeed a member of the language.

In order for Maude-NPA to generate grammars, it needs a set of initial grammars to start from. In NPA, users had to define their own initial grammars. Maude-NPA, however, generates initial grammars automatically. For any intruder strand of the form $[(M_1)^-, \ldots, (M_k)^-, (f(M_1, \ldots, M_k)^+]$, it generates $k + 1$ initial grammars: an initial grammar with the

production $f(M_1, \ldots, M_k) \in L$, and, for each negative term M_i, an initial grammar with the production $M_i \notin \mathcal{I} \Rightarrow f(M_1, \ldots, M_k) \in L$.

4.3.2 Early Detection of Inconsistent States

There are several types of states that are always unreachable or inconsistent. We give examples below.

Example 4.4 Consider again the attack pattern (†) in Page 70. After a couple of backwards narrowing steps, the Maude-NPA finds the following state, where the intruder learns $e(exp(E', n(B, r')), SR)$ by assuming it can learn $exp(E', n(B, r'))$ and SR and combine them:

$$
\begin{array}{l}
[\ nil \mid\ -(exp(E', n(B, r'))), -(SR), \\
\qquad +(e(exp(E', n(B, r')), SR))\ ]\ \& \\
:: r' :: \\
[\ -(A; B; E'), +(B; A; exp(g, n(B, r')))\ \mid \\
\quad -(e(exp(E', n(B, r')), SR))\ ]\ \& \\
(SR \in \mathcal{I},\ exp(E', n(B, r')) \in \mathcal{I},\ e(exp(E', n(B, r')), SR) \notin \mathcal{I})
\end{array}
\qquad (\ddagger)
$$

From this state, the intruder tries to learn SR by assuming it can learn messages $e(exp(E', n(B, r')), SR)$ and $exp(E', n(B, r'))$ and combines them in a decryption:

$$
\begin{array}{l}
[\ nil \mid\ -(exp(E', n(B, r'))), -(e(exp(E', n(B, r')), SR)), +(SR)\ ]\ \& \\
[\ nil \mid\ -(exp(E', n(B, r'))), -(SR), +(e(exp(E', n(B, r')), SR))\ ]\ \& \\
:: r' :: \\
[\ -(A; B; E'), +(B; A; exp(g, n(B, r')))\ \mid\ -(e(exp(E', n(B, r')), SR))\ ]\ \& \\
(SR \in \mathcal{I},\ exp(E', n(B, r')) \in \mathcal{I}, \\
e(exp(E', n(B, r')), SR) \in \mathcal{I},\ e(exp(E', n(B, r')), SR) \notin \mathcal{I})
\end{array}
$$

But then this state is inconsistent, since we have both the challenge $e(exp(E', n(B, r')), SR) \in \mathcal{I}$ and the already learned message $e(exp(E', n(B, r')), SR) \notin \mathcal{I}$ at the same time, violating the *learn-only-once* condition in Maude-NPA. ■

If Maude-NPA attempts to search beyond an inconsistent state, it will never find an initial state. For this reason, the Maude-NPA search strategy always marks the following types of states as unreachable, and does not search beyond them any further.

Proposition 4.5 *A state is marked as unreachable if one of the following situations holds:*

1. *A state St containing two contradictory facts $t \in \mathcal{I}$ and $t \notin \mathcal{I}$ (modulo $E_{\mathcal{P}}$) for a term t.*
2. *A state St whose intruder's knowledge contains the fact $t \notin \mathcal{I}$ and a strand of the form $[m_1^{\pm}, \ldots, t^{-}, \ldots, m_{j-1}^{\pm} \mid m_j^{\pm}, \ldots, m_k^{\pm}]$ (modulo $E_{\mathcal{P}}$).*
3. *A state St containing a fact $t \in \mathcal{I}$ such that t contains a fresh variable r and the strand in St indexed by r, i.e., $s =:: r_1, \ldots, r, \ldots, r_k :: [m_1^{\pm}, \ldots, m_{j-1}^{\pm} \mid m_j^{\pm}, \ldots, m_k^{\pm}]$, cannot produce r, i.e., r is not a subterm of any output message in $m_1^{\pm}, \ldots, m_{j-1}^{\pm}$.*
4. *A state St containing a strand of the form $s = [m_1^{\pm}, \ldots, t^{-}, \ldots, m_{j-1}^{\pm} \mid m_j^{\pm}, \ldots, m_k^{\pm}]$ for some term t such that t contains a fresh variable r and the strand in St indexed by r cannot produce r.*

Proof. The proofs of unreachability of each case are given below.

1. After backwards narrowing, this will result in a state that violates the "intruder-learns-only-once" rule.
2. This state will become a case of 1 after backwards narrowing.
3. In order for t to be found by the intruder, some other strand besides the strand in St indexed by r would need to produce it. But that strand would also need to be indexed by r, which contradicts the unique origin of fresh values.
4. We first note that any backward narrowing step will leave strand s still unable to produce r. Moreover, eventually, a backward narrowing step must result in the addition of $t \in \mathcal{I}$. Thus this state becomes a case of 3 after backwards narrowing. □

4.4 Redundant States

In this section we describe how Maude-NPA identifies and removes redundant states.

4.4.1 Limiting Dynamic Introduction of New Strands

As pointed out in Section 3.4, rules of type (3.4) are intended to be the only ones that introduce new strands. Rules of type (3.1), (3.2), and (3.3) are not intended for such an introduction. However, unless they are modified, they will introduce new strands, but in an unproductive way. That is, new strands can also be introduced by unification of a state containing a variable SS denoting a set of strands and one of the rules of (3.1), (3.2), and (3.3), where variables L and L' denoting lists of input/output messages will be introduced by instantiation of SS. The same can happen with new intruder facts of the form $X{\in}\mathcal{I}$, where X is a variable, by instantiation of a variable IK denoting the rest of the intruder knowledge.

Example 4.6 Consider a state St of the form $SS\,\&\,IK$ where SS denotes a set of strands and IK denotes a set of facts in the intruder's knowledge. Now, consider Rule (3.1):

$$SS'\,\&\,[L \mid M^-, L']\,\&\,(M{\in}\mathcal{I}, IK') \rightarrow SS'\,\&\,[L, M^- \mid L']\,\&\,(M{\in}\mathcal{I}, IK')$$

The following backwards narrowing step applying such a rule can be performed from $St = SS\,\&\,IK$ using the unifier $\sigma = \{SS \mapsto SS'\,\&\,[L, M^- \mid L'], IK \mapsto (M{\in}\mathcal{I}, IK')\}$

$$SS\ \&\ IK \overset{\sigma}{\rightsquigarrow}_{R^{-1}_{B^{\mathcal{P}}}, E_{\mathcal{P}}} SS'\,\&\,[L \mid M^-, L']\,\&\,(M{\in}\mathcal{I}, IK')$$

but this backwards narrowing step is unproductive, since it is not guided by the information in the attack state. Indeed, the same rule can be applied again using variables SS' and IK' and this can be repeated many times. ■

In order to avoid a huge number of unproductive narrowing steps by useless instantiation, we allow the introduction of new strands and/or

new intruder facts *only by rule application* instead of just by unification. For this, we do two things:

1. we remove any of the following variables from attack patterns: SS denoting a set of strands, IK denoting a set of intruder facts, and L, L' denoting a set of input/output messages; and

2. we replace Rule (3.1) by the following Rule (4.1), since we no longer have a variable denoting a set of intruder facts that has to be instantiated:

$$SS \,\&\, [L \mid M^-, L'] \,\&\, (M{\in}\mathcal{I}, IK) \rightarrow SS \,\&\, [L, M^- \mid L'] \,\&\, IK \quad (4.1)$$

One might imagine that Rule (3.3) and rules of type (3.4) must also be modified in order to remove the $M{\in}\mathcal{I}$ expression from the intruder's knowledge of the right-hand side of each rule. However, this is not so, since, by keeping the expression $M{\in}\mathcal{I}$, we force the backwards application of the rule only when there is indeed a message for the intruder to be learned. This provides some form of on-demand evaluation of the protocol.

Since this optimization is achieved by putting restrictions on attack patterns and rewrite rules, the soundness proof is trivial and thus omitted. However, a proof of completeness is still needed. The set of rewrite rules actually used for backwards narrowing is $\overline{R_{B\mathcal{P}}} = \{(4.1), (3.2), (3.3)\} \cup \{(3.4)\}$; note that (3.4) represents a set of rules. The following result ensures that $R_{B\mathcal{P}}$ and $\overline{R_{B\mathcal{P}}}$ compute similar initial states by backwards reachability analysis.

Definition 4.7 (Inclusion) *Given a topmost rewrite theory $\mathcal{R}_{\mathcal{P}} = (\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, R_{B\mathcal{P}})$ representing protocol $\mathcal{P}$, and two states St_1, St_2, we abuse notation and write $St_1 \subseteq St_2$ to denote that every state element (i.e., strand or intruder fact) in St_1 appears in St_2 (modulo $E_{\mathcal{P}}$).*

Proposition 4.8 *Let $\mathcal{R}_{\mathcal{P}} = (\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, R_{B\mathcal{P}})$ be a topmost rewrite theory representing protocol $\mathcal{P}$ and let $\overline{R_{B\mathcal{P}}}$ be defined as above. Let $St = ss \,\&\, SS \,\&\, (ik, IK)$. Let $ss = \{s_1, \ldots, s_n\}$ be a multiset of strands, $ik = \{k_1, \ldots, k_m\}$ be a set of intruder facts, SS is a variable denoting a set of strands, and IK is a variable denoting the intruder knowledge. Let*

$St' = ss \,\&\, ik$. If there is an initial state St_{ini} and a substitution σ such that $St \leadsto^*_{\sigma, R^{-1}_{B\mathcal{P}}, E_{\mathcal{P}}} St_{ini}$, then there is an initial state St'_{ini} and two substitutions σ', ρ such that $St' \leadsto^*_{\sigma', \overline{R_{B\mathcal{P}}}^{-1}, E_{\mathcal{P}}} St'_{ini}$, $\sigma =_{E_{\mathcal{P}}} \sigma' \circ \rho$, and $(St'_{ini})\rho \subseteq St_{ini}$.

Proof. We obtain the narrowing sequence 1) $St' \leadsto^*_{\sigma', \overline{R_{B\mathcal{P}}}^{-1}, E_{\mathcal{P}}} St'_{ini}$ from 2) $St \leadsto^*_{\sigma, R^{-1}_{B\mathcal{P}}, E_{\mathcal{P}}} St_{ini}$ by removing $SS\sigma$ and $IK\sigma$ from every state in 2) and then deleting the transitions that become trivial. We then check that this results in a sequence resulting from application of the rules $\overline{R_{B\mathcal{P}}}$. This is straightforward, except that we need to check that any conditions required by the new Rule (4.1) are fulfilled. The only such condition is that the intruder's knowledge be a set of intruder facts without repeated elements, i.e., the union operator `_,_` is $ACUI$ (associative-commutative-identity-idempotent). This follows directly from the restriction in [Escobar et al., 2006] that the intruder learns a term only once. □

4.4.2 Partial Order Reduction Giving Priority to Input Messages

The different rewrite rules on which the backwards narrowing search from an attack pattern is based are in general executed non-deterministically. This is because the order of execution can make a difference as to what subsequent rules can be executed. For example, an intruder cannot receive a term until it is sent by somebody, and that send action within a strand may depend upon other receives in the past. There is one major exception, Rule (4.1) (originally Rule (3.1)), which, in a backwards search, only moves a negative term appearing right before the bar into the intruder's knowledge.

Example 4.9 Consider, for instance, the attack pattern (†) in Page 70 Since the strand in the attack pattern has the input message $-(e(exp(E', n(B, r')), SR))$ but also has the intruder challenge $SR{\in}\mathcal{I}$ there are several possible backwards narrowing steps: some processing the intruder challenge, and Rule (4.1) processing the input message. ■

The execution of Rule (4.1) in a backwards search does not disable any other transitions; indeed, it only enables send transitions. Thus, it is safe to execute it at each stage *before* any other transition. For the same reason, if several applications of Rule (4.1) are possible, it is safe to execute them all at once before any other transition. Requiring all executions of Rule (4.1) to execute first thus eliminates interleavings of Rule (4.1) with send and receive transitions, which are equivalent to the case in which Rule (4.1) executes first. In practice, this typically cuts down in half the search space size. The completeness proof for this optimization is trivial and thus omitted.

Similar strategies have been employed by other tools in forward searches. For example, in [Shmatikov and Stern, 1998] a strategy is introduced that always executes send transitions first whenever they are enabled. Since a send transition does not depend on any other component of the state in order to take place, it can safely be executed first. The original NPA also used this strategy; it had a receive transition (similar to the input message in Maude-NPA) which had the effect of adding new terms to the intruder's knowledge, and which always was executed before any other transition once it was enabled.

4.4.3 Subsumption Partial Order Reduction

Partial order reduction (POR) techniques are common in state exploration. However, POR techniques for narrowing-based state exploration do not seem to have been explored in detail, although they may be extremely relevant and may afford greater reductions than in standard state exploration based on ground terms rather than on terms with logical variables. For instance, the simple concept of two states being equivalent modulo renaming of variables does not apply to standard state exploration, whereas it does apply to narrowing-based state exploration. In [Escobar and Meseguer, 2007], the authors studied narrowing-based state exploration and POR techniques, which may transform an infinite-state system into a finite one. However, the Maude-NPA needs a dedicated POR technique applicable to its specific execution model.

Let us motivate this POR technique with an example before giving a more detailed explanation.

Example 4.10 Consider again the attack pattern (†) in Page 70

$$:: r' :: [-(A;B;E'), +(B;A;exp(g,n(B,r'))),\\ -(e(exp(E',n(B,r')),SR)) \mid nil] \ \&\ SS\ \&\ \{SR{\in}\mathcal{I}\} \quad (\dagger)$$

After a couple of backwards narrowing steps, the Maude-NPA finds the state (‡) of Example 4.4 in Page 72:

$$[\ nil \mid\ -(exp(E',n(B,r'))), -(SR),\\ +(e(exp(E',n(B,r')),SR))\ ]\ \&\\ :: r' ::\\ [\ -(A;B;E'), +(B;A;exp(g,n(B,r'))) \mid\\ -(e(exp(E',n(B,r')),SR))\ ]\ \&\\ (SR{\in}\mathcal{I},\ exp(E',n(B,r')){\in}\mathcal{I},\ e(exp(E',n(B,r')),SR){\notin}\mathcal{I}) \quad (\ddagger)$$

However, the following state is also generated after a couple of narrowing steps from the attack pattern, where, thanks to the equational theory, variable E' is instantiated to $exp(G,N)$ for G a generator –indeed the constant g— and N a nonce variable:

$$[\ nil \mid -(exp(G,n(B,r'))), -(N), +(exp(G,N*n(B,r')))\ ]\ \&\\ [\ nil \mid -(exp(G,N*n(B,r'))), -(SR), +(e(exp(G,N*n(B,r')),SR))\ ]\ \&\\ :: r' :: [\ -(A;B;exp(G,N)), +(B;A;exp(g,n(B,r')))\\ \mid\ -(e(exp(G,N*n(B,r')),SR))\ ]\ \&\\ (SR{\in}\mathcal{I},\ exp(G,n(B,r')){\in}\mathcal{I},\ N{\in}\mathcal{I},\\ exp(G,N*n(B,r')){\notin}\mathcal{I},\ e(exp(G,N*n(B,r')),SR){\notin}\mathcal{I})$$

However, the unreachability of the second state is implied (modulo $E_{\mathcal{P}}$) by the unreachability of the first state; unreachability in the sense of Definition 4.2. Intuitively, the challenges present in the first state that are relevant for backwards reachability are included in the second state, namely, the challenges $SR{\in}\mathcal{I}$ and $exp(E',n(B,r')){\in}\mathcal{I}$. Indeed, the unreachability of the following "kernel" state implies the unreachability of both states, although this kernel state is never computed by the Maude-NPA:

$$:: r' ::$$
$$[\ -(A;B;E'), +(B;A;exp(g,n(B,r'))) \mid -(e(exp(E',n(B,r')),SR))\ ]\ \&$$
$$(SR{\in}\mathcal{I},\ exp(E',n(B,r')){\in}\mathcal{I}$$

Note that the converse is not true, i.e., the second state does not imply the first one, since it contains one more intruder item relevant for backwards reachability purposes, namely $N{\in}\mathcal{I}$. ■

Let us now formalize this state space reduction and prove its completeness.

Definition 4.11 *Given a topmost rewrite theory $\mathcal{R}_{\mathcal{P}} = (\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, R_{B\mathcal{P}})$ representing protocol $\mathcal{P}$, and two non-initial states St_1 and St_2, we write $St_1 \rhd St_2$ (or $St_2 \lhd St_1$) if each intruder fact of the form $t{\in}\mathcal{I}$ in St_1 appears in St_2 (modulo $E_{\mathcal{P}}$) and each non-initial strand in St_1 appears in St_2 (modulo $E_{\mathcal{P}}$ and with the vertical bar at the same position).*

This is similar to the relation $\subseteq$ given in Definition 4.7 in Section 4.4.1, except for the condition that the two states be non-initial. This condition is imposed because, otherwise, an initial state will imply any other state, erroneously making the search space finite after an initial state has been found.

We define the relation $St_1 \blacktriangleright St_2$ which extends $St_1 \rhd St_2$ to the case where St_1 is more general than St_2 w.r.t. variable instantiation.

Definition 4.12 ($\mathcal{P}$-subsumption relation) *Given a topmost rewrite theory $\mathcal{R}_{\mathcal{P}} = (\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, R_{B\mathcal{P}})$ representing protocol $\mathcal{P}$ and two non-initial states St_1, St_2, we write $St_1 \blacktriangleright St_2$ (or $St_2 \blacktriangleleft St_1$) and say that St_2 is* $\mathcal{P}$-subsumed *by St_1 if there is a substitution θ s.t. $St_1\theta \rhd St_2$.*

We now show that, if $St_1 \blacktriangleright St_2$, then St_2 can be discarded without sacrificing completeness. We do this by showing how every path from St_2 to an initial state can be used to construct a path from St_1 to an initial state, so that unreachability of St_1 implies unreachability of St_2.

We first show that if $St_1 \blacktriangleright St_2$, i.e. there is a θ such that $St_1\theta \rhd St_2$, and a narrowing step $St_2 \leadsto_{\sigma_2, R^{-1}_{B\mathcal{P}}, E_{\mathcal{P}}} St'_2$ for some state St'_2, then

either $(St_1\theta)\sigma_2 \rhd St_2'$ (and so $St_1 \blacktriangleright St_2'$), or there is a narrowing step $St_1 \leadsto_{\sigma_1, R_{B\mathcal{P}}^{-1}, E_{\mathcal{P}}} St_1'$ for some state St_1' that is either initial or for which there is a substitution ρ such that $St_1'\rho \rhd St_2'$ (and so $St_1' \blacktriangleright St_2'$). Once we have proven these results, we can then use induction to show that, if there is a path from St_2 to an initial state, there is a path from St_1 to an initial state, and we are done.

The following results provide the appropriate connection between $\mathcal{P}$-subsumption and narrowing transitions. First, we make use of the following lemma, whose proof follows directly from the definition of $\rhd$.

Lemma 4.13 *Suppose that we have a topmost rewrite theory $\mathcal{R}_{\mathcal{P}} = (\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, R_{B\mathcal{P}})$ representing protocol $\mathcal{P}$ and two non-initial states St_1, St_2 and substitution θ such that $St_1\theta \rhd St_2$, i.e., $St_1 \blacktriangleright St_2$. If there is a narrowing step $St_2 \leadsto_{\sigma_2, R_{B\mathcal{P}}^{-1}, E_{\mathcal{P}}} St_2'$ where St_2' is non-initial such that $(St_1\theta)\sigma_2 \not\blacktriangleright St_2'$, then either (a) there is an intruder fact of the form $t{\in}\mathcal{I}$ in $(St_1\theta)\sigma_2$ that does not appear in St_2' (modulo $E_{\mathcal{P}}$), or (b) there is a non-initial strand in $(St_1\theta)\sigma_2$ that does not appear in St_2' (modulo $E_{\mathcal{P}}$). In particular, if conditions (a) and (b) are not satisfied, then $St_1 \blacktriangleright St_2'$.*

Suppose now that $(St_1\theta)\sigma_2 \not\blacktriangleright St_2'$ so that we consider both cases of Lemma 4.13 separately: either an expression $t{\in}\mathcal{I}$ in St_2' or a non-initial strand in $(St_1\theta)\sigma_2$, not appearing in St_2'. First, the case where an expression $t{\in}\mathcal{I}$ in $(St_1\theta)\sigma_2$ does not appear in St_2'.

Lemma 4.14 *Suppose that we have a topmost rewrite theory $\mathcal{R}_{\mathcal{P}} = (\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, R_{B\mathcal{P}})$ representing protocol $\mathcal{P}$ and two non-initial states St_1, St_2. If (i) there is a substitution θ s.t. $St_1\theta \rhd St_2$, i.e., $St_1 \blacktriangleright St_2'$, (ii) there is a narrowing step $St_2 \leadsto_{\sigma_2, R_{B\mathcal{P}}^{-1}, E_{\mathcal{P}}} St_2'$, and (iii) there is an intruder fact of the form $t{\in}\mathcal{I}$ in $(St_1\theta)\sigma_2$ that does not appear in St_2' (modulo $E_{\mathcal{P}}$), then (a) $t{\notin}\mathcal{I}$ does appear in St_2' (modulo $E_{\mathcal{P}}$) and (b) there is a state St_1' and a substitution σ_1 such that $St_1 \leadsto_{\sigma_1, R_{B\mathcal{P}}^{-1}, E_{\mathcal{P}}} St_1'$ and either St_1' is an initial state or there is a substitution ρ s.t. $St_1'\rho \rhd St_2'$, i.e., $St_1' \blacktriangleright St_2'$,*

Proof. We prove the result by considering the different rules applicable to St_2 (remember that in $\mathcal{R}$, rewriting and narrowing steps always happen at the top position). Note that property (a) is immediate because rules in $R_{B\mathcal{P}}$ do not remove expressions of the form $m{\in}\mathcal{I}$. Note also that if $t{\in}\mathcal{I}$

does appear in St_2 (modulo $E_{\mathcal{P}}$) and $t \notin \mathcal{I}$ does appear in St'_2 (modulo $E_{\mathcal{P}}$), then only Rule (3.3) or rules of type (3.4) have been applied to St_2 as follows:

- Reversed version of Rule (3.3), i.e., $St_2 \rightsquigarrow_{\sigma_2, R^{-1}_{B^{\mathcal{P}}}, E_{\mathcal{P}}} St'_2$ using the following rule

  $$[L, M^+ \mid L'] \,\&\, SS \,\&\, (M \in \mathcal{I}, IK) \rightarrow [L \mid M^+, L'] \,\&\, SS \,\&\, (M \notin \mathcal{I}, IK).$$

  Recall that there is an intruder fact in $(St_1\theta)\sigma_2$ of the form $t \in \mathcal{I}$ for t a message term that does not appear in St'_2 (modulo $E_{\mathcal{P}}$) and $t =_{E_{\mathcal{P}}} M\sigma_2$. Thus, $M\sigma_2 \in \mathcal{I}$ does appear in $(St_1\theta)\sigma_2$ (modulo $E_{\mathcal{P}}$). Here we have several cases:

  - If the strand $([L, M^+ \mid L'])\sigma_2$ appears in $(St_1\theta)\sigma_2$, then the very same narrowing step can be performed on St_1, i.e., there exist σ_1, ρ such that $St_1 \rightsquigarrow_{\sigma_1, R^{-1}_{B^{\mathcal{P}}}, E_{\mathcal{P}}} St'_1$ with the same rule and $\theta \circ \sigma_2 =_{E_{\mathcal{P}}} \sigma_1 \circ \rho$. Thus, either St'_1 is an initial state or $St'_1\rho \rhd St'_2$, since: (i) each positive intruder fact in $(St_1\theta)\sigma_2$ of the form $u \in \mathcal{I}$ for u a message term, except $M\sigma_2 \in \mathcal{I}$, appears in $St'_1\rho$ (modulo $E_{\mathcal{P}}$), (ii) $M\sigma_2 \notin \mathcal{I}$ appears in $St'_1\rho$ (modulo $E_{\mathcal{P}}$), (iii) each non-initial strand in $(St_1\theta)\sigma_2$, except $[L, M^+ \mid L']\sigma_2$, has not been modified and appears in $St'_1\rho$ as well (modulo $E_{\mathcal{P}}$), and (iv) for $[L, M^+ \mid L']\sigma_2$ in $(St_1\theta)\sigma_2$, $[L \mid M^+, L']\rho'$ appears in $St'_1\rho$ and in St'_2.
  - If the strand $[L_m, M^+ \mid L']\sigma_2$ does not appear in $(St_1\theta)\sigma_2$, then the strand $[L, M^+ \mid L']\sigma_2$ corresponds to a strand $\mathcal{S}_{\mathcal{P}}$ in the protocol specification that had been introduced via a rule of the set (3.4), where the strand's bar was clearly more to the right than in $[L, M^+ \mid L']\sigma_2$. Note that it cannot correspond to a strand included originally in the attack pattern, because we assume that St_1 and St_2 are states generated by backwards narrowing from the same attack state and then both St_1 and St_2 should have the strand. Therefore, since the strand $[L, M^+ \mid L']\sigma_2$ corresponds to a strand in $\mathcal{S}_{\mathcal{P}}$ and the set (3.4) contains a rewrite rule for each strand of the form $[\, l_1,\ u^+,\ l_2\,]$ in $\mathcal{S}_{\mathcal{P}}$, there must be a rule α in (3.4) introducing

a strand of the form $[\ l_1,\ u^+,\ l_2\ ]$ and there must be substitutions σ_1, ρ such that $St_1 \leadsto_{\sigma_1, R_{B\mathcal{P}}^{-1}, E_\mathcal{P}} St_1'$ using the rule α and $\theta \circ \sigma_2 =_{E_\mathcal{P}} \sigma_1 \circ \rho$. Thus, either St_1' is an initial state or $St_1'\rho \rhd St_2'$, since: (i) each positive intruder fact in $(St_1\theta)\sigma_2$ of the form $u{\in}\mathcal{I}$ for u a message term, except $M\sigma_2{\in}\mathcal{I}$, appears in $St_1'\rho$ (modulo $E_\mathcal{P}$), (ii) $M\sigma_2{\notin}\mathcal{I}$ appears in $St_1'\rho$ (modulo $E_\mathcal{P}$), (iii) each non-initial strand in $(St_1\theta)\sigma_2$ has not been modified and appears in $St_1'\rho$ as well (modulo $E_\mathcal{P}$), and (iv) $[\ l_1 \mid u^+,\ l_2\ ]\sigma_2$ appears in $St_1'\rho$ and in St_2'.

- Rules in (3.4), i.e., $St_2 \leadsto_{\sigma_2, R_{B\mathcal{P}}^{-1}, E_\mathcal{P}} St_2'$ using a rule of the form

$$\{SS\,\&\,(u{\in}\mathcal{I}, IK) \rightarrow [l_1 \mid u^+, l_2]\,\&\,SS\,\&\,(u{\notin}\mathcal{I}, IK) \mid [l_1, u^+, l_2] \in \mathcal{P}\}.$$

  Recall that there is an intruder fact in $(St_1\theta)\sigma_2$ of the form $t{\in}\mathcal{I}$ for t a message term that does not appear in St_2' (modulo $E_\mathcal{P}$) and $t =_{E_\mathcal{P}} u\sigma_2$, where u is the message term used by the rewrite rule. Thus, $u\sigma_2{\in}\mathcal{I}$ does appear in $(St_1\theta)\sigma_2$ (modulo $E_\mathcal{P}$). That is, the same narrowing step is available from $(St_1\theta)\sigma_2$ and there exist σ_1, ρ such that $St_1 \leadsto_{\sigma_1, R_{B\mathcal{P}}^{-1}, E_\mathcal{P}} St_1'$ with the same rule and $\theta \circ \sigma_2 =_{E_\mathcal{P}} \sigma_1 \circ \rho$. Thus, either St_1' is an initial state or $St_1'\rho \rhd St_2'$.

This concludes the proof. □

Second, we examine the case in which a non-initial strand in St_2' does not appear in $(St_1\theta)\sigma_2$.

Lemma 4.15 *Given a topmost rewrite theory $\mathcal{R}_\mathcal{P} = (\Sigma_\mathcal{P}, E_\mathcal{P}, R_{B\mathcal{P}})$ representing protocol $\mathcal{P}$ and two non-initial states St_1, St_2. If (i) there is a substitution θ s.t. $St_1\theta \rhd St_2$, (ii) there is a narrowing step $St_2 \leadsto_{\sigma_2, R_{B\mathcal{P}}^{-1}, E_\mathcal{P}} St_2'$, and (iii) there is a non-initial strand $[m_1^\pm, \ldots, m_i^\pm \mid m_{i+1}^\pm, \ldots, m_n^\pm]$ in $(St_1\theta)\sigma_2$ that does not appear in St_2' (modulo $E_\mathcal{P}$), then (a) $\sigma_2|_{\mathcal{V}ar(St_2)} = id$, (b) $[m_1^\pm, \ldots, m_{i-1}^\pm \mid m_i^\pm, \ldots, m_n^\pm]$ does appear in St_2' (modulo $E_\mathcal{P}$) and (c) there is a state St_1' such that $St_1 \leadsto_{id, R_{B\mathcal{P}}^{-1}, E_\mathcal{P}} St_1'$ and either St_1' is an initial state or $St_1' \rhd St_2'$.*

Proof. We prove the result by considering the different rules applicable to St_2 (remember that in $\mathcal{R}$, rewriting and narrowing steps always

happen at the top position). Note that property (a) is immediate because rules in $R_{B\mathcal{P}}$ do not remove strands, only move the vertical bar to the left of the sequences of messages in the strands. Note also that if $[m_1^{\pm}, \ldots, m_i^{\pm} \mid m_{i+1}^{\pm}, \ldots, m_n^{\pm}]$ appears in $(St_1\theta)\sigma_2$ and $[m_1^{\pm}, \ldots, m_{i-1}^{\pm} \mid m_i^{\pm}, \ldots, m_n^{\pm}]$ appears in St_2', then only Rule (3.2) or Rule (4.1) have been applied to St_2 as follows:

- Reversed version of Rule (3.2), i.e., $St_2 \rightsquigarrow_{\sigma_2, R_{B\mathcal{P}}^{-1}, E_{\mathcal{P}}} St_2'$ using the following rule

$$[L, M^+ \mid L'] \,\&\, SS \,\&\, IK \rightarrow [L \mid M^+, L'] \,\&\, SS \,\&\, IK.$$

- Reversed version of Rule (4.1), i.e., $St_2 \rightsquigarrow_{\sigma_2, R_{B\mathcal{P}}^{-1}, E_{\mathcal{P}}} St_2'$ using the following rule

$$[L, M^- \mid L'] \,\&\, SS \,\&\, IK \rightarrow [L \mid M^-, L'] \,\&\, SS \,\&\, (M{\in}\mathcal{I}, IK).$$

Note, however, that $\sigma_2|_{\mathcal{V}ar(St_2)} = id$ in both possible rewrite steps. Then, there is a state St_1' such that $St_1 \rightsquigarrow_{id, R_{B\mathcal{P}}^{-1}, E_{\mathcal{P}}} St_1'$ with the same rule and it is straightforward that either St_1' is an initial state or $St_1' \triangleright St_2'$, since only the vertical bar has been moved. □

Now we can formally define the relation between $\mathcal{P}$-subsumption and one narrowing step. In the following, $\rightsquigarrow^{\{0,1\}}_{\sigma, R_{B\mathcal{P}}^{-1}, E_{\mathcal{P}}}$ denotes zero or one narrowing steps.

Lemma 4.16 *Given a topmost rewrite theory $\mathcal{R}_{\mathcal{P}} = (\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, R_{B\mathcal{P}})$ representing protocol $\mathcal{P}$ and two non-initial states St_1, St_2. If $St_1 \triangleright St_2$ and $St_2 \rightsquigarrow_{\sigma_2, R_{B\mathcal{P}}^{-1}, E_{\mathcal{P}}} St_2'$, then there is a state St_1' and a substitution σ_1 such that $St_1 \rightsquigarrow^{\{0,1\}}_{\sigma_1, R_{B\mathcal{P}}^{-1}, E_{\mathcal{P}}} St_1'$ and either St_1' is an initial state or $St_1' \triangleright St_2'$.*

Proof. Since $St_1 \triangleright St_2$, there is a substitution θ s.t. $St_1\theta \rhd St_2$. If each intruder fact of the form $t{\in}\mathcal{I}$ in $(St_1\theta)\sigma_2$ appears in St_2' (modulo $E_{\mathcal{P}}$) and each non-initial strand in $(St_1\theta)\sigma_2$ appears in St_2' (modulo $E_{\mathcal{P}}$), then, by Lemma 4.13, $(St_1\theta)\sigma_2 \rhd St_2'$, i.e., $St_1 \triangleright St_2'$. Otherwise, Lemma 4.13 states that either (a) there is an intruder fact of the form $t{\in}\mathcal{I}$ in $(St_1\theta)\sigma_2$ that does not appear in St_2' (modulo $E_{\mathcal{P}}$), or (b) there is a non-initial

strand in $(St_1\theta)\sigma_2$ that does not appear in St'_2 (modulo $E_\mathcal{P}$). For case (a), by Lemma 4.14, there is a state St'_1 and a substitution σ_1 such that $St_1 \rightsquigarrow_{\sigma_1, R^{-1}_{B\mathcal{P}}, E_\mathcal{P}} St'_1$ and either St'_1 is an initial state or there is a substitution ρ s.t. $St'_1\rho \rhd St'_2$. For case (b), by Lemma 4.15, $\sigma_2|_{\mathcal{V}ar(St_2)} = id$, and there is a state St'_1 such that $St_1 \rightsquigarrow_{id, R^{-1}_{B\mathcal{P}}, E_\mathcal{P}} St'_1$ and either St'_1 is an initial state or $St'_1 \rhd St'_2$, i.e., $St'_1 \blacktriangleright St'_2$. □

Preservation of reachability follows from the following main theorem. Note that the relation $\blacktriangleright$ is applicable only to non-initial states, whereas the relation $\subseteq_{E_\mathcal{P}}$ of Definition 4.7 is applicable to both initial and non-initial states.

Theorem 4.17 *Given a topmost rewrite theory $\mathcal{R}_\mathcal{P} = (\Sigma_\mathcal{P}, E_\mathcal{P}, R_{B\mathcal{P}})$ representing protocol $\mathcal{P}$ and two states St_1, St_2. If $St_1 \blacktriangleright St_2$, St_2^{ini} is an initial state, and $St_2 \rightsquigarrow^*_{\sigma_2, R^{-1}_{B\mathcal{P}}, E_\mathcal{P}} St_2^{ini}$, then there is an initial state St_1^{ini} and substitutions σ_1 and θ such that $St_1 \rightsquigarrow^*_{\sigma_1, R^{-1}_{B\mathcal{P}}, E_\mathcal{P}} St_1^{ini}$, and $St_1^{ini}\theta \subseteq_{E_\mathcal{P}} St_2^{ini}$.*

Proof. Consider $St_2 = U_0$, $St_2^{ini} = U_n$, $\sigma_2 = \rho_1 \cdots \rho_n$, and $U_0 \rightsquigarrow^n_{\rho_i, R^{-1}_{B\mathcal{P}}, E_\mathcal{P}} U_n$. Note that $n \neq 0$, since St_2 cannot be an initial state because $St_1 \blacktriangleright St_2$ implies that both St_1 and St_2 are not initial states. Then, by Lemma 4.16, there is $j \leqslant n$ such that for each $i < j$, $U_{i-1} \rightsquigarrow_{\rho_i, R^{-1}_{B\mathcal{P}}, E_\mathcal{P}} U_i$ and there is a step $U'_{i-1} \rightsquigarrow_{\rho'_i, R^{-1}_{B\mathcal{P}}, E_\mathcal{P}} U'_i$ s.t. $U'_i \blacktriangleright U_i$. Note that U'_j is an initial state and there is a substitution θ s.t. $U'_j\theta \subseteq_{E_\mathcal{P}} U_j \subseteq_{E_\mathcal{P}} U_n$. □

This POR technique is used as follows: we keep all the states of the backwards narrowing-based tree and compare each new node of the tree produced by the narrowing algorithm with all the states in the tree that have already been produced. If a node is $\mathcal{P}$-subsumed by a previously generated node in the tree, we discard the subsumed node.

4.5 The Super-Lazy Intruder

Sometimes terms appear in the intruder's knowledge that are trivially learnable by the intruder. These include terms initially available to the intruder (such as names) and variables. In the case of variables, specially,

the intruder can substitute any arbitrary term of the same sort as the variable,[1] and so there is no need to try to determine all the ways in which the intruder can do this. For this reason it is safe to temporarily drop these terms from the state. We will refer to those terms as *(super) lazy intruder* terms, after the name *lazy intruder* coined by Basin et al. [2005] to describe another optimization technique that involves delaying instantiation of variables.

To see how super-lazy terms arise, we consider the following example.

Example 4.18 Consider again the attack pattern (†) in Page 70.

$$:: r'' :: [-(a;b;Y), +(b;a;exp(g,n(b,r''))), \\ -(e(exp(Y,n(b,r'')),SR)) \mid nil] \ \& \ SS \ \& \ \{SR{\in}\mathcal{I}\} \quad (\dagger)$$

After a couple of backwards narrowing steps, the Maude-NPA finds the following state that describes how the intruder can learn SR by assuming he can learn a message $e(K, SR)$ and the key K:

$$[\ nil \mid -(K),\ -(e(K,SR)),\ +(SR)\ ] \ \& \\ :: r' :: \\ [\ -(A;B;E'), +(B;A;exp(g,n(B,r'))) \mid \\ \ -(e(exp(E',n(B,r')),SR))\ ] \ \& \\ (e(exp(E',n(B,r')),SR){\in}\mathcal{I},\ K{\in}\mathcal{I},\ e(K,SR)){\in}\mathcal{I}, SR{\notin}\mathcal{I}) \quad (\natural)$$

Here variable K is a super-lazy term. The intruder can find it by instantiating it by any term of the sort for keys from its initial knowledge, so we drop $K{\in}\mathcal{I}$ from the state description. ■

Dropping super-lazy terms is complete by Theorem 4.17; but if we drop them permanently we lose soundness. If the variables used in creating those terms appear elsewhere in the state, they may become instantiated as the backwards search continues. In that case, the super-lazy

[1] This, of course, is subject to the assumption that the intruder can produce at least one term of that sort. But since the intruder is assumed to have access to the network and to all the operations available to an honest principal, this is a reasonable restriction to make.

terms that were deleted may no longer be trivial to find. This may result in the construction of narrowing sequences from the state that has the super-lazy terms removed to an initial state, that does not correspond to any narrowing sequence that could be obtained if the terms had been retained.

Example 4.19 Consider the state ($\natural$) described in Example 4.18. After some more backwards narrowing steps, the tool unifies message $e(K, SR)$ with an output message $e(exp(\overline{X}, n(\overline{A}, \overline{r_1})), sec(\overline{A}, \overline{r_2}))$ of an explicitly added Alice's strand of the form:

$$:: \overline{r_1}, \overline{r_2} ::$$
$$[\ +(\overline{A}; \overline{B}; exp(g, n(\overline{A}, \overline{r_1}))),\ -(\overline{B}; \overline{A}; \overline{X}), +(e(exp(\overline{X}, n(\overline{A}, \overline{r_1})), sec(\overline{A}, \overline{r_2})))]$$

thus getting an instantiation for the super-lazy term K, namely $\{K \mapsto exp(\overline{X}, n(a, \overline{r_1}))\}$. That is, obtaining the following state by some backwards narrowing steps from the state ($\natural$) of Example 4.18:

$$\begin{array}{l}
[\ nil \mid -(exp(\overline{X}, n(a, \overline{r_1}))),\ -(e(exp(\overline{X}, n(a, \overline{r_1})), sec(a, \overline{r_2}))), \\
\qquad +(sec(a, \overline{r_2}))\ ]\ \& \\
:: r' :: \\
[\ -(A; B; E'), +(B; A; exp(g, n(B, r'))) \mid \\
\quad -(e(exp(E', n(B, r')), sec(a, \overline{r_2})))\ ]\ \& \\
:: \overline{r_1}, \overline{r_2} :: \\
[\ +(a; \overline{B}; exp(g, n(a, \overline{r_1}))),\ -(\overline{B}; a; \overline{X}) \mid \\
\quad +(e(exp(\overline{X}, n(a, \overline{r_1})), sec(a, \overline{r_2})))]\ \& \\
(e(exp(E', n(B, r')), sec(a, \overline{r_2})) \in \mathcal{I},\ exp(\overline{X}, n(a, \overline{r_1})) \in \mathcal{I}, \\
(e(exp(\overline{X}, n(a, \overline{r_1})), sec(a, \overline{r_2}))) \notin \mathcal{I},\ sec(a, \overline{r_2}) \notin \mathcal{I})
\end{array} \qquad (\natural_+)$$

Now the intruder can no longer construct K out of terms in its initial knowledge, because $n(\overline{a}, \overline{r_1})$ is not in its initial knowledge. ■

Since we intend Maude-NPA to be both sound and complete, we elect to remove super-lazy terms only temporarily. When super-lazy terms are deleted from a state, a copy of the original state known as a *ghost*

state is retained. The variables in the super-lazy terms are monitored to determine whether or not they become instantiated during a narrowing step. If that is the case, the state resulting from this narrowing step is deleted and replaced with the ghost state with the super-lazy terms instantiated.

The operation of *resuscitating* the ghost state is complex; in particular care must be taken to avoid interaction with subsumption partial order reduction. Removing super-lazy terms from a state affects its status in the subsumption partial order, and it is again affected when a ghost is resuscitated. The result is that, if we wish to allow states with ghosts to participate in the subsumption partial order reduction, we must proceed very carefully. In particular, if we apply the subsumption partial order reduction indiscriminately, a resuscitated ghost state will be dominated in the partial order by the ancestor that introduced the ghost, and so will be removed. Thus, we have implemented a procedure for identifying the ancestor of a resuscitated ghost state when checking the subsumption partial order. See Sections 4.5.4 and 4.5.5.

The remainder of this section is organized as follows. In Section 4.5.1 we give a formal definition of super-lazy terms. In Section 4.5.2 we describe the procedure for creating and resuscitating ghost states. In Section 4.5.3 we describe an optimization of the super-lazy intruder that allows one to identify cases in which super-lazy terms will never be further instantiated, and thus can be safely removed without creating a ghost state. Finally, in Sections 4.5.4 and 4.5.5 we describe how the potentially harmful interaction between the subsumption partial order and the super-lazy intruder is handled.

4.5.1 Definition of Super-Lazy Terms

The set $\mathcal{L}(St)$ of super-lazy terms w.r.t. a state St is inductively generated as a subset $\mathcal{L}(St) \subseteq \mathcal{T}_\Omega(Y \cup IK_0)$ where IK_0 is the basic set of terms known by the intruder at the beginning of a protocol execution, Y is a subset of the variables of St, and Ω is the set of operations available to the intruder. The idea of super-lazy terms is that we also want to exclude from $\mathcal{L}(St)$ the set $IK^{\notin}(St)$ of terms that the intruder does not know and all its possible combinations with symbols in Ω.

Definition 4.20 (Super-lazy terms) *Let $\mathcal{R}_{\mathcal{P}} = (\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, R_{B\mathcal{P}})$ be a topmost rewrite theory representing protocol $\mathcal{P}$. Let IK_0 be the basic set of terms known by the intruder at the beginning of a protocol execution, defined as $IK_0 = \{t' \mid [t^+] \in \mathcal{S}_{\mathcal{P}},\ t' =_{E_{\mathcal{P}}} t\}$ Let Ω be the set of operations available to the intruder, defined indirectly as follows:*

$$\Omega = \{f : \mathsf{s_1} \cdots \mathsf{s_n} \to \mathsf{s} \mid [(X_1{:}\mathsf{s_1})^-, \ldots, (X_k{:}\mathsf{s_k})^-, (f(X_1{:}\mathsf{s_1}, \ldots, X_k{:}\mathsf{s_k}))^+] \in \mathcal{S}_{\mathcal{P}}\}.$$

Let St be a state (with logical variables). Let $IK^{\notin}(St)$ be the set of terms that the intruder does not known at state St, defined as $IK^{\notin}(St) = \{m' \mid (m{\notin}\mathcal{I}) \in St,\ m' =_{E_{\mathcal{P}}} m\}$. The set $\mathcal{L}(St)$ of super-lazy terms w.r.t. *St (or simply super-lazy terms) is defined inductively as follows:*

1. *$IK_0 \subseteq \mathcal{L}(St)$,*
2. *$\mathcal{V}ar(St) - IK^{\notin}(St) \subseteq \mathcal{L}(St)$,*
3. *for each $f : \mathsf{s_1} \cdots \mathsf{s_n} \to \mathsf{s} \in \Omega$ and for all $t_1{:}\mathsf{s_1}, \ldots, t_k{:}\mathsf{s_k} \in \mathcal{L}(St)$, if $f(t_1{:}\mathsf{s_1}, \ldots, t_k{:}\mathsf{s_k}) \notin IK^{\notin}(St)$, then $f(t_1{:}\mathsf{s_1}, \ldots, t_k{:}\mathsf{s_k}) \in \mathcal{L}(St)$.*

The idea behind the super-lazy intruder is that, given a term made out of lazy intruder terms, such as "$a; e(K, Y)$", where a is a public name and K and Y are variables, the term "$a; e(K, Y)$" is also a (super) lazy intruder term by applying the public operations e and $\_;\_$ available to the intruder.

4.5.2 The Super-Lazy Intruder and Ghost States

Let us first briefly explain how the ghost state mechanism works before formally describing it. A *ghost state* is a state extended to allow expressions of the form ghost(m) in the intruder's knowledge, where m is a super-lazy term. When, during the backwards reachability analysis, we detect a state St having a super lazy term t in an expression $t{\in}\mathcal{I}$ in the intruder's knowledge, we replace the intruder fact $t{\in}\mathcal{I}$ in St by $ghost(t)$ and keep the ghost version of St in the history of states used by the transition subsumption of Section 4.4.3.

Example 4.21 The state (♮) of Example 4.18 with a super-lazy intruder term K would be represented as follows, where we have just replaced $K{\in}\mathcal{I}$ by $ghost(K)$:

$$
\begin{array}{l}
[\ nil \mid -(K),\ -(e(K,SR)),\ +(SR)\ ]\ \&\\
::\ r'\ ::\\
[\ -(A;B;E'), +(B;A;exp(g,n(B,r'))) \mid \\
\quad -(e(exp(E',n(B,r')),SR))\ ]\ \&\\
(ghost(K),\ e(exp(E',n(B,r')),SR)\in\mathcal{I},\\
e(K,SR))\in\mathcal{I},\ SR\notin\mathcal{I})
\end{array}
\qquad (\bar{\natural})
$$

Similarly, the state ($\natural_+$) of Example 4.19 would be represented as follows, where we have just added the expression $ghost(exp(\overline{X}, n(a,\overline{r_1})))$ to the intruder knowledge:

$$
\begin{array}{l}
[\ nil \mid -(exp(\overline{X},n(a,\overline{r_1}))),\ -(e(exp(\overline{X},n(a,\overline{r_1})),sec(a,\overline{r_2}))),\\
\qquad +(sec(a,\overline{r_2}))\ ]\ \&\\
::\ r'\ ::\\
[\ -(A;B;E'), +(B;A;exp(g,n(B,r'))) \mid \\
\quad -(e(exp(E',n(B,r')),sec(a,\overline{r_2})))\ ]\ \&\\
::\ \overline{r_1},\overline{r_2}\ ::\\
[\ +(a;\overline{B};exp(g,n(a,\overline{r_1}))),\ -(\overline{B};a;\overline{X}) \mid \\
\quad +(e(exp(\overline{X},n(a,\overline{r_1})),sec(a,\overline{r_2})))]\ \&\\
(ghost(exp(\overline{X},n(a,\overline{r_1}))),\ e(exp(E',n(B,r')),sec(a,\overline{r_2}))\in\mathcal{I},\\
exp(\overline{X},n(a,\overline{r_1}))\in\mathcal{I}, (e(exp(\overline{X},n(a,\overline{r_1})),sec(a,\overline{r_2})))\notin\mathcal{I},\ sec(a,\overline{r_2})\notin\mathcal{I})
\end{array}
\qquad (\overline{\natural_+})
$$

■

Suppose that later in the backwards search tree we find a descendant state St' in which $ghost(u)$ has been instantiated to $ghost(t)$, where t is not a super lazy intruder term as in the state ($\overline{\natural_+}$) of Example 4.21. For the intruder to learn such a term t, it may be necessary for certain actions to occur before St' was produced. That is, we must "roll back" and replace the current state St', containing expression $ghost(t)$, by an instantiated version of its ancestor state St, namely $St\theta$, where $t =_{E_\mathcal{P}} u\theta$. This is explained in detail in Definition 4.28 below.

A complication is introduced if the substitution θ binding variables in u includes variables of sort Fresh. These must have been introduced by strands indexed by these fresh variables. If the strand indexed by a fresh

variable in t already appears in St, then there is no problem. However, if the strand was introduced later in the backwards narrowing process, and we do not include them in St, then this will result in difficulties. If such a strand is not in the reactivated version of St, it will not be re-introduced in the backwards narrowing search, because the fresh variables in newly introduced strands are non-unifiable with any of the fresh variables already present. Therefore, the strands indexed by these fresh variables must also be included in the "rolled back" state, even if they were not there originally. Moreover, they must have the bar at the place where it was when the strands were originally introduced. We show below how this is accomplished. Furthermore, if any of the strands thus introduced have other variables of sort Fresh as subterms, then the strands indexed by those variables must be included too, and so on. That is, when a state St' properly instantiating a ghost expression $ghost(t)$ is found, the procedure of rolling back to the original state St that gave rise to that ghost expression implies not only applying the bindings for the variables of t to St, but also introducing in St all the strands from St' that produced fresh variables and that either appear in the variables of t or are recursively connected with them.

Example 4.22 Consider the states $(\overline{\natural})$ and $(\overline{\natural_+})$ of Example 4.21. After the tool finds an instantiation for variable K in the narrowing step from state $(\overline{\natural})$ to state $(\overline{\natural_+})$, the tool rolls back to the state $(\natural)$, originating the super-lazy term K, as follows; where we have transformed state $(\natural_+)$ by moving the vertical bar of Alice's strand at the rightmost position because it is the strand generating the Fresh variable $\overline{r_2}$:

$$
\begin{array}{l}
[\ nil \mid -(exp(\overline{X}, n(a, \overline{r_1}))),\ +(e(exp(\overline{X}, n(a, \overline{r_1})), sec(a, \overline{r_2}))), \\
\qquad +(sec(a, \overline{r_2}))\ ]\ \& \\
:: r' :: \\
[\ -(A; B; E'), +(B; A; exp(g, n(B, r'))) \mid \\
\ -(e(exp(E', n(B, r')), sec(a, \overline{r_2})))\ ]\ \& \\
:: \overline{r_1}, \overline{r_2} :: \\
[\ +(a; \overline{B}; exp(g, n(a, \overline{r_1}))),\ -(\overline{B}; a; \overline{X}), \\
\ +(e(exp(\overline{X}, n(a, \overline{r_1})), sec(a, \overline{r_2}))) \mid nil]\ \& \\
(e(exp(E', n(B, r')), sec(a, \overline{r_2})) \in \mathcal{I},\ exp(\overline{X}, n(a, \overline{r_1})) \in \mathcal{I}, \\
\ e(exp(\overline{X}, n(a, \overline{r_1})), sec(a, \overline{r_2})) \in \mathcal{I},\ sec(a, \overline{r_2}) \notin \mathcal{I})
\end{array}
\qquad (\natural^{\circlearrowleft})
$$

■

In order for the super-lazy intruder mechanism to be able to tell where the bar was when a strand was introduced, we must modify the set of rules of type (3.4) introducing new strands:

$$\{\, [\, l_1 \,|\, u^+ \,] \,\&\, \{u{\in}\mathcal{I}, K\} \rightarrow \{u{\in}\mathcal{I}, K\} \;\mid\; [\, l_1,\, u^+,\, l_2 \,] \in \mathcal{S}_{\mathcal{P}}\} \qquad (4.2)$$

Note that rules of type (3.4) introduce strands $[\, l_1 \mid u^+, l_2 \,]$, whereas here rules of type (4.2) introduce strands $[\, l_1 \mid u^+ \,]$. This slight modification makes it possible to safely move the position of the bar back to the place where the strand was introduced. However, now the strands added may be *partial*, since the whole sequence of actions performed by the principal is not directly recorded in the strand. Therefore, the set of rewrite rules used by narrowing in reverse are now $\widetilde{R_{B\mathcal{P}}} = \{(4.1), (3.2), (3.3)\} \cup \{(4.2)\}$; note that (4.2) represents a set of rules.

First, we define a new relation $\sqsubseteq_{E_{\mathcal{P}}}$ between states, which is similar to $\subseteq_{E_{\mathcal{P}}}$ of Definition 4.7 but considers partial strands.

Definition 4.23 (Partial Inclusion) *Given two states St_1, St_2, we abuse notation and write $St_1 \sqsubseteq_{E_{\mathcal{P}}} St_2$ to denote that every intruder fact in St_1 appears in St_2 (modulo $E_{\mathcal{P}}$) and that every strand $[m_1^{\pm}, \ldots, m_k^{\pm}]$ in St_1, either appears in St_2 (modulo $E_{\mathcal{P}}$) or there is $i \in \{1, \ldots, k\}$ s.t. $m_i^{\pm} = m_i^{+}$ and $[m_1^{\pm}, \ldots, m_i^{+}]$ appears in St_2 (modulo $E_{\mathcal{P}}$).*

The following result ensures that if a state is reachable via backwards reachability analysis using $R_{B\mathcal{P}}$, then it is also reachable using $\widetilde{R_{B\mathcal{P}}}$. Its proof is straightforward, and we omit it.

Proposition 4.24 *Let $\mathcal{R}_{\mathcal{P}} = (\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, R_{B\mathcal{P}})$ be a topmost rewrite theory representing protocol $\mathcal{P}$. Let $St = ss \,\&\, SS \,\&\, (ik, IK)$ where ss is a term representing a set of strands, ik is a term representing a set of intruder facts, SS is a variable for strands, and IK is a variable for intruder knowledge. If there is an initial state St_{ini} and a substitution σ such that $St \rightsquigarrow^*_{\sigma, R_{B\mathcal{P}}^{-1}, E_{\mathcal{P}}} St_{ini}$, then there is an initial state St'_{ini} and two substitutions σ', ρ such that $St \rightsquigarrow^*_{\sigma', \widetilde{R_{B\mathcal{P}}}^{-1}, E_{\mathcal{P}}} St'_{ini}$, $\sigma =_{E_{\mathcal{P}}} \sigma' \circ \rho$, and $(St'_{ini})\rho \sqsubseteq_{E_{\mathcal{P}}} St_{ini}$.*

Now, we describe how to reactivate a state. First, we formally define a ghost state.

Definition 4.25 (Ghost State) *Given a topmost rewrite theory* $\mathcal{R}_{\mathcal{P}} = (\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, R_{B\mathcal{P}})$ *representing protocol* $\mathcal{P}$ *and a state* St *containing an intruder fact* $t{\in}\mathcal{I}$ *such that* t *is a super-lazy term, we define the* ghost *version of* St*, written* $\overline{St}$*, by replacing* $t{\in}\mathcal{I}$ *in* St *by* $ghost(t)$ *in* $\overline{St}$*.*

Now, in order to resuscitate a state, we need to formally compute the strands that are generating Fresh variables relevant to the instantiation found for the super-lazy term.

Definition 4.26 (Strand Reset) *Given a strand* s *of the form* $:: r_1, \ldots, r_k :: [m_1^{\pm}, \ldots \mid \ldots, m_n^{\pm}]$*, when we want to move the bar to the rightmost position (denoting a final strand), we write* $s{\gg} =:: r_1, \ldots, r_k :: [m_1^{\pm}, \ldots, m_n^{\pm} \mid nil]$*.*

Definition 4.27 (Fresh Generating Strands) *Given a state* St *containing an intruder fact* $ghost(t)$ *for some term* t *with variables, we define the set of strands associated to* t*, denoted* $\text{strands}_{St}(t)$*, as the least set satisfying the following two conditions:*

- *for each strand* s *in* St *of the form* $:: r_1, \ldots, r_k :: [m_1^{\pm}, \ldots \mid \ldots, m_n^{\pm}]$*, if there is* $i \in \{1, \ldots, k\}$ *s.t.* $r_i \in \mathcal{V}ar(t)$*, then* $s{\gg}$ *is included into* $\text{strands}_{St}(t)$*; or*
- *for each strand* s *in* St *of the form* $:: r_1, \ldots, r_k :: [m_1^{\pm}, \ldots \mid \ldots, m_n^{\pm}]$*, if there is another strand* s' *of the form* $:: r'_1, \ldots, r'_{k'} :: [w_1^{\pm}, \ldots \mid \ldots, w_{n'}^{\pm}]$ *in* $\text{strands}_{St}(t)$*, and there are* $i \in \{1, \ldots, k\}$ *and* $j \in \{1, \ldots, n'\}$ *s.t.* $r_i \in \mathcal{V}ar(w_j)$*, then* $s{\gg}$ *is included into* $\text{strands}_{St}(t)$*.*

Now, we formally define how to resuscitate a state.

Definition 4.28 (Resuscitation) *Given a topmost rewrite theory* $\mathcal{R}_{\mathcal{P}} = (\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, \widetilde{R_{B\mathcal{P}}})$ *representing protocol* $\mathcal{P}$ *and a state* St *containing an intruder fact* $t{\in}\mathcal{I}$ *such that* t *is a super-lazy term, i.e.,* $St = ss \,\&\, (t{\in}\mathcal{I}, ik)$ *where* ss *is a term denoting a set of strands and* ik *is a term denoting the rest of the intruder knowledge. Let* $\overline{St}$ *be the ghost version of* St*. Let* St' *be a state such that* $\overline{St} \leadsto^*_{\sigma, \widetilde{R_{B\mathcal{P}}}^{-1}, E_{\mathcal{P}}} St'$ *and* $t\sigma$ *is not a super-lazy term. Let* $\sigma_t = \sigma|_{\mathcal{V}ar(t)}$*. The* reactivated *(or* resuscitated*) version of* St *w.r.t. state* St' *and substitution* σ_t *is defined as* $\widehat{St} = ss\sigma_t \,\&\, (t{\in}\mathcal{I}, ik)\sigma_t \,\&\, \text{strands}_{St'}(t\sigma_t)$*.*

Example 4.29 Consider the state ($\natural^{\circlearrowleft}$) in Example 4.22. We can check that state ($\natural^{\circlearrowleft}$) is the resuscitated version of state ($\overline{\natural}$) w.r.t. state ($\overline{\natural_+}$) and the substitution $\{K \mapsto exp(\overline{X}, n(a, \overline{r_1})), \overline{A} \mapsto a\}$. ■

Let us now prove the completeness of this state space reduction technique.

Theorem 4.30 *Given a topmost rewrite theory $\mathcal{R}_\mathcal{P} = (\Sigma_\mathcal{P}, E_\mathcal{P}, \widetilde{R_{B\mathcal{P}}})$ representing protocol $\mathcal{P}$ and a state St containing an intruder fact $t{\in}\mathcal{I}$ such that t is a super-lazy term, if there exist an initial state St_{ini} and substitution θ such that $St \leadsto^*_{\theta,\widetilde{R_{B\mathcal{P}}}^{-1},E_\mathcal{P}} St_{ini}$, then (i) there exist a state St' and substitutions τ, τ' such that $St \leadsto^*_{\tau,\widetilde{R_{B\mathcal{P}}}^{-1},E_\mathcal{P}} St'$, $\theta =_{E_\mathcal{P}} \tau \circ \tau'$, and $t\tau$ is not a super-lazy term, and (ii) there exist a reactivated version $\widehat{St}$ of St w.r.t. St' and τ, an initial state St'_{ini}, and substitutions θ', ρ such that $\widehat{St} \leadsto^*_{\theta',\widetilde{R_{B\mathcal{P}}}^{-1},E_\mathcal{P}} St'_{ini}$, $\theta =_{E_\mathcal{P}} \theta' \circ \rho$, and $(St'_{ini})\rho \subseteq_{E_\mathcal{P}} St_{ini}$.*

Proof. The sequence from St to St_{ini} can be decomposed into two fragments, computing substitutions τ and τ', respectively, such that there is a state St' and substitutions τ, τ' such that $t\tau$ is not a super-lazy term, $\theta = \tau \circ \tau'$, $St \leadsto^*_{\tau,\widetilde{R_{B\mathcal{P}}}^{-1},E_\mathcal{P}} St' \leadsto^*_{\tau',\widetilde{R_{B\mathcal{P}}}^{-1},E_\mathcal{P}} St_{ini}$, and the sequence $St \leadsto^*_{\tau,\widetilde{R_{B\mathcal{P}}}^{-1},E_\mathcal{P}} St'$ can be viewed as $St = St_0 \leadsto_{\tau_1,\widetilde{R_{B\mathcal{P}}}^{-1},E_\mathcal{P}} \cdots \leadsto_{\tau_k,\widetilde{R_{B\mathcal{P}}}^{-1},E_\mathcal{P}} St_k = St'$ such that for all $i \in \{1, \ldots, k-1\}$, $t\tau_i$ is a super-lazy term. However, using the completeness results of narrowing, Theorem 2.5, there must be a narrowing sequence from $\overline{St}$ computing such substitution τ. That is, there is a state St'' such that $\overline{St} \leadsto^*_{\tau,\widetilde{R_{B\mathcal{P}}}^{-1},E_\mathcal{P}} St''$ and St'' differs from St' (modulo $E_\mathcal{P}$-equivalence and variable renaming) only in that $t\tau{\in}\mathcal{I}$ is replaced by $ghost(t\tau)$. Let $\tau_t = \tau|_{\mathcal{V}ar(t)}$, there exists a substitution τ'' s.t. $\tau =_{E_\mathcal{P}} \tau_t \circ \tau''$. Let $\widehat{St}$ be the resuscitated version of St w.r.t. state St'' and substitution τ_t. Then, by narrowing completeness, i.e., Theorem 2.5, there exist a state St'_{ini} and substitutions σ, ρ such that $\widehat{St} \leadsto^*_{\sigma,\widetilde{R_{B\mathcal{P}}}^{-1},E_\mathcal{P}} St'_{ini}$, $\tau'' \circ \tau' =_{E_\mathcal{P}} \sigma \circ \rho$, and $(St'_{ini})\rho =_{E_\mathcal{P}} St_{ini}$. □

4.5.3 Optimizing the Super-Lazy Intruder

When we detect a state St with a super lazy term t, we may want to analyze whether the variables of t may be eventually instantiated or not before creating a ghost state. The following definition provides the key idea.

Definition 4.31 (Void Super-Lazy Term) *Given a topmost rewrite theory $\mathcal{R}_{\mathcal{P}} = (\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, \widetilde{R_{B\mathcal{P}}})$ representing protocol $\mathcal{P}$, and a state St containing an intruder fact $t \in \mathcal{I}$ such that t is a super-lazy term, if for each strand $[m_1^{\pm}, \ldots, m_{j-1}^{\pm} \mid m_j^{\pm}, \ldots, m_k^{\pm}]$ in St and each $i \in \{1, \ldots, j-1\}$, $\mathcal{V}ar(t) \cap \mathcal{V}ar(m_i) = \varnothing$, and for each term $w \in \mathcal{I}$ in the intruder's knowledge, $\mathcal{V}ar(t) \cap \mathcal{V}ar(w) = \varnothing$, then, t is called a* void super-lazy term.

Proposition 4.32 *Given a topmost rewrite theory $\mathcal{R}_{\mathcal{P}} = (\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, \widetilde{R_{B\mathcal{P}}})$ representing protocol $\mathcal{P}$ and a state St containing an intruder fact $t \in \mathcal{I}$ such that t is a void super-lazy term, let $\overline{St}$ be the ghost version of St w.r.t. the void super-lazy term t. If there exist an initial state St_{ini} and a substitution θ such that $St \leadsto^*_{\theta, \widetilde{R_{B\mathcal{P}}}^{-1}, E_{\mathcal{P}}} St_{ini}$, then there exist an initial state St'_{ini} and substitutions σ, ρ such that $\overline{St} \leadsto^*_{\sigma, \widetilde{R_{B\mathcal{P}}}^{-1}, E_{\mathcal{P}}} St'_{ini}$, $\theta =_{E_{\mathcal{P}}} \sigma \circ \rho$, and $(St'_{ini})\rho \subseteq_{E_{\mathcal{P}}} St_{ini}$.*

Proof. Since t is a super-lazy term, St_{ini} contains a sequence of intruder strands of $\mathcal{S}_{\mathcal{P}}$ generating t. Let $\theta_t = \theta|_{\mathcal{V}ar(t)}$, there exists a substitution θ' s.t. $\theta =_{E_{\mathcal{P}}} \theta_t \circ \theta'$. Since t is a void super-lazy term, there is a state St''_{ini} such that $\overline{St}\theta' \rightarrow^*_{\widetilde{R_{B\mathcal{P}}}^{-1}, E_{\mathcal{P}}} St''_{ini}$. Then, by narrowing completeness, i.e., Theorem 2.5, there are an initial state St'_{ini} and substitutions σ, ρ such that $\overline{St} \leadsto^*_{\sigma, \widetilde{R_{B\mathcal{P}}}^{-1}, E_{\mathcal{P}}} St'_{ini}$, $\theta' =_{E_{\mathcal{P}}} \sigma \circ \rho$, and $(St'_{ini})\rho \subseteq_{E_{\mathcal{P}}} St''_{ini}$. Finally, $St''_{ini} \subseteq_{E_{\mathcal{P}}} St_{ini}$, since St_{ini} simply has the strands generating t that St''_{ini} does not contain. □

4.5.4 Transition Subsumption and the Super-Lazy Intruder

Transition subsumption in the presence of the lazy intruder is computed for the most part as if the lazy intruder did not exist. That is, we

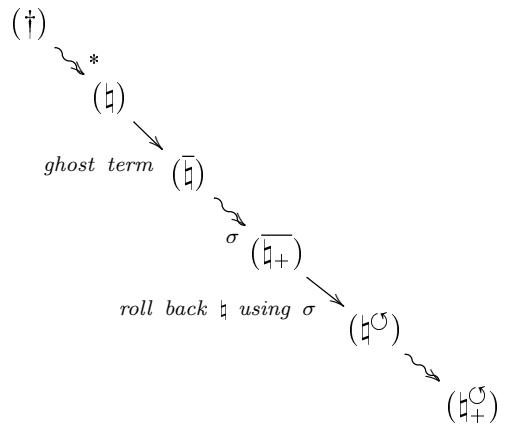

Figure 4.2: States obtained using the super-lazy intruder optimization

define a partial order on states with ghosts that extends the relation $\triangleright$ of Section 4.4.3:

Definition 4.33 *Let* $St_1 = ss_1 \,\&\, (ghost(t_1), \ldots, ghost(t_n), ik_1)$ *and let* $St_2 = ss_2 \,\&\, (ghost(t'_1), \ldots, ghost(t'_m), ik_2)$. *Let* $St'_1 = ss_1 \,\&\, (t_1 \in \mathcal{I}, \ldots, t_n \in \mathcal{I}, ik_1)$ *and let* $St'_2 = ss_2 \,\&\, (t'_1 \in \mathcal{I}, \ldots, t'_m \in \mathcal{I}, ik_2)$. *We say that* $St_1 \triangleright_0 St_2$ *if* $St'_1 \triangleright St'_2$.

However, we cannot use this definition without modification. If we do, then when a ghost state is reactivated, we see from Definition 4.33 that such a reactivated state will be $\mathcal{P}$-subsumed by the original state that raised the ghost expression, as shown in the following example.

Example 4.34 As explained in Example 4.29, the state $(\natural^{\circlearrowleft})$ is the resuscitated version of state $(\bar{\natural})$ w.r.t. state $(\overline{\natural_+})$. But, since the state $(\overline{\natural_+})$ is obtained after one backwards narrowing step from state $(\bar{\natural})$, then a state $(\natural_+^{\circlearrowleft})$, described below, is obtained after one backwards narrowing step from state $(\natural^{\circlearrowleft})$, where $(\natural_+^{\circlearrowleft})$ is similar to $(\overline{\natural_+})$ except that the ghost expression is transformed into a proper intruder fact. That is, we will find state $(\natural_+)$ twice in the search space, one as the descendant $(\overline{\natural_+})$ of $(\bar{\natural})$ and again as the descendant $(\natural_+^{\circlearrowleft})$ of $(\natural^{\circlearrowleft})$. This situation is depicted in Figure 4.2. ■

Therefore, we modify the relation $\triangleright_0$ into a new relation $\ltimes$, given in Proposition 4.37 below, by excluding resuscitated descendants which are defined by a new relation $\rightsquigarrow$ as follows.

Definition 4.35 (Resuscitated Descendant) *Given a topmost rewrite theory $\mathcal{R}_{\mathcal{P}} = (\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, \widetilde{R_{B\mathcal{P}}})$ representing protocol $\mathcal{P}$, and three non-initial states St_0, St, and St' such that St_0 contains an intruder fact $t{\in}\mathcal{I}$, and t is a super-lazy term, we say that state St' is a* resuscitated descendant *of St, written $St \rightsquigarrow St'$, if:*

1. *given the ghost version $\overline{St_0}$ of St_0 w.r.t. the super-lazy term t, then there exist $k \geqslant 1$, states $St_1, \ldots, St_k$, substitutions $\tau_1, \ldots, \tau_k$, and $i \in \{1, \ldots, k\}$ such that $\overline{St_0} \rightsquigarrow_{\tau_1, \widetilde{R_{B\mathcal{P}}}^{-1}, E_{\mathcal{P}}} St_1 \cdots St_{i-1} \rightsquigarrow_{\tau_i, \widetilde{R_{B\mathcal{P}}}^{-1}, E_{\mathcal{P}}} St_i \cdots St_{k-1} \rightsquigarrow_{\tau_k, \widetilde{R_{B\mathcal{P}}}^{-1}, E_{\mathcal{P}}} St_k$, $St =_{E_{\mathcal{P}}} St_k$, $t\tau_1 \circ \cdots \circ \tau_j$ is a super-lazy term for $1 \leqslant j \leqslant i-1$, and $t\tau_1 \circ \cdots \circ \tau_i$ is not a super-lazy term, and*

2. *given the reactivated version $\widetilde{St_0}$ of St_0 w.r.t. St_i and $\tau = \tau_1 \circ \cdots \circ \tau_i$, and let $\tau_t = \tau|_{\mathcal{V}ar(t)}$, then there exist states $St'_1, \ldots, St'_k$, substitutions $\tau'_1, \ldots, \tau'_k$ such that $\tau_j = \tau_t \circ \tau'_j$ for $1 \leqslant j \leqslant k$, and a narrowing sequence*

$$\widetilde{St} \rightsquigarrow_{\tau'_1, \widetilde{R_{B\mathcal{P}}}^{-1}, E_{\mathcal{P}}} St'_1 \cdots St'_{k-1} \rightsquigarrow_{\tau'_k, \widetilde{R_{B\mathcal{P}}}^{-1}, E_{\mathcal{P}}} St'_k$$

such that $St' =_{E_{\mathcal{P}}} St'_k$.

Example 4.36 As explained in Example 4.34, state $(\overline{\natural_+})$ is a descendant of state $(\overline{\natural})$ and state $(\natural_+^{\circlearrowleft})$ is a descendant of state $(\natural^{\circlearrowleft})$ where $(\overline{\natural_+})$ and $(\natural_+^{\circlearrowleft})$ are the same state. It is very easy to check that any descendant of $(\natural^{\circlearrowleft})$ is a resuscitated descendant of an appropriate descendant of $(\overline{\natural})$ according to Definition 4.35, e.g. $(\natural_+^{\circlearrowleft})$ is a resuscitated descendant of $(\overline{\natural_+})$. ■

Proposition 4.37 ($\mathcal{P}$-subsumption relation I) *Given a topmost rewrite theory $\mathcal{R}_{\mathcal{P}} = (\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, \widetilde{R_{B\mathcal{P}}})$ representing protocol $\mathcal{P}$, let $\ltimes$ be a partial order on states such that $St \ltimes St'$ implies that $St \triangleright_0 St'$ and $St \not\rightsquigarrow St'$. Then the partial order reduction imposed by $\ltimes$ preserves completeness of reachability as defined in Theorem 4.17.*

4.5.5 Implementing Subsumption Partial Order Reduction in the Presence of the Super-Lazy Intruder

In this section we describe how the subsumption partial order is actually implemented in Maude-NPA in the presence of the super-lazy intruder. Proposition 4.37 gives a simple way of doing this, but is not very efficient if implemented in a straightforward way, since it requires extensive examination of the search tree, examining not only the two states being compared but the narrowing path between them. Instead, we use an approximation of the $\curvearrowright$ relation that can be computed directly via a syntactic check on the state information.

This section is quite technical and depends heavily on details about Maude-NPA. The reader who is interested mainly in understanding the basic principles of the super-lazy intruder, and is not concerned about how it is actually implemented in Maude-NPA, can safely skip it. However, we believe that it is valuable to include there these technical details because it not only documents what actually is implemented in the tool, but demonstrates how one can use approximation to maximize state space reduction while minimizing the amount of search tree examination required.

In order make the presentation easier to follow, we describe our approximation in terms of a series of approximations, relations $\dashrightarrow$, $\twoheadrightarrow$, and $\twoheadrightarrow^+$, where $\dashrightarrow$ and $\twoheadrightarrow^+$ are over-approximations of the relation $\curvearrowright$ (and thus preserve completeness of reachability) but $\twoheadrightarrow$ is an under-approximation of the relation $\curvearrowright$ (and thus does not preserve completeness of reachability though it helps us to define the relation $\twoheadrightarrow^+$).

To begin with, we extend protocol states to include the actual message exchange sequence between principal or intruder strands and add a new expression $\mathit{resuscitated}(m)$ to indicate when a state has been resuscitated. This information, except for $\mathit{resuscitated}(m)$, was already included in Maude-NPA states, but its purpose before had been to assist the user in reconstructing attacks, not in performing the search itself.

The set of rewrite rules is extended to compute the exchange sequence as follows, where X is a variable denoting an exchange sequence:

$$
\begin{array}{l}
[L \mid M^-, L'] \;\&\; SS \;\&\; (M{\in}\mathcal{I}, IK) \;\&\; (M^-, X) \\
\quad \rightarrow [L, M^- \mid L'] \;\&\; SS \;\&\; (M{\in}\mathcal{I}, IK) \;\&\; X \\
[L \mid M^+, L'] \;\&\; SS \;\&\; IK \qquad\qquad \&\; (M^+, X) \\
\quad \rightarrow [L, M^+ \mid L'] \;\&\; SS \;\&\; IK \;\&\; X \\
[L \mid M^+, L'] \;\&\; SS \;\&\; (M{\notin}\mathcal{I}, IK) \;\&\; (M^+, X) \\
\quad \rightarrow [L, M^+ \mid L'] \;\&\; SS \;\&\; (M{\in}\mathcal{I}, IK) \;\&\; X
\end{array}
$$

$$
\begin{array}{l}
\text{for each } [\ l_1,\ u^+,\ l_2\ ] \in \mathcal{S}_{\mathcal{P}} : \\
[\ l_1 \mid u^+, l_2\ ] \,\&\, SS \,\&\, (u{\notin}\mathcal{I}, IK) \;\&\; (u^+, X) \rightarrow SS \,\&\, (u{\in}\mathcal{I}, IK) \;\&\; X
\end{array}
$$

Completeness reachability and soundness is clearly preserved for this set of rules and for the obvious extensions to $\overline{R_{B\mathcal{P}}}$ and $\widetilde{R_{\mathcal{P}}}$.

Example 4.38 The state ($\bar{\natural}$) of Example 4.21 will be written as the following state ($\underline{\bar{\natural}}$) by adding the message exchange sequence:

$$
\begin{array}{l}
[\ nil \mid -(K),\ -(e(K, SR)),\ +(SR)\ ] \;\& \\
:: r' :: \\
[\ -(A; B; E'), +(B; A; exp(g, n(B, r'))) \mid \\
\quad -(e(exp(E', n(B, r')), SR))\ ] \;\& \\
(ghost(K),\ e(exp(E', n(B, r')), SR){\in}\mathcal{I}, \\
e(K, SR)){\in}\mathcal{I},\ SR{\notin}\mathcal{I}) \;\& \\
(-(K),\ -(e(K, SR)),\ +(SR), \\
\quad -(e(exp(E', n(b, r')), SR)))
\end{array}
\qquad (\underline{\bar{\natural}})
$$

The state ($\overline{\natural_+}$) of Example 4.21 will be written as the following state ($\underline{\overline{\natural_+}}$) by adding the message exchange sequence:

$$[\ nil \mid -(exp(\overline{X}, n(a, \overline{r_1}))),\ -(e(exp(\overline{X}, n(a, \overline{r_1})), sec(a, \overline{r_2}))),\ +(sec(a, \overline{r_2}))\ ]\ \&$$

$$:: r' ::$$

$$[\ -(A; B; E'), +(B; A; exp(g, n(B, r'))) \mid -(e(exp(E', n(B, r')), sec(a, \overline{r_2})))\ ]\ \&$$

$$:: \overline{r_1}, \overline{r_2} ::$$

$$[\ +(a; \overline{B}; exp(g, n(a, \overline{r_1}))),\ -(\overline{B}; a; \overline{X}) \mid +(e(exp(\overline{X}, n(a, \overline{r_1})), sec(a, \overline{r_2})))]\ \& \qquad (\underline{\overline{\natural_+}})$$

$$(ghost(exp(\overline{X}, n(a, \overline{r_1}))),\ e(exp(E', n(B, r')), sec(a, \overline{r_2})) \in \mathcal{I},\ exp(\overline{X}, n(a, \overline{r_1})) \in \mathcal{I},\ (e(exp(\overline{X}, n(a, \overline{r_1})), sec(a, \overline{r_2}))) \notin \mathcal{I},\ sec(a, \overline{r_2}) \notin \mathcal{I})\ \&$$

$$(e(exp(\overline{X}, n(a, \overline{r_1})), sec(a, \overline{r_2}))^+,\ exp(\overline{X}, n(a, \overline{r_1}))^-,\ e(exp(\overline{X}, n(a, \overline{r_1})), sec(a, \overline{r_2}))^-,\ sec(a, \overline{r_2})^+,\ e(exp(E', n(b, r')), sec(a, \overline{r_2}))^-)$$

The resuscitated state ($\natural^{\circlearrowleft}$) of Example 4.22 will be written as the following state ($\underline{\natural^{\circlearrowleft}}$), where the resuscitated message is the first item in the exchange sequence:

$$[\ nil \mid -(exp(\overline{X}, n(a, \overline{r_1}))),\ +(e(exp(\overline{X}, n(a, \overline{r_1})), sec(a, \overline{r_2}))),\ +(sec(a, \overline{r_2}))\ ]\ \&$$

$$:: r' ::$$

$$[\ -(A; B; E'), +(B; A; exp(g, n(B, r'))) \mid -(e(exp(E', n(B, r')), sec(a, \overline{r_2})))\ ]\ \&$$

$$:: \overline{r_1}, \overline{r_2} ::$$

$$[\ +(a; \overline{B}; exp(g, n(a, \overline{r_1}))),\ -(\overline{B}; a; \overline{X}), +(e(exp(\overline{X}, n(a, \overline{r_1})), sec(a, \overline{r_2}))) \mid nil]\ \& \qquad (\underline{\natural^{\circlearrowleft}})$$

$$(e(exp(E', n(B, r')), sec(a, \overline{r_2})) \in \mathcal{I},\ exp(\overline{X}, n(a, \overline{r_1})) \in \mathcal{I},\ e(exp(\overline{X}, n(a, \overline{r_1})), sec(a, \overline{r_2})) \in \mathcal{I},\ sec(a, \overline{r_2} \notin \mathcal{I})\ \&$$

$$(resuscitated(exp(\overline{X}, n(a, \overline{r_1}))),\ -(exp(\overline{X}, n(a, \overline{r_1}))),\ -(e(exp(\overline{X}, n(a, \overline{r_1})), sec(a, \overline{r_2}))),\ +(sec(a, \overline{r_2})),\ -(e(exp(E', n(b, r')), sec(a, \overline{r_2}))))$$

And the state $(\underline{\natural^{\circlearrowleft}_{+}})$ of Example 4.19 will be given as follows:

$$
\begin{array}{l}
[\ nil \mid -(exp(\overline{X}, n(a, \overline{r_1}))),\ -(e(exp(\overline{X}, n(a, \overline{r_1})), sec(a, \overline{r_2}))), \\
\qquad +(sec(a, \overline{r_2}))\ ]\& \\
:: r' :: \\
[\ -(A; B; E'), +(B; A; exp(g, n(B, r'))) \mid \\
\quad -(e(exp(E', n(B, r')), sec(a, \overline{r_2})))\ ]\& \\
:: \overline{r_1}, \overline{r_2} :: \\
[\ +(a; \overline{B}; exp(g, n(a, \overline{r_1}))),\ -(\overline{B}; a; \overline{X}) \mid \qquad (\underline{\natural^{\circlearrowleft}_{+}}) \\
\quad +(e(exp(\overline{X}, n(a, \overline{r_1})), sec(a, \overline{r_2})))]\& \\
(e(exp(E', n(B, r')), sec(a, \overline{r_2})) \in \mathcal{I},\ exp(\overline{X}, n(a, \overline{r_1})) \in \mathcal{I}, \\
(e(exp(\overline{X}, n(a, \overline{r_1})), sec(a, \overline{r_2}))) \notin \mathcal{I},\ sec(a, \overline{r_2}) \notin \mathcal{I})\ \& \\
(+(e(exp(\overline{X}, n(a, \overline{r_1})), sec(a, \overline{r_2}))), \\
resuscitated(exp(\overline{X}, n(a, \overline{r_1}))),\ -(exp(\overline{X}, n(a, \overline{r_1}))), \\
\ -(e(exp(\overline{X}, n(a, \overline{r_1})), sec(a, \overline{r_2}))),\ +(sec(a, \overline{r_2})), \\
\ -(e(exp(E', n(b, r')), sec(a, \overline{r_2}))))
\end{array}
$$

where $(\underline{\natural^{\circlearrowleft}_{+}})$ is obtained from $\underline{\natural^{\circlearrowleft}}$ by one backwards narrowing step. ■

In [Escobar et al., 2008], we provided a very simple rule for approximating the relation $\rightsquigarrow$.

Definition 4.39 *Given a topmost rewrite theory $\mathcal{R}_{\mathcal{P}} = (\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, \widetilde{R_{B\mathcal{P}}})$ representing protocol $\mathcal{P}$ and two non-initial states St_1, St_2, we write St_1 --↠ St_2 if there exists a message term m such that St_1 contains an expression* ghost(m) *and St_2 contains the expression resuscitated(m).*

Example 4.40 Consider the state $(\underline{\natural})$, the state $(\underline{\natural_{+}})$, the state $(\underline{\natural^{\circlearrowleft}})$, the state $(\underline{\natural^{\circlearrowleft}_{+}})$, and the substitution $\rho = \{K \mapsto exp(\overline{X}, n(a, \overline{r_1})), r'' \mapsto \overline{r_2}\}$. It is clear now that $(\underline{\natural})\rho$ --↠ $(\underline{\natural^{\circlearrowleft}})$, $(\underline{\natural})\rho$ --↠ $(\underline{\natural^{\circlearrowleft}_{+}})$, $(\underline{\natural_{+}})\rho$ --↠ $(\underline{\natural^{\circlearrowleft}})$, and $(\underline{\natural_{+}})\rho$ --↠ $(\underline{\natural^{\circlearrowleft}_{+}})$ because all have $ghost(exp(\overline{X}, n(a, \overline{r_1})))$ in one side and $resuscitated(exp(\overline{X}, n(a, \overline{r_1})))$ on the other side. However, it is a rough approximation of the relation $\rightsquigarrow$, since we only have $(\underline{\natural})\rho \rightsquigarrow (\underline{\natural^{\circlearrowleft}})$ and $(\underline{\natural_{+}})\rho \rightsquigarrow (\underline{\natural^{\circlearrowleft}_{+}})$. ■

The following result establishes that $\dashrightarrow$ is an over-approximation of $\rightsquigarrow$, as shown in the previous example.

Lemma 4.41 *Given a topmost rewrite theory $\mathcal{R}_{\mathcal{P}} = (\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, \widetilde{R_{B\mathcal{P}}})$ representing protocol $\mathcal{P}$ and two non-initial states St_1, St_2, if $St_1 \rightsquigarrow St_2$, then there is a substitution ρ such that $St_1\rho \dashrightarrow St_2$.*

Proof. Immediate, since $St_1 \rightsquigarrow St_2$ implies that there is a substitution ρ such that $St_1\rho$ contains $ghost(m)$ and St_2 contains $resuscitated(m)$. □

Now, we can provide a better transition subsumption relation.

Proposition 4.42 ($\mathcal{P}$-subsumption relation II) *Given a topmost rewrite theory $\mathcal{R}_{\mathcal{P}} = (\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, \widetilde{R_{B\mathcal{P}}})$ representing protocol $\mathcal{P}$, let $\ltimes_{II}$ be a partial order on states such that $St \ltimes_{II} St'$ implies that $St \triangleright_0 St'$ and there is a substitution θ s.t. $St\theta \dashv\dashrightarrow St'$. Then the partial order reduction imposed by $\ltimes_{II}$ preserves completeness of reachability as defined in Theorem 4.17.*

Though this method solves the problem, since it is safe when a state St' is discarded by $St \ltimes_{II} St'$, and $St \ltimes_{II} St'$ implies $St \ltimes St'$ but not vice versa, it disables almost completely the benefits of transition subsumption for those states after a resuscitation, since the relation $\ltimes$ is able to remove many more states than $\ltimes_{II}$ in that case. Consider just Figure 4.2, then for any narrowing sequence from $(\bar{\natural})$ to a state St, there is a narrowing sequence from $(\natural^{\circlearrowleft})$ to a similar state St' but the transition subsumption would never be attempted between St and St'. Here, we provide a more concise definition of the interaction between the transition subsumption and the super-lazy intruder reduction techniques.

We characterize those states after a resuscitation that are truly linked to the parent state. First, we identify those states that are resuscitated versions of a former state. Intuitively, by comparing the exchange sequences of the two states, we can see whether the exchange sequence of the former is (L_1, M_1^-, L_2) and it has a ghost expression ghost(M_1), whereas the exchange sequence of the resuscitated version is $(L_1,$ $resuscitated(M_1), M_1^-, L_2)$.

Definition 4.43 *Given a topmost rewrite theory* $\mathcal{R}_{\mathcal{P}} = (\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, \widetilde{R_{B\mathcal{P}}})$ *representing protocol* $\mathcal{P}$ *and two non-initial states* St_1, St_2*, we say that* St_2 *is a* resuscitated version *of* St_1*, written* $St_1 \twoheadrightarrow St_2$*, if there are messages* M_1 *and* M_2 *and a substitution* ρ *such that:*

1. *state* St_1 *has a ghost of the form* $ghost(M_1)$*,*

2. *the exchange sequence of state* St_1 *is of the form*

$$(L_1, M_1^-, L_2)$$

3. *the exchange sequence of state* St_2 *is of the form*

$$(L_1', resuscitated(M_2), M_2^-, L_2'),$$

4. *and* $(L_1, M_1^-, L_2)\rho =_{E_{\mathcal{P}}} (L_1', M_2^-, L_2')$.

Example 4.44 Consider again the state $(\underline{\overline{\natural}})$, the state $(\underline{\overline{\natural_+}})$, the state $(\underline{\overline{\natural^{\circlearrowleft}}})$, the state $(\underline{\overline{\natural^{\circlearrowleft}_+}})$, and the substitution $\rho = \{K \mapsto exp(\overline{X}, n(a, \overline{r_1})), r'' \mapsto \overline{r_2}\}$. It is easy to check that $(\underline{\overline{\natural}})\rho \twoheadrightarrow (\underline{\overline{\natural^{\circlearrowleft}}})$ and $(\underline{\overline{\natural_+}})\rho \twoheadrightarrow (\underline{\overline{\natural^{\circlearrowleft}_+}})$, whereas in Example 4.40, we had $(\underline{\overline{\natural}})\rho \dashrightarrow (\underline{\overline{\natural^{\circlearrowleft}}})$, $(\underline{\overline{\natural}})\rho \dashrightarrow (\underline{\overline{\natural^{\circlearrowleft}_+}})$, $(\underline{\overline{\natural_+}})\rho \dashrightarrow (\underline{\overline{\natural^{\circlearrowleft}}})$, and $(\underline{\overline{\natural_+}})\rho \dashrightarrow (\underline{\overline{\natural^{\circlearrowleft}_+}})$. Indeed, the relation $\twoheadrightarrow$ has approximated the relation $\rightsquigarrow$, where we have only $(\underline{\overline{\natural}})\rho \rightsquigarrow (\underline{\overline{\natural^{\circlearrowleft}}})$ and $(\underline{\overline{\natural_+}})\rho \rightsquigarrow (\underline{\overline{\natural^{\circlearrowleft}_+}})$. ■

Relation $\twoheadrightarrow$ tries to approximate $\rightsquigarrow$ better than $\dashrightarrow$, but it fails, since it is an under-approximation, as shown by the following example, rather than an over-approximation, which is necessary for completeness.

Example 4.45 With one backwards narrowing step from the state $(\underline{\overline{\natural^{\circlearrowleft}_+}})$, we get the following state where message $exp(\overline{X}, n(a, \overline{r_1}))$ is learned, e.g. by extracting it from a longer message $(Y; exp(\overline{X}, n(a, \overline{r_1})))$:

$$
\begin{array}{l}
[\ -(Y; exp(\overline{X}, n(a, \overline{r_1}))) \mid +(exp(\overline{X}, n(a, \overline{r_1})))\ ]\& \\
[\ nil \mid -(exp(\overline{X}, n(a, \overline{r_1}))),\ -(e(exp(\overline{X}, n(a, \overline{r_1})), sec(a, \overline{r_2}))), \\
\qquad +(sec(a, \overline{r_2}))\ ]\& \\
:: r' :: \\
[\ -(A; B; E'), +(B; A; exp(g, n(B, r'))) \mid \\
\quad -(e(exp(E', n(B, r')), sec(a, \overline{r_2})))\ ]\& \\
:: \overline{r_1}, \overline{r_2} :: \\
[\ +(a; \overline{B}; exp(g, n(a, \overline{r_1}))),\ -(\overline{B}; a; \overline{X}) \mid \\
\quad +(e(exp(\overline{X}, n(a, \overline{r_1})), sec(a, \overline{r_2})))]\& \\
(e(exp(E', n(B, r')), sec(a, \overline{r_2})) \in \mathcal{I},\ exp(\overline{X}, n(a, \overline{r_1})) \notin \mathcal{I}, \\
(e(exp(\overline{X}, n(a, \overline{r_1})), sec(a, \overline{r_2}))) \notin \mathcal{I},\ sec(a, \overline{r_2}) \notin \mathcal{I})\ \& \\
(exp(\overline{X}, n(a, \overline{r_1}))^+, +(e(exp(\overline{X}, n(a, \overline{r_1})), sec(a, \overline{r_2}))), \\
resuscitated(exp(\overline{X}, n(a, \overline{r_1}))),\ -(exp(\overline{X}, n(a, \overline{r_1}))), \\
\quad -(e(exp(\overline{X}, n(a, \overline{r_1})), sec(a, \overline{r_2}))),\ +(sec(a, \overline{r_2})), \\
\quad -(e(exp(E', n(b, r')), sec(a, \overline{r_2}))))
\end{array}
\qquad (\gamma)
$$

We have that $\overline{\natural_+} \not\twoheadrightarrow \gamma$ because γ contains one more action in the message exchange sequence than $\overline{\natural_+}$ does, namely $exp(\overline{X}, n(a, \overline{r_1}))^+$. However, $\overline{\natural_+} \blacktriangleright_0 \gamma$, since $\overline{\natural_+}$ has only two challenges in the intruder knowledge: $e(exp(E', n(B, r')), sec(a, \overline{r_2})) \in \mathcal{I}$ and $exp(\overline{X}, n(a, \overline{r_1})) \in \mathcal{I}$, and so the transition subsumption is applied, discarding state γ as unreachable whereas it should not be discarded. ■

We cannot prove completeness of reachability but we can prove soundness of the relation $\twoheadrightarrow$.

Lemma 4.46 *Given a topmost rewrite theory $\mathcal{R}_\mathcal{P} = (\Sigma_\mathcal{P}, E_\mathcal{P}, \widetilde{R_{B\mathcal{P}}})$ representing protocol $\mathcal{P}$ and two non-initial states St_1, St_2, if $St_1 \twoheadrightarrow St_2$, then $St_1 \rightsquigarrow St_2$.*

Proof. Immediate by the fact that $St_1 \twoheadrightarrow St_2$ implies there is a substitution ρ such that $St_1\rho$ has a term $ghost(m)$ and St_2 has a term $resuscitated(m)$, and the exchange sequence of both states is identical except the message $resuscitated(m)$. □

We need to go even further and restrict $\twoheadrightarrow$ to obtain a closer characterization of $\rightsquigarrow$, namely a new relation $\twoheadrightarrow^+$. Relation $St_1 \twoheadrightarrow St_2$ takes into account only whether St_2 is a resuscitated version of St_1, but does not consider what happens beyond the state that produced the instantiation that reactivated the ghost state. That is, descendants of the state that produced the ghost state that are instantiations of descendants St_1. Intuitively, in the following definition below, we compare the exchange sequences of the two states to see whether the exchange sequence of the first is (L_1, L_2, M_1^-, L_3) and it has a ghost expression ghost(M_1), whereas the exchange sequence of the second is $(L_1, M_1^+, L_2, \mathit{resuscitated}(M_1), M_1^-, L_3)$. Indeed, a recursive definition can be given here that becomes extremely useful when several resuscitations have happened in a concrete state.

Definition 4.47 *Given a topmost rewrite theory $\mathcal{R}_\mathcal{P} = (\Sigma_\mathcal{P}, E_\mathcal{P}, \widetilde{R_{B\mathcal{P}}})$ representing protocol $\mathcal{P}$ and two non-initial states St_1, St_2, we say that St_2 is a* resuscitated version *of St_1, written $S_1 \twoheadrightarrow^+ St_2$, if either $S_1 \twoheadrightarrow St_2$ or there are messages M_1 and M_2, a substitution ρ, and sequences L_1', L_1'' such that:*

1. *state St_1 has a ghost of the form ghost(M_1),*
2. *the exchange sequence of state St_1 is of the form*

$$(L_1, L_2, M_1^-, L_3)$$

3. *the exchange sequence of state St_2 is of the form*

$$(L_1', M_2^+, L_2', \mathit{resuscitated}(M_2), M_2^-, L_3')$$

4. $(L_2, M_1^-, L_3)\rho =_{E_\mathcal{P}} (L_2', M_2^-, L_3')$
5. *and either*
   (a) *there is a subsequence L_1''' of L_1' such that $L_1\rho =_{E_p} (L_1''')$ or*
   (b) *$St_1' \twoheadrightarrow^+ St_2'$ where St_1' is St_1 without the ghost(M_1) expression and St_2' is St_2 with the shorter exchange sequence $(L_1', L_2', M_2^-, L_3')$.*

The following result establishes that $\twoheadrightarrow^+$ is a better approximation of $\rightsquigarrow$ than $\dashrightarrow$.

Lemma 4.48 *Given a topmost rewrite theory $\mathcal{R}_\mathcal{P} = (\Sigma_\mathcal{P}, E_\mathcal{P}, \widetilde{R_{B\mathcal{P}}})$ representing protocol $\mathcal{P}$ and two non-initial states St_1, St_2, if $St_1 \rightsquigarrow St_2$, then $St_1 \twoheadrightarrow^+ St_2$.*

Proof. If $St_1 \rightsquigarrow St_2$, then state St_1 has at least one ghost expression of the form $ghost(M_1)$ for a message M_1, the exchange sequence of state St_1 is of the form (L_1, M_1^-, L_2) for two message sequences L_1 and L_2, the exchange sequence of state St_2 is of the form $(L_1', resuscitated(M_2), M_2^-, L_2')$ for a message M_2 and two message sequences L_1' and L_2', and there is a substitution ρ such that $(M_1^-, L_2)\rho =_{E_\mathcal{P}} (M_2^-, L_2')$. Now, we prove the result by induction on the number of ghost expressions in St_1.

If there is only one ghost expression in St_1, then we consider whether L_1 and L_1' match or not. If there is a substitution σ_1 such that $L_1\sigma_1 =_{E_\mathcal{P}} L_1'$, then we are done, since $(L_1\sigma_1, (M_1\rho)^-, L_2\rho) =_{E_\mathcal{P}} (L_1', M_2^-, L_2')$ and this implies $St_1 \twoheadrightarrow St_2$. Otherwise, L_1' contains at least one action M_2^+ that does not appear in L_1 which could never be done from St_1 because $M_1\rho$ appeared in St_1 as a ghost expression instead of an expression $M_1\rho{\in}\mathcal{I}$. That is, L_1' is of the form $(L_{1,1}', M_2^+, L_{1,2}')$ for message sequences $L_{1,1}'$ and $L_{1,2}'$, i.e., the message sequence of St_2 is of the form $(L_{1,1}', M_2^+, L_{1,2}', resuscitated(M_2), M_2^-, L_2')$, and L_1 is of the form $(L_{1,1}, L_{1,2})$ for message sequences $L_{1,1}, L_{1,2}$, i.e., the message sequence of St_1 is of the form $(L_{1,1}, L_{1,2}, (M_1\rho)^-, L_2\rho)$. Then, there is a substitution σ_2 such that $(L_{1,2})\sigma_2 =_{E_\mathcal{P}} L_{1,2}'$, since $L_{1,2}$ and $L_{1,2}'$ correspond to actions related to M_1^- and not related to M_2^+. Furthermore, since $St_1 \twoheadrightarrow^+ St_2$, there is a subsequence of $L_{1,1}'$ containing all the elements of $(L_{1,1})\sigma_2$, so that the remaining elements of $L_{1,1}'$, which are not contained in $(L_{1,1})\sigma_2$, are related to M_2^+. This concludes the case for one ghost expression.

If there are more than one ghost expression in St_1, then there is no subsequence of $L_{1,1}'$ containing all the elements of $L_{1,1}$ in the previous case because there is more than one resuscitation between St_1 and St_2. In this case, we remove all the material from St_1 and St_2 connected to the ghost term M_2 (and $M_1\rho$) obtaining terms St_1' and St_2' and, since $St_1 \rightsquigarrow St_2$, we have that $St_1' \rightsquigarrow St_2'$ because only the message exchange sequence is altered and, by induction hypothesis, $St_1' \twoheadrightarrow^+ St_2'$ concluding that $St_1 \twoheadrightarrow^+ St_2$. □

Protocol	none					All optimizations					%
Needham-Shroeder Public Key	5	19	142	727	4904	4	6	4	2	1	99
Needham-Shroeder Lowe's fix	5	19	142	727	4902	4	7	6	2	-	81
SecReT06	1	6	22	111	312	2	3	2	-	-	98
SecReT07	8	24	212	902	8047	5	1	1	1	-	99
Diffie-Hellman	3	24	72	316	1884	4	6	10	9	12	98
Homo-hpc	1	8	22	100	533	2	2	1	1	1	98
Homo-NSL	4	15	92	418	2409	4	9	10	9	11	98
Amended Needham-Shroeder	1	8	24	121	781	2	4	9	20	41	91
Carlsen's Secret Key Initiator Protocol	5	19	145	764	5931	4	7	10	18	12	99
Denning-Sacco	6	16	65	357	1628	1	2	3	5	7	99
ISO-5-Pass Authentication	5	19	145	766	5982	5	5	13	19	20	99
Kao Chow Repeated Authentication	5	19	144	795	6059	4	7	9	11	5	99
Kao Chow Repeated Authentication-Handshake Key	1	7	22	99	555	2	2	6	12	18	94
Kao Chow Repeated Authentication-Ticket	5	19	139	715	5382	8	35	33	27	118	96
NSL-ECB-homo	5	18	116	532	3006	3	8	15	16	10	98
NSL-XOR-modified	1	10	42	442	6184	4	5	5	5	2	99
Otway-Rees	1	7	18	55	237	2	6	14	34	84	55
Wide Mouthed Frog	6	31	226	1492	11788	6	13	27	44	65	98
Woo-Lam	1	8	24	115	936	3	5	6	5	6	97

Table 4.1: Number of new states produced in each of 1,2,3,4 and 5 backwards narrowing steps comparing each optimization of Sections 4.3.1,4.3.2,4.4.2,4.4.3, and 4.5.

Now, we can provide a better transition subsumption relation.

Proposition 4.49 ($\mathcal{P}$-subsumption relation III) *Given a topmost rewrite theory $\mathcal{R}_{\mathcal{P}} = (\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, \widetilde{R_{B\mathcal{P}}})$ representing protocol $\mathcal{P}$, let $\ltimes_{III}$ be a partial order on states such that $St \ltimes_{III} St'$ implies that $St \blacktriangleright_0 St'$ and $St \twoheadrightarrow^+ St'$. Then the partial order reduction imposed by $\ltimes_{III}$ preserves completeness of reachability as defined in Theorem 4.17.*

4.6 Experimental Evaluation

In order to measure the contribution that the various optimization techniques make to improve the Maude-NPA protocol analysis, we ran Maude-NPA on several example protocols, with no reduction method in place, with only one reduction method in place, and with all the reduction methods in place. Note that Maude-NPA was never intended to run without these optimizations. Maude-NPA will never terminate unless at least the grammars or the subsumption relation are used and, thus, we

also provide information on whether or not we were able to achieve termination, that is, if we reached a depth at which all states are initial. In our experiments, we use the results for the case with no optimizations as a baseline that allows us to compare the different optimization techniques with each other.

In Table 4.1 we summarize the experimental evaluation of the impact of the different state space reduction techniques for these example protocols searching up to depth 5. We also provide tables for each single reduction method. Table 4.3 analyzes the most powerful reduction method, grammars, of Section 4.3.1. Table 4.4 shows the comparison of the reduction method of Section 4.3.2. Table 4.5 analyzes the transition subsumption of Section 4.4.3, which is one of the most powerful reduction methods. And Table 4.6 analyzes the also powerful super-lazy intruder reduction method of Section 4.5. Note that the label "-" means that the reachability analysis finished some levels before. The source files of the protocol specifications are available at:

`http://www.dsic.upv.es/~sescobar/Maude-NPA/redTechniques.html`

The overall percentage of state-space reduction for each protocol and the average of nearly 95% suggest that our combined techniques are remarkably effective (the reduced number of states is almost only 5% or less of the original number of states).

Table 4.2 shows whether the protocols have a finite state space or not. We show the different techniques yielding a finite space for each protocol, when it is the case. The use of grammars and the transition subsumption are clearly the most useful techniques in general. Note that grammars are insufficient to achieve termination for the SecReT07 example, while subsumption and the super lazy intruder are essential in this case.

4.7 Conclusions

The Maude-NPA can analyze the security of cryptographic protocols, modulo given algebraic properties of the protocol's cryptographic functions in executions with an unbounded number of sessions and with no approximations or data abstractions. In this full generality, protocol security properties are well-known to be undecidable. The Maude-NPA

Protocol	Is State Space Finite?
Needham-Shroeder Public Key	Yes, by Grammars and Subsumption
Needham-Shroeder Lowe's fix	Yes, by Grammars and Subsumption
SecReT06	Yes, by Subsumption or (Grammars and Lazy)
SecReT07	Yes, by Subsumption and Lazy
Diffie-Hellman	Yes, by Grammars and Subsumption
Homo-hpc	Yes, by Grammars and Subsumption
Homo-NSL	Yes, by Grammars, Subsumption and Lazy
Amended Needham-Shroeder	No and no attack is found
Carlsen's Secret Key Initiator Protocol	No and no attack is found
Denning-Sacco	Yes, by Grammars, Subsumption and Lazy
ISO-5-Pass Authentication	No and no attack is found
Kao Chow Repeated Authentication	Yes, Grammars, Subsumption and Lazy
Kao Chow Repeated Authentication-Handshake Key	No and no attack is found
Kao Chow Repeated Authentication-Ticket	Yes, by Grammars, Subsumption and Lazy
NSL-ECB-homo	Yes, by Grammars and Subsumption
NSL-XOR-modified	Yes, by Grammars, Subsumption and Lazy
Otway-Rees	No, but an authentication attack is found
Wide Mouthed Frog	No, but a secrecy attack is found
Woo-Lam	Yes, Grammars and Subsumption

Table 4.2: Finite state space achieved by reduction techniques

uses backwards narrowing-based search from a symbolic description of a set of attack states by means of patterns to try to reach an initial state of the protocol. If an attack state is reachable from an initial state, the Maude-NPA's complete narrowing methods are guaranteed to prove it. But if the protocol is secure, the backwards search may be infinite and never terminate.

It is therefore very important, both for efficiency, and to achieve full verification whenever possible when a protocol is secure, to use *state-space reduction techniques* that: (i) can drastically cut down the number of states to be explored; and (ii) have in practice a good chance to make the, generally infinite, search space finite without compromising the completeness of the analysis; that is, so that if a protocol is indeed secure, failure to find an attack in such a finite state space guarantees the protocol's security for that attack relative to the assumptions about the intruder actions and the algebraic properties. We have presented a number of state-space reduction techniques used in combination by the Maude-NPA for exactly these purposes. We have given precise characterizations of theses techniques and have shown that they preserve soundness and completeness, so that: 1) any attack that is found is valid, and 2) if no attack is found and the state space is finite, full verification of the given

Protocol	none					Grammars					%
Needham-Shroeder Public Key	5	19	142	727	4904	4	12	49	185	769	82
Needham-Shroeder Lowe's fix	5	19	142	727	4902	4	12	50	190	804	81
SecReT06	1	6	22	111	312	1	2	6	15	36	86
SecReT07	8	24	212	902	8047	7	21	181	747	6713	16
Diffie-Hellman	3	24	72	316	1884	3	12	30	80	233	84
Homo-hpc	1	8	22	100	533	1	2	4	10	23	93
Homo-NSL	4	15	92	418	2409	4	14	78	336	1671	28
Amended Needham-Shroeder	1	8	24	121	781	1	3	7	24	96	85
Carlsen's Secret Key Initiator Protocol	5	19	145	764	5931	4	13	62	265	1322	75
Denning-Sacco	6	16	65	357	1628	2	4	10	27	54	95
ISO-5-Pass Authentication	5	19	145	766	5982	4	14	73	322	1562	71
Kao Chow Repeated Authentication	5	19	144	795	6059	4	13	63	271	1336	75
Kao Chow Repeated Authentication-Handshake Key	1	7	22	99	555	1	5	13	40	152	69
Kao Chow Repeated Authentication-Ticket	5	19	139	715	5382	5	19	139	715	5382	0
NSL-ECB-homo	5	18	116	532	3006	2	6	21	75	295	89
NSL-XOR-modified	1	10	42	442	6184	1	8	34	353	5193	16
Otway-Rees	1	7	18	55	237	1	3	6	13	44	78
Wide Mouthed Frog	6	31	226	1492	11788	4	18	93	458	2357	78
Woo-Lam	1	8	24	115	936	1	3	7	23	135	84

Table 4.3: Number of new states produced in each of 1,2,3,4 and 5 backwards narrowing steps with and without the optimization of Section 4.3.1.

security property is achieved.

Using several representative examples we have also given an experimental evaluation of these techniques. Our experiments support the conclusion that, when used in combination, these techniques: (i) typically provide drastic state space reductions, removing as much as 95 percent of the states that would otherwise be generated; and (ii) they can often yield a *finite* state space, so that whether the desired security property holds or not can in fact be decided automatically, in spite of the general undecidability of such problems.

Protocol	none					Inconsistency					%
Needham-Shroeder Public Key	5	19	142	727	4904	5	18	96	318	1663	63
Needham-Shroeder Lowe's fix	5	19	142	727	4902	5	18	97	318	1664	63
SecReT06	1	6	22	111	312	1	6	22	107	290	5
SecReT07	8	24	212	902	8047	8	22	178	697	6210	22
Diffie-Hellman	3	24	72	316	1884	3	21	27	132	342	77
Homo-hpc	1	8	22	100	533	1	8	22	91	427	17
Homo-NSL	4	15	92	418	2409	4	14	60	190	883	60
Amended Needham-Shroeder	1	8	24	121	781	1	7	8	59	162	74
Carlsen's Secret Key Initiator Protocol	5	19	145	764	5931	5	18	100	348	2182	61
Denning-Sacco	6	16	65	357	1628	5	5	29	74	439	73
ISO-5-Pass Authentication	5	19	145	766	5982	5	18	100	349	2206	61
Kao Chow Repeated Authentication	5	19	144	795	6059	5	18	99	362	2186	61
Kao Chow Repeated Authentication-Handshake Key	1	7	22	99	555	1	7	22	93	476	12
Kao Chow Repeated Authentication-Ticket	5	19	139	715	5382	5	19	135	677	4589	13
NSL-ECB-homo	5	18	116	532	3006	5	17	90	281	1291	54
NSL-XOR-modified	1	10	42	442	6184	1	7	10	143	1422	76
Otway-Rees	1	7	18	55	237	1	6	6	33	65	65
Wide Mouthed Frog	6	31	226	1492	11788	6	29	189	1183	8591	26
Woo-Lam	1	8	24	115	936	1	8	24	108	812	12

Table 4.4: Number of new states produced in each of 1,2,3,4 and 5 backwards narrowing steps with and without the optimization of Section 4.3.2.

Protocol	none					Subsumption					%
Needham-Shroeder Public Key	5	19	142	727	4904	5	15	61	107	237	92
Needham-Shroeder Lowe's fix	5	19	142	727	4902	5	15	61	107	237	92
SecReT06	1	6	22	111	312	1	6	15	31	40	79
SecReT07	8	24	212	902	8047	6	15	61	165	506	91
Diffie-Hellman	3	24	72	316	1884	2	14	26	102	288	81
Homo-hpc	1	8	22	100	533	1	8	15	72	174	59
Homo-NSL	4	15	92	418	2409	4	12	41	68	181	89
Amended Needham-Shroeder	1	8	24	121	781	1	8	15	70	203	68
Carlsen's Secret Key Initiator Protocol	5	19	145	764	5931	5	15	69	192	825	83
Denning-Sacco	6	16	65	357	1628	6	11	44	102	393	73
ISO-5-Pass Authentication	5	19	145	766	5982	5	15	69	194	837	83
Kao Chow Repeated Authentication	5	19	144	795	6059	5	15	68	194	792	84
Kao Chow Repeated Authentication-Handshake Key	1	7	22	99	555	1	7	13	53	144	68
Kao Chow Repeated Authentication-Ticket	5	19	139	715	5382	5	15	69	204	858	81
NSL-ECB-homo	5	18	116	532	3006	5	14	55	129	410	83
NSL-XOR-modified	1	10	42	442	6184	1	8	16	71	152	96
Otway-Rees	1	7	18	55	237	1	7	12	43	104	47
Wide Mouthed Frog	6	31	226	1492	11788	6	20	76	212	721	92
Woo-Lam	1	8	24	115	936	1	8	15	67	246	68

Table 4.5: Number of new states produced in each of 1,2,3,4 and 5 backwards narrowing steps with and without the optimization of Section 4.4.3.

Protocol	none					Super-lazy Intruder					%
Needham-Shroeder Public Key	5	19	142	727	4904	5	19	142	726	4822	1
Needham-Shroeder Lowe's fix	5	19	142	727	4902	5	19	142	726	4820	1
SecReT06	1	6	22	111	312	1	6	22	111	306	1
SecReT07	8	24	212	902	8047	8	22	62	199	648	89
Diffie-Hellman	3	24	72	316	1884	3	24	72	297	1307	25
Homo-hpc	1	8	22	100	533	1	8	22	100	533	0
Homo-NSL	4	15	92	418	2409	4	15	92	417	2335	2
Amended Needham-Shroeder	1	8	24	121	781	1	8	24	95	573	25
Carlsen's Secret Key Initiator Protocol	5	19	145	764	5931	5	15	59	266	1291	76
Denning-Sacco	6	16	65	357	1628	6	16	63	237	1014	35
ISO-5-Pass Authentication	5	19	145	766	5982	5	15	59	268	1307	76
Kao Chow Repeated Authentication	5	19	144	795	6059	5	15	59	269	1321	76
Kao Chow Repeated Authentication-Handshake Key	1	7	22	99	555	1	1	1	6	13	96
Kao Chow Repeated Authentication-Ticket	5	19	139	715	5382	5	15	57	243	1115	77
NSL-ECB-homo	5	18	116	532	3006	5	18	116	531	2896	3
NSL-XOR-modified	1	10	42	442	6184	1	10	42	311	2077	63
Otway-Rees	1	7	18	55	237	1	7	18	49	193	15
Wide Mouthed Frog	6	31	226	1492	11788	6	23	113	663	3872	65
Woo-Lam	1	8	24	115	936	1	8	24	90	331	57

Table 4.6: Number of new states produced in each of 1,2,3,4, and 5 backwards narrowing steps with and without the optimization of Section 4.5.

Chapter 5

A Rewriting-based Forwards Semantics for Maude-NPA

In this chapter we define a novel rewriting-based forwards semantics appropiate for the Maude-NPA protocol analysis tool that can be safely integrated with its narrowing-based backwards semantics.

Section 5.1 gives an overview of our approach. More details on how to perform a foward execution of a protocol are given in Section 5.2. The rules of Maude-NPA's forwards operational semantics are described in Section 5.3. In Section 5.4 we prove the soundness and completeness of the backwards semantics with respect to the forwards semantics. Section 5.5 presents the experimental results obtained with our forwards rewriting tool. Finally, Section 5.6 concludes the chapter and discusses some of the issues that comparing the backwards with the forwards semantics helped us resolve.

These results have been published in [Escobar et al., 2014b].

5.1 Overview

Over the years a number of different techniques have been applied to security analysis of cryptographic protocols via state space exploration. These techniques can either rely on an explicit-state model-cheking using forward search, or on a symbolic-state model-checking, typically using backward search. As explained in Chapter 1, each approach is more appropiated than the other for certain purposes. Therefore, one may

prefer not to limit oneself to one approach, but to switch back and forth between these two. By integrating the two approaches we can obtain the best of both worlds, using each technique where it works best. But for such an integration to be correct and useful two requirements should be met: (1) the forwards and backwards tools should share the *same semantic model* and language; and (2) the operational semantics used in the forwards and backwards analyses should *agree with each other*.

Although Maude-NPA already has an intuitive forwards semantics obtained by reversing the rewrite rules defining the backwards semantics, it is not suitable for model checking. Designing a suitable forwards semantics requires much more than simply reversing the transition rules of the backwards semantics. There are three main facts that make such forwards semantics unsuitable for model-checking purposes. First, the rewrite rules in the backwards semantics can introduce *extra variables*. This is unproblematic for narrowing-based symbolic analysis, but unacceptable for rewriting-based forwards execution. Second, in the backwards semantics a state contains explicit information about events occurring in the future (since they were observed "earlier" in the backwards search). Reversing rules of type (3.1), (3.2), and (3.3) (see Section 3.4, in Page 50) allow the definition of an intuitive forwards semantics associated to Maude-NPA but it works only for validation of a given execution sequence, where the initial state must contain all the strands, and not for searching for an attack, where the initial state must be empty and strands would have to be added during forwards search. This information must be removed in the forwards semantics, while still ensuring that reachability is not affected. Third, in the backwards semantics fresh values (nonces, session keys, etc.) are represented by special variables. These must be replaced by constants in the forwards semantics, again without affecting reachability. In the following, we explain how these three issues have been approached in the rewriting-based forwards semantics for Maude-NPA presented in this chapter.

In a forwards semantics we also specify protocols and states in Maude-NPA using the strand space model, similarly as in the backwards semantics (see Section 3.2). However, there are some minor differences, which we explain in the following.

In contrast to the backwards semantics, we explore *concrete states* in the forwards semantics, That is, a state is an $E_{\mathcal{P}}$-equivalence class

$[t]_{E_\mathcal{P}} \in T_{\Sigma_\mathcal{P}/E_\mathcal{P}}$ with t a ground $\Sigma_\mathcal{P}$-term.

Similarly as the backwards semantics, a state consists of a multiset of partially executed *strands* S_i and a set of terms in the intruder's knowledge, i.e., a state is a term of the form $\{S_1 \,\&\, \cdots \,\&\, S_n \,\&\, \{IK\}\}$ where & is an associative-commutative union operator. However, in the forward semantics, & *cannot be* idempotent, since there will be situations where two occurrences of the same strand will lead to completely different strands later on but their current partial representation in the state makes them equal. So keeping only one occurrence of the partial strand is wrong.

In the forwards semantics intruder's knowledge facts are all of the form $m{\in}\mathcal{I}$ (the intruder knows m) where m is a message expression. Moreover, strands do not evolve over time; that is, a strand is a term of the form $[msg_1^{\pm}, \ldots, msg_{k-1}^{\pm}, msg_k^{\pm}]$, with no vertical bar | separating past and future messages. A strand represents the exchange of message performed up to the present moment of the protocol execution.

Similarly as in the backwards semantics, variables of each positive message in a strand must appear in previous input messages (see Footnote 3 in Page 44). However, in the forwards semantics, this restriction must be applied to variables denoting principal names and variables of sort Fresh created by each strand too, unlike the backwards semantics. This requirement is essential in the rewriting-based forwards semantics for obtaining rewrite rules without extra variables, i.e., rewrite rules $l \to r$ where $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$, which allows for effectively executable rewriting computations.

Our solution is based on two ideas. First, to match input terms in a strand always with the intruder's knowledge. Let us illustrate this with the intruder encryption capability $[-(K), -(M), +(e(K, M)]$. In the backward semantics of Section 3.4 the encryption strand above produces the following rewrite rule adding a new strand (when the rules are executed backwards) if a message of the form $e(K, M)$ appears in the intruder knowledge:

$$\{SS \,\&\, [-(K), -(M) \mid +(e(K, M)] \,\&\, \{(e(K, M){\notin}\mathcal{I}, IK\}\}$$
$$\to \{SS \,\&\, \{e(K, M){\in}\mathcal{I}, IK\}\}$$

whereas in the new forward semantics that rule is defined differently:

$$\{SS \,\&\, \{K{\in}\mathcal{I}, M{\in}\mathcal{I}, IK\}\}$$
$$\rightarrow \{SS \,\&\, [-(K), -(M), +(e(K,M))] \,\&\, \{K{\in}\mathcal{I}, M{\in}\mathcal{I}, e(K,M){\in}\mathcal{I}, IK\}\}$$

Second, principal names as well as new fresh variables are treated as numeric constants by using a global counter $\langle N \rangle$ that will be appropriately incremented. For instance, the Dolev-Yao strand for new nonces $:: r :: [+(n(i,r))]$ will be represented by a transition rule of the form:

$$\{SS \;\&\; \{IK\} \;\&\; \langle N \rangle\} \rightarrow \{SS \;\&\; [+(n(i,N))] \;\&\; \{IK\} \;\&\; \langle N+1 \rangle\}$$

where the global counter N is incremented by one. The formal definition of how forwards transition rules are generated from the strand specification now requires some notation to indicate how the global counter is increased. Given a message u and a counter $\langle i \rangle$, we write $u{\uparrow}_i^n$, to denote that those principal names and fresh variables appearing in term u that are identified as new have been numbered starting with i and ending in $n-1$, with n the next available value of the counter. For example, given the term $u = exp(g, n(0,1) * n(A,r))$ where 0 corresponds to a principal name already replaced and 1 to a fresh variable already replaced, but A is a new principal name and r is a new fresh variable, we write $u{\uparrow}_{10}^{12} = exp(g, n(0,1) * n(10,11))$, i.e., the substitution $\{A \mapsto 10, r \mapsto 11\}$ has been applied. In the forwards semantics, we remove the list of fresh variables at the beginning of each strand and the vertical bar, since they are no longer necessary.

5.2 Forward Reachability Analysis

In a forward execution of a protocol we begin with an empty initial state containing no information in the intruder knowledge and, since we consider the unbounded session case, no strand in the initial state. The execution of the protocol implies searching for a final state which is an instance of the pattern denoting the desired class of attack states.

Example 5.1 Given the protocol with Diffie-Hellman exponentiation of Example 3.1 in Page 45, the initial state is just the empty state:

$$\{\varnothing \;\&\; \{empty\} \;\&\; \langle 0 \rangle\}$$

The final state pattern where the intruder has learned the secret is as follows, where Y, SR, SS, IK, and r are variables and a and b represent the actual names of Alice and Bob:

$$\{\ [-(a;b;Y), +(b;a;exp(g,n(b,r))), -(e(exp(Y,n(b,r)),SR))] \\ \&\ SS\ \&\ \{SR{\in}\mathcal{I}, IK\}\ \}$$

The forwards analysis is easily performed in the Maude system by using Maude's `search` command, which receives the initial term and the final pattern as input and generates the search state space. Similar to the backwards analysis, the solution to the forwards reachability analysis is as follows. The principal strands are (where ca, cb, cb', r_1, r_2, r_3, r_4 are natural numbers but the actual value is irrelevant):

$$[+(ca;cb';exp(g,n(ca,r_2))), \\ -(cb';ca;exp(g,n(i,r_4))), \\ +(e(exp(g,n(i,r_4)*n(ca,r_2)),sec(ca,r_1)))]) \;\& \\ [-(ca;cb;exp(g,n(i,r_4))), \\ +(cb;ca;exp(g,n(cb,r_3))), \\ -(e(exp(g,n(i,r_4)*n(cb,r_3)),sec(ca,r_1)))] \;\&$$

The Dolev-Yao intruder strands are as follows:

$$
\begin{array}{l}
[+(n(i,r_4))] \;\&\\
[-(ca;cb';exp(g,n(ca,r_2))),+(cb';exp(g,n(ca,r_2)))] \;\&\\
[-(cb';exp(g,n(ca,r_2))),+(exp(g,n(ca,r_2)))] \;\&\\
[-(exp(g,n(ca,r_2))),-(n(i,r_4)),\\
\;+(exp(g,n(i,r_4)*n(ca,r_2)))] \;\&\\
[-(exp(g,n(i,r_4)*n(ca,r_2))),\\
\;-(e(exp(g,n(i,r_4)*n(ca,r_2)),sec(ca,r_1))),\\
\;+(sec(ca,r_1))] \;\&\\
[-(exp(g,n(cb,r_3))),-(n(i,r_4)),\\
\;+(exp(g,n(i,r_4)*n(cb,r_3)))] \;\&\\
[-(exp(g,n(i,r_4)*n(cb,r_3))),-(sec(ca,r_1)),\\
\;+\,(e(exp(g,n(i,r_4)*n(cb,r_3)),sec(ca,r_1)))] \;\&\\
[-(cb;ca;exp(g,n(cb,r_3))),+(ca;exp(g,n(cb,r_3)))] \;\&\\
[-(ca;exp(g,n(cb,r_3))),+(exp(g,n(cb,r_3)))]
\end{array}
$$

■

It is also possible to specify authentication attacks in Maude for the forwards semantics by using again Maude's `search` command (see Section 2.3, Page 37) to search for an attack, but making it *conditional* to the never patterns not having been encountered. Note that the forwards semantics is *monotonic* in the sense that for s, s' concrete states such that $s \rightarrow^* s'$, then s' "stores" s as a "substate." This means that if a never pattern is avoided by s' it is also avoided by s. Therefore, given an attack pattern S and never patterns $S_1, \ldots, S_n$, we can search for an attack avoiding such patterns by giving to Maude the conditional search command:

$$
\begin{array}{l}
\mathit{search\ init} \rightarrow^* S\ \mathit{such\ that}\\
p_{S_1}(S) = \mathit{false} \wedge \ldots \wedge\ p_{S_n}(S)\ \ =\ \ \mathit{false}\ .
\end{array}
$$

where each predicate p_{S_i} holds for a concrete state s iff s is an instance of the pattern S_i. When a never pattern S_i shares variables $v_0, \ldots, v_{k_i}$ with S, the predicate p_{S_i} is extended in the form $p_{S_i}(S, v_0, \ldots, v_{k_i})$; if a never

pattern has variables not appearing in S, these will be created and used within the predicate. This method has allowed us to analyze by forward model checking all the examples in Section 5.5.

Example 5.2 Given the following attack S, and never patterns $Alice_1$ and $Alice_2$ (not sharing variables with S) of the Diffie-Hellman protocol in Example 3.2:

$$\begin{aligned} S \;=\; \{ :: r'' :: [-(a;b;Y), +(b;a;exp(g,n(b,r''))),\\ -(e(exp(Y,n(b,r'')),SR)) \mid nil]\\ \&\; SS \;\&\; \{IK\}\}\\ Alice_1 = :: r, r' :: [+(a;b;exp(g,n(a,r)))]\\ Alice_2 = :: r, r' :: [+(a;b;exp(g,n(a,r))), -(b;a;X),\\ +(e(exp(X,n(a,r)),sec(a,r')))] \end{aligned}$$

The Maude conditional search command to search for the attack S avoding the never patterns for Alice's strand, is as follows:

$$\begin{aligned} &search\ init \rightarrow^* S\ such\ that\\ &p_{Alice_1}(S) = false \wedge p_{Alice_2}(S) \;=\; false\ . \end{aligned}$$

where the predicates p_{Alice_1} and p_{Alice_2} check whether any strand of the concrete state S is an instance of the strands $Alice_1$ and $Alice_2$, respectively. ■

5.3 Forwards Operational Semantics

In a forward reachability analysis, we define state changes by means of a set $R_{F\mathcal{P}}$ of *rewrite rules*, so that the rewrite theory $(\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, R_{F\mathcal{P}})$ characterizes the behavior of protocol $\mathcal{P}$ modulo the equations $E_{\mathcal{P}}$. Here we do not have generic transition rules, as in the backwards semantics, and all the rules are generated from principal and intruder strands. The intuitive idea is that a state consists of a multiset of partially executed strands and a set of terms in the intruder's knowledge. Unlike the backwards semantics, only the part of the strand that has already executed is present in the state, and each such partial strand instantiates a prefix

of a strand in $\mathcal{P}$. One progresses by either: (i) adding a positive term m^+ to an existing strand and either adding or not adding m to the intruder's knowledge, (ii) adding a negative term m^- to an existing strand only if it is already present in the intruder's knowledge, or (iii) starting a new strand, and if it starts with a m^+ that either adds or not to the intruder's knowledge. For example, the intruder encryption capability $[-(K), -(M), +(e(K,M)]$ produces the following three rewrite rules:

$$\begin{array}{l}\{SS \,\&\, \{K{\in}\mathcal{I}, IK\} \,\&\, \langle N\rangle\} \\ \rightarrow \{SS \,\&\, [-(K)] \,\&\, \{K{\in}\mathcal{I}, IK\} \,\&\, \langle N\rangle\}\end{array}$$

$$\begin{array}{l}\{SS \,\&\, [-(K)] \,\&\, \{M{\in}\mathcal{I}, IK\} \,\&\, \langle N\rangle\} \\ \rightarrow \{SS \,\&\, [-(K), -(M)] \,\&\, \{M{\in}\mathcal{I}, IK\} \,\&\, \langle N\rangle\}\end{array}$$

$$\begin{array}{l}\{SS \,\&\, [-(K), -(M)] \,\&\, \{IK\} \,\&\, \langle N\rangle\} \\ \rightarrow \{SS \,\&\, [-(K), -(M), +(e(K,M)] \,\& \\ \quad \{e(K,M){\in}\mathcal{I}, IK\} \,\&\, \langle N\rangle\}\end{array}$$

The sets of rewrite rules for output messages are generated as follows, note that some rewrite rules are conditional:

$$\left\{\begin{array}{l}\forall\, [u_1^{\pm}, \ldots, u_{j-1}^{\pm}, u_j^{+}, u_{j+1}^{\pm}, \ldots, u_n^{\pm}] \in \mathcal{P} \wedge j > 1 : \\ \{SS \,\&\, \{IK\} \,\&\, [u_1^{\pm}, \ldots, u_{j-1}^{\pm}] \,\&\, \langle N\rangle\} \\ \rightarrow \\ \{SS \,\&\, \{u_j{\uparrow}_N^M{\in}\mathcal{I}, IK\} \,\&\, [u_1^{\pm}, \ldots, u_{j-1}^{\pm}, (u_j{\uparrow}_N^M)^+] \,\&\, \langle M\rangle\} \\ \qquad \mathsf{IF}\ (u_j{\uparrow}_N^M{\in}\mathcal{I}) \notin IK\end{array}\right\} \tag{5.1}$$

$$\left\{\begin{array}{l}\forall\, [u_1^{\pm}, \ldots, u_{j-1}^{\pm}, u_j^{+}, u_{j+1}^{\pm}, \ldots, u_n^{\pm}] \in \mathcal{P} \wedge j > 1 : \\ \{SS \,\&\, \{IK\} \,\&\, [u_1^{\pm}, \ldots, u_{j-1}^{\pm}] \,\&\, \langle N\rangle\} \\ \rightarrow \{SS \,\&\, \{IK\} \,\&\, [u_1^{\pm}, \ldots, u_{j-1}^{\pm}, (u_j{\uparrow}_N^M)^+] \,\&\, \langle M\rangle\}\end{array}\right\} \tag{5.2}$$

$$\left\{\begin{array}{l}\forall\, [u_1^{+}, \ldots, u_n^{\pm}] \in \mathcal{P} : \\ \{SS \,\&\, \{IK\} \,\&\, \langle N\rangle\} \\ \rightarrow \{SS \,\&\, [(u_1{\uparrow}_N^M)^+] \,\&\, \{u_1{\uparrow}_N^M{\in}\mathcal{I}, IK\} \,\&\, \langle M\rangle\} \\ \qquad \mathsf{IF}\ (u_1{\in}\mathcal{I}{\uparrow}_N^M) \notin IK\end{array}\right\} \tag{5.3}$$

$$\left\{\begin{array}{l}\forall\, [u_1^{+}, \ldots, u_n^{\pm}] \in \mathcal{P} : \\ \{SS \,\&\, \{IK\} \,\&\, \langle N\rangle\} \rightarrow \{SS \,\&\, [(u_1{\uparrow}_N^M)^+] \,\&\, \{IK\} \,\&\, \langle M\rangle\}\end{array}\right\} \tag{5.4}$$

Each transition rule of type (5.1) accepts output messages and the intruder's knowledge is positively increased, while each transition rule of type (5.2) simply accepts output messages without modifying the intruder's knowledge. Each transition rule in (5.3) and (5.4) introduces a new strand beginning with an output message. Similarly, rules of type (5.3) introduce a new strand and the intruder's knowledge is positively increased, whereas rules of type (5.4) introduce a new strand but the intruder's knowledge is not increased [1] .

The following set of rewrite rules describes the general state transition for a negative message, generating specific rewrite rules according to the protocol strands:

$$\left\{\begin{array}{l}\forall\, [u_1^{\pm}, \ldots, u_{j-1}^{\pm}, u_j^{-}, u_{j+1}^{\pm}, \ldots, u_n^{\pm}] \in \mathcal{P} \wedge j > 1 : \\ \{SS \,\&\{u_j{\in}\mathcal{I}, IK\} \,\&\, [u_1^{\pm}, \ldots, u_{j-1}^{\pm}] \,\&\, \langle N\rangle\} \\ \rightarrow \{SS \,\&\, \{u_j{\in}\mathcal{I}, IK\} \,\&\, [u_1^{\pm}, \ldots, u_{j-1}^{\pm}, u_j^{-}] \,\&\, \langle N\rangle\}\end{array}\right\} \tag{5.5}$$

$$\left\{\begin{array}{l}\forall\, [u_1^{-}, u_2^{\pm}, \ldots, u_n^{\pm}] \in \mathcal{P} : \\ \{SS \,\&\, \{u_1{\in}\mathcal{I}, IK\} \,\&\, \langle N\rangle\} \\ \rightarrow \{SS \;\&\; [u_1^{-}] \,\&\, \{u_1{\in}\mathcal{I}, IK\} \,\&\, \langle N\rangle\}\end{array}\right\} \tag{5.6}$$

Each transition rule in (5.5) and (5.6) accepts input messages if the intruder's knowledge matches them. Note that in (5.6) a new strand is introduced.

Definition 5.3 *Let $\mathcal{P}$ be a protocol with signature $\Sigma_{\mathcal{P}}$ and equational theory $E_{\mathcal{P}}$. We define the* forward rewrite theory characterizing *$\mathcal{P}$ to be $(\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, R_{F\mathcal{P}})$ where $R_{F\mathcal{P}} = \{(5.1) \cup (5.2) \cup (5.3) \cup (5.4) \cup (5.5) \cup (5.6)\}$.*

The forwards execution of a protocol induces a transition system as follows.

[1] Note that the use of the global counter for new principal names in previous rules has to take into account when one of those principals is indeed the intruder; see Example 5.5 for the case in which the intruder impersonates Bob.

Definition 5.4 (Transition System induced by a Protocol) *Given a protocol $\mathcal{P}$ characterized by the forward rewrite theory $(\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, R_{F\mathcal{P}})$ such that $(\Sigma_{\mathcal{P}}, B, E_0)$ is a decomposition of $(\Sigma, E_{\mathcal{P}})$, we can associate to it a transition system $\mathcal{L}_{\mathcal{P}}$ whose states are B-equivalence classes of terms in E_0, B-canonical form and whose transitions are of the form:*

$$[t]_B \rightarrow [t']_B$$

where $t \rightarrow_{R_{F\mathcal{P}},B} u$ and $t' =_B u{\downarrow}_{E_0,B}$.

Unlike the case with process calculi, no information is removed from a state and the history of previous actions can be recovered from a state. Therefore there is no need to record this information in the transition system through labels in order to obtain a labeled transition system. However, labels can be added if desired (e.g. as a compact way of encoding essential information).

Example 5.5 Let us show the rewrite rules generated for the protocol with Diffie-Hellman exponentation of Example 3.1 in Page 45. The rewrite rules associated to Alice's strand in the forwards semantics of our running example are as follows, where the increment of the global counter can be clearly identified. Alice's strand is defined as

$$\begin{array}{c} ::: r, r' ::: [+(A; B; exp(g, n(A, r))), -(B; A; X), \\ +(e(exp(X, n(A, r)), sec(A, r')))] \end{array}$$

and the rewrite rules associated to it are as follows:

$$\begin{array}{l} \{SS \;\&\; \{IK\} \;\&\; \langle N \rangle\} \\ \rightarrow \{SS \;\&\; [+(N; N+1; exp(g, n(N, N+2)))] \;\& \\ \quad \{(N; N+1; exp(g, n(N, N+2))) \in \mathcal{I}, IK\} \;\&\; \langle N+3 \rangle\} \end{array}$$

$$\begin{array}{l} \{SS \;\&\; [+(A; B; exp(g, n(A, R)))] \;\&\{(B; A; X) \in \mathcal{I}, IK\} \;\&\; \langle N \rangle\} \\ \rightarrow \{SS \;\&\; [+(A; B; exp(g, n(A, R))), -(B; A; X)] \;\& \\ \quad \{(B; A; X) \in \mathcal{I}, IK\} \;\&\; \langle N \rangle\} \end{array}$$

$$\begin{array}{l} \{SS \;\&\; [+(A; B; exp(g, n(A, R))), -(B; A; X)] \;\&\{IK\} \;\&\; \langle N \rangle\} \\ \rightarrow \{SS \;\&\; [+(A; B; exp(g, n(A, R))), -(B; A; X), \\ \qquad +(e(exp(X, n(A, R)), sec(A, N)))] \;\& \\ \quad \{e(exp(X, n(A, R)), sec(A, N))) \in \mathcal{I}, IK\} \;\&\; \langle N+1 \rangle\} \end{array}$$

When the intuder impersonates Bob, the first rule is:

$$\begin{array}{l}\{SS \;\&\; \{IK\} \;\&\; \langle N\rangle\}\\ \rightarrow \{SS \;\&\; [+(N; i; exp(g, n(N, N+1)))] \;\&\;\\ \quad \{(N; i; exp(g, n(N, N+1)))\in\mathcal{I}, IK\} \;\&\; \langle N+2\rangle\}\end{array}$$

where i is a constant denoting the intruder's name. Note that it is not necessary to duplicate the other two rules.

Bob's strand is defined as

$$\begin{array}{c}::: r, r' ::: [-(A; B; Y), +(B; A; exp(g, n(B, r'')))),\\ -(e(exp(Y, n(B, r'')), Sr))]\end{array}$$

and the rewrite rules associated to it are as follows:

$$\begin{array}{l}\{SS \;\&\; \{A; B; Y\in\mathcal{I}, IK\} \;\&\; \langle N\rangle\}\\ \rightarrow \{SS \;\&\; [-(A; B; Y), +(B; A; exp(g, n(B, N)))] \;\&\;\\ \quad \{A; B; Y\in\mathcal{I}, (B; A; exp(g, n(B, N)))\in\mathcal{I}, IK\} \;\&\; \langle N+1\rangle\}\end{array}$$

$$\begin{array}{l}\{SS \;\&\; [-(A; B; Y), +(B; A; exp(g, n(B, J)))] \;\&\;\\ \quad \{e(exp(g, NS * n(B, J)), Sr)\in\mathcal{I}, IK\} \;\&\; \langle N\rangle\}\\ \rightarrow \{SS \;\&\; [-(A; B; Y), +(B; A; exp(g, n(B, J))),\\ \qquad -(e(exp(g, NS * n(B, J)), Sr))] \;\&\;\\ \quad \{e(exp(g, NS * n(B, J)), Sr)\in\mathcal{I}, IK\} \;\&\; \langle N\rangle\}\end{array}$$

Let us now show the rewrite rules generated for each intruder action. The strands denoting the intruder's ability to perform the inverses of the concatenation are defined as follows:

$$\begin{array}{c}:: nil :: [-(M1; M2), +(M1)]\\ :: nil :: [-(M1; M2), +(M2)]\end{array}$$

We show below the rewrite rules associated to these two strands:

$$\begin{array}{l}\{SS \;\&\; \{(M1; M2)\in\mathcal{I}, IK\} \;\&\; \langle N\rangle\}\\ \rightarrow \{SS \;\&\; [-(M1; M2), +(M1)] \;\&\; \{(M1; M2)\in\mathcal{I}, M1\in\mathcal{I}, IK\} \;\&\; \langle N\rangle\}\end{array}$$

$$\begin{array}{l}\{SS \;\&\; \{(M1; M2)\in\mathcal{I}, IK\} \;\&\; \langle N\rangle\}\\ \rightarrow \{SS \;\&\; [-(M1; M2), +(M2)] \;\&\; \{(M1; M2)\in\mathcal{I}, M2\in\mathcal{I}, IK\} \;\&\; \langle N\rangle\}\end{array}$$

The intruder strand denoting its ability to concatenate two messages $M1$ and $M2$ is defined as follows:

$$:: nil :: [-(M1), -(M2), +(M1; M2)]$$

and its associated rewrite rules are as shown below:

$$\{SS \;\&\; \{M1 \in \mathcal{I}, IK\} \;\&\; \langle N \rangle\}$$
$$\rightarrow \{SS \;\&\; [-(M1)] \;\&\; \{M1 \in \mathcal{I}, IK\} \;\&\; \langle N \rangle\}$$

$$SS \;\&\; [-(M1)] \;\&\; \{M2 \in \mathcal{I}, IK\} \;\&\; \langle N \rangle\}$$
$$\rightarrow \{SS \;\&\; [-(M1), -(M2)] \;\&\; \{M2 \in \mathcal{I}, IK\} \;\&\; \langle N \rangle\}$$

$$\{SS \;\&\; [-(M1), -(M2)] \;\&\; \{IK\} \;\&\; \langle N \rangle\}$$
$$\rightarrow \{SS \;\&\; [-(M1), -(M2), +(M1; M2)] \;\&\; \{(M1; M2) \in \mathcal{I}, IK\} \;\&\; \langle N \rangle\}$$

In this protocol the intruder is allowed to encrypt and decrypt a message M with a given key Ke, which is denoted by the strands shown below:

$$:: nil :: [-(M), -(Ke), +(e(Ke, M))]$$
$$:: nil :: [-(M), -(Ke), +(d(Ke, M))]$$

The rewrite rules generated for these two strands are as follows:

$$\{SS \;\&\; \{M \in \mathcal{I}, IK\} \;\&\; \langle N \rangle\}$$
$$\rightarrow \{SS \;\&\; [-(M)] \;\&\; \{M \in \mathcal{I}, IK\} \;\&\; \langle N \rangle\}$$

$$\{SS \;\&\; [-(M)] \;\&\; \{Ke \in \mathcal{I}, IK\} \;\&\; \langle N \rangle\}$$
$$\rightarrow \{SS \;\&\; [-(M), -(Ke)] \;\&\; \{(Ke \in \mathcal{I}), IK\} \;\&\; \langle N \rangle\}$$

$$\{SS \;\&\; [-(M), -(Ke)] \;\&\; \{IK\} \;\&\; \langle N \rangle\}$$
$$\rightarrow \{SS \;\&\; [-(M), -(Ke), +(e(Ke, M))] \;\&\; \{e(Ke, M) \in \mathcal{I}, IK\} \;\&\; \langle N \rangle\}$$

$$\{SS \;\&\; \{(M) \in \mathcal{I}, IK\} \;\&\; \langle N \rangle\}$$
$$\rightarrow \{SS \;\&\; [-(M)] \;\&\; \{M \in \mathcal{I}, IK\} \;\&\; \langle N \rangle\}$$

$$\{SS \;\&\; [-(M)] \;\&\; \{Ke \in \mathcal{I}, IK\} \;\&\; \langle N \rangle\}$$
$$\rightarrow \{SS \;\&\; [-(M), -(Ke)] \;\&\; \{(Ke \in \mathcal{I}), IK\} \;\&\; \langle N \rangle\}$$

$$\{SS \;\&\; [-(M), -(Ke)] \;\&\; \{IK\} \;\&\; \langle N \rangle\}$$
$$\rightarrow \{SS \;\&\; [-(M), -(Ke), +(e(Ke, M))] \;\&\; \{d(Ke, M) \in \mathcal{I}, IK\} \;\&\; \langle N \rangle\}$$

The intruder's ability to perform the product of two exponents *NS1* ,and *NS2* is denoted by the strand shown below:

$$:: nil :: [-(NS1), -(NS2), +(NS1 * NS2))]$$

The rewrite rules associated to this strand are as follows:

$$\{SS \;\&\; \{(NS1)\in\mathcal{I}, IK\} \;\&\; \langle N\rangle\}$$
$$\rightarrow \{SS \;\&\; [-(NS1)] \;\&\; \{NS1\in\mathcal{I}, IK\} \;\&\; \langle N\rangle\}$$

$$SS \;\&\; [-(NS1)] \;\&\; \{(NS2)\in\mathcal{I}, IK\} \;\&\; \langle N\rangle\}$$
$$\rightarrow \{SS \;\&\; [-(NS1), -(NS2)] \;\&\; \{NS2\in\mathcal{I}, IK\} \;\&\; \langle N\rangle\}$$

$$\{SS \;\&\; [-(NS1), -(NS2)] \;\&\; \{IK\} \;\&\; \langle N\rangle\}$$
$$\rightarrow \{SS \;\&\; [-(NS1), -(NS2), +(NS1 * NS2)] \;\&$$
$$\{(NS1 * NS2)\in\mathcal{I}, IK\} \;\&\; \langle N\rangle\}$$

The strand denoting the intruder's ability to perform a Diffie-Hellman exponentiation of GE to the power of NS is as shown below:

$$:: nil :: [-(GE), -(NS), +(exp(GE, NS))]$$

The rewrite rules associated to this strand are as follows:

$$\{SS \;\&\; \{GE\in\mathcal{I}, IK\} \;\&\; \langle N\rangle\}$$
$$\rightarrow \{SS \;\&\; [-(GE)] \;\&\; \{GE\in\mathcal{I}, IK\} \;\&\; \langle N\rangle\}$$

$$SS \;\&\; [-(GE)] \;\&\; \{NS\in\mathcal{I}, IK\} \;\&\; \langle N\rangle\}$$
$$\rightarrow \{SS \;\&\; [-(GE), -(NS)] \;\&\; \{NS\in\mathcal{I}, IK\} \;\&\; \langle N\rangle\}$$

$$\{SS \;\&\; [-(GE), -(NS)] \;\&\; \{IK\} \;\&\; \langle N\rangle\}$$
$$\rightarrow \{SS \;\&\; [-(GE), -(NS), +(exp(GE, NS))] \;\&$$
$$\{exp(GE, NS)\in\mathcal{I}, IK\} \;\&\; \langle N\rangle\}$$

The intruder capability to generate the g constant, denoted by the strand shown below:

$$:: nil :: [+(g)]$$

has associated the following rewrite rule:

$$\{SS \;\&\; \{IK\} \;\&\; \langle N \rangle\} \rightarrow \{SS \;\&\; [+(g)] \;\&\; \{g{\in}\mathcal{I}, IK\} \;\&\; \langle N \rangle\}$$

Finally, for the intruder's capability to generate nonces and any arbitray name A, denoted by the two strands shown below, respectively:

$$:: r :: [+(n(i,r))]$$
$$:: r :: [+(A)]$$

the following rewrite rules are generated, respectively

$$\{SS \;\&\; \{IK\} \;\&\; \langle N \rangle\}$$
$$\rightarrow \{SS \;\&\; [+(n(i,N))] \;\&\; \{n(i,N){\in}\mathcal{I}, IK\} \;\&\; \langle N+1 \rangle\}$$

$$\{SS \;\&\; \{IK\} \;\&\; \langle N \rangle\}$$
$$\rightarrow \{SS \;\&\; [+(name(N))] \;\&\; \{name(N){\in}\mathcal{I}, IK\} \;\&\; \langle N+1 \rangle\}$$

Note that, as explained above, names and fresh variables are treated as numeric constants.

■

5.4 Soundness and Completeness of the Forwards Semantics

In the previous section we defined the rewriting-based forwards semantics for Maude-NPA. Now we need to prove that the backwards operational semantics of Maude-NPA given in Section 3.4 is sound and complete w.r.t. this semantics. Note that throughout this section $s \rightarrow s'$ denotes a *(forward) rewriting* step using a rule of $R_{F\mathcal{P}}$, whereas $S' \rightsquigarrow S$ denotes a *backwards narrowing* step using a rule of $R_{B\mathcal{P}}$. We first introduce some definitions and concepts that will be used in these proofs.

First, we define what a symbolic state is, i.e., a state with variables.

Definition 5.6 (Symbolic $\mathcal{P}$-state) *Given a protocol $\mathcal{P}$, a* symbolic

$\mathcal{P}$-state S is a term of the form:

$$\begin{aligned} S = \{ &:: r_{1_1}, \ldots, r_{m_1} :: [u^{\pm}_{1_1}, \ldots u^{\pm}_{i_1-1} \mid u^{\pm}_{i_1}, \ldots, u^{\pm}_{n_1}] \;\&\\ &\vdots\\ &:: r_{1_k}, \ldots, r_{m_k} :: [u^{\pm}_{1_k}, \ldots, u^{\pm}_{i_k-1} \mid u^{\pm}_{i_k}, \ldots, u^{\pm}_{n_k}] \;\&\; SS\\ &\{w_1 \in \mathcal{I}, \ldots, w_m \in \mathcal{I},\; w'_1 \notin \mathcal{I}, \ldots, w'_{m'} \notin \mathcal{I}, IK\}\} \end{aligned}$$

where for each $1 \leqslant j \leqslant k$, there exists a strand $[m^{\pm}_{1_j}, \ldots m^{\pm}_{i_j-1}, m^{\pm}_{i_j}, \ldots, m^{\pm}_{n_j}] \in \mathcal{P}$ and a substitution $\rho_j : \mathcal{X} \to \mathcal{T}_\Sigma(\mathcal{X})$ such that $m_{1_j}\rho_j =_{E_\mathcal{P}} u_{1_j}$, $\ldots$, $m_{n_j}\rho_j =_{E_\mathcal{P}} u_{n_j}$, SS is a variable denoting a (possibly empty) set of strands, and IK is a variable denoting a (possibly empty) set of intruder's knowledge facts.

Second, we define what a ground state is, i.e., a state without variables.

Definition 5.7 (Ground $\mathcal{P}$-state) *Given a protocol $\mathcal{P}$, a* ground $\mathcal{P}$*-state s is a term without variables of the form:*

$$\begin{aligned} s = \{&[u^{\pm}_{1_1}, \ldots u^{\pm}_{i_1-1}] \;\&\cdots\&\; [u^{\pm}_{1_k}, \ldots, u^{\pm}_{i_k-1}] \;\&\\ &\{w_1 \in \mathcal{I}, \ldots, w_m \in \mathcal{I}\} \;\&\; \langle J \rangle\} \end{aligned}$$

where for each $1 \leqslant j \leqslant k$, there exists a strand $[m^{\pm}_{1_j}, \ldots m^{\pm}_{i_j-1}, m^{\pm}_{i_j}, \ldots, m^{\pm}_{n_j}] \in \mathcal{P}$ and a substitution $\rho_j : \mathcal{X} \to \mathcal{T}_\Sigma$ such that $m_{1_j}\rho_j =_{E_\mathcal{P}} u_{1_j}$, $\ldots$, $m_{i_j}\rho_j =_{E_\mathcal{P}} u_{i_j}$.

Third, we define a suitable instantiation relation between symbolic and ground states.

Definition 5.8 (Lifting relation) *Given a symbolic $\mathcal{P}$-state S and a ground state s we say that s* lifts *to S, or that S* instantiates *to s with a* grounding *substitution $\theta : (\mathcal{V}ar(S) - \{SS, IK\}) \to \mathcal{T}_\Sigma$, writen $S >^\theta s$ iff*

- *for each strand $:: r_1, \ldots, r_m :: [u^{\pm}_1, \ldots u^{\pm}_{i-1} \mid u^{\pm}_i, \ldots, u^{\pm}_n]$ in S, there exists a strand $[v^{\pm}_1, \ldots v^{\pm}_{i-1}]$ in s such that $\forall 1 \leqslant j \leqslant i-1$, $v_j =_{E_\mathcal{P}} u_j\theta$.*
- *for each positive intruder fact $w \in \mathcal{I}$ in S, there exists a positive intruder fact $w' \in \mathcal{I}$ in s such that $w' =_{E_\mathcal{P}} w\theta$, and*

- *for each negative intruder fact $w \notin \mathcal{I}$ in S, there is no positive intruder fact $w' \in \mathcal{I}$ in s such that $w' =_{E_\mathcal{P}} w\theta$.*

Let us now prove that narrowing with the backwards rewrite theory $(\Sigma_\mathcal{P}, E_\mathcal{P}, R^{-1}_{B\mathcal{P}})$ is complete with respect to rewriting with the forwards rewrite theory $(\Sigma_\mathcal{P}, E_\mathcal{P}, R_{F\mathcal{P}})$. First, the lemma below shows how the lifting of a ground term to a symbolic state induces a lifting of a forward rewriting step in the forwards semantics to a backwards narrowing step in the backwards semantics. This will be used to prove Theorem 5.10, which allows us to lift a rewriting sequence in the forwards semantics to a narrowing sequence in the backwards semantics.

Lemma 5.9 (Lifting Lemma) *Given a protocol $\mathcal{P}$, two states s and s', a $\mathcal{P}$-symbolic state S' and a substitution θ' s.t. $s \to s'$ and $S' >^{\theta'} s'$, then there exist a $\mathcal{P}$-symbolic state S and a substitution θ s.t $S >^{\theta} s$ and either $S' \rightsquigarrow S$ or $S = S'$.*

Proof. First of all, all the forward rewriting rules act on the ground state s' by either adding one more element to an existing strand (rules (5.1), (5.2), and (5.5)), adding a positive fact to the intruder knowledge (rules (5.1), and (5.3)), adding a new strand (rules (5.3), (5.4), and (5.5)), or repeating a positive intruder fact that is already in s' (rules (5.5) and (5.6)). This allows us to identify six cases for the grounding substitution θ' of S' into s', depending upon whether the grounding substitution of S under θ contains the relevant strands and positive intruder facts.

a) There is a strand $[u_1^\pm, \ldots, u_{i-1}^\pm, u_i^\pm, \ldots, u_n^\pm]$ in $\mathcal{P}$, $n \geqslant 1$, $1 \leqslant i \leqslant n$, and a substitution ρ such that $[u_1^\pm, \ldots, u_{i-1}^\pm, u_i^\pm]\rho$ is a strand in s', $[u_1^\pm, \ldots, u_{i-1}^\pm \mid u_i^\pm, \ldots, u_n^\pm]\rho$ is a strand in $S'\theta'$, and $u_i\rho \in \mathcal{I}$ appears in the intruder knowledge of $S'\theta'$. This is valid for rules in sets (5.1), (5.3), (5.5), and (5.6). If $i > 1$, then we also know that $[u_1^\pm, \ldots, u_{i-1}^\pm]\rho$ is a strand in s.

b) There is a strand $[u_1^\pm, \ldots, u_{i-1}^\pm, u_i^\pm, \ldots, u_n^\pm]$ in $\mathcal{P}$, $n \geqslant 1$, $1 \leqslant i \leqslant n$, and a substitution ρ such that $[u_1^\pm, \ldots, u_{i-1}^\pm, u_i^\pm]\rho$ is a strand in s', $[u_1^\pm, \ldots, u_{i-1}^\pm \mid u_i^\pm, \ldots, u_n^\pm]\rho$ is a strand in $S'\theta'$, but $u_i\rho \in \mathcal{I}$ does not appear in the intruder knowledge of $S'\theta'$. This is valid for rules in sets (5.1), (5.3), (5.5), and (5.6). If $i > 1$, then we also know that $[u_1^\pm, \ldots, u_{i-1}^\pm]\rho$ is a strand in s.

c) There is a strand $[u_1^{\pm}, \ldots, u_{i-1}^{\pm}, u_i^{\pm}, \ldots, u_n^{\pm}]$ in $\mathcal{P}$, $n \geqslant 1$, $1 \leqslant i \leqslant n$, and a substitution ρ such that $[u_1^{\pm}, \ldots, u_{i-1}^{\pm}, u_i^{\pm}]\rho$ is a strand in s', $u_i\rho{\in}\mathcal{I}$ appears in the intruder knowledge of $S'\theta'$, but $[u_1^{\pm}, \ldots, u_{i-1}^{\pm} \mid u_i^{\pm}, \ldots, u_n^{\pm}]\rho$ is not a strand in $S'\theta'$. This is valid for rules in sets (5.1), (5.3), (5.5), and (5.6). If $i > 1$, then we also know that $[u_1^{\pm}, \ldots, u_{i-1}^{\pm}]\rho$ is a strand in s.

d) There is a strand $[u_1^{\pm}, \ldots, u_{i-1}^{\pm}, u_i^{\pm}, \ldots, u_n^{\pm}]$ in $\mathcal{P}$, $n \geqslant 1$, $1 \leqslant i \leqslant n$, and a substitution ρ such that $[u_1^{\pm}, \ldots, u_{i-1}^{\pm}, u_i^{\pm}]\rho$ is a strand in s' but $u_i\rho{\in}\mathcal{I}$ does not appear in the intruder knowledge of $S'\theta'$ and $[u_1^{\pm}, \ldots, u_{i-1}^{\pm} \mid u_i^{\pm}, \ldots, u_n^{\pm}]\rho$ is not a strand in $S'\theta'$. This is valid for rules in sets (5.1), (5.3), (5.5), and (5.6). If $i > 1$, then we also know that $[u_1^{\pm}, \ldots, u_{i-1}^{\pm}]\rho$ is a strand in s.

e) There is a strand $[u_1^{\pm}, \ldots, u_{i-1}^{\pm}, u_i^{\pm}, \ldots, u_n^{\pm}]$ in $\mathcal{P}$, $n \geqslant 1$, $1 \leqslant i \leqslant n$, and a substitution ρ such that $[u_1^{\pm}, \ldots, u_{i-1}^{\pm}, u_i^{\pm}]\rho$ is a strand in s' and $[u_1^{\pm}, \ldots, u_{i-1}^{\pm} \mid u_i^{\pm}, \ldots, u_n^{\pm}]\rho$ is a strand in $S'\theta'$. This is valid for rules in sets (5.2) and (5.4). If $i > 1$, then we also know that $[u_1^{\pm}, \ldots, u_{i-1}^{\pm}]\rho$ is a strand in s.

f) There is a strand $[u_1^{\pm}, \ldots, u_{i-1}^{\pm}, u_i^{\pm}, \ldots, u_n^{\pm}]$ in $\mathcal{P}$, $n \geqslant 1$, $1 \leqslant i \leqslant n$, and a substitution ρ such that $[u_1^{\pm}, \ldots, u_{i-1}^{\pm}, u_i^{\pm}]\rho$ is a strand in s' but $[u_1^{\pm}, \ldots, u_{i-1}^{\pm} \mid u_i^{\pm}, \ldots, u_n^{\pm}]\rho$ is not a strand in $S'\theta'$. This is valid for rules in sets (5.2) and (5.4). If $i > 1$, then we also know that $[u_1^{\pm}, \ldots, u_{i-1}^{\pm}]\rho$ is a strand in s.

Now, we consider each forward rewrite rule application in the step $s \to s'$.

- Given states s and s' such that $s \to s'$ using a rule in set (5.1), then there exist a substitution τ, variables SS' and IK', and a strand $[u_1^{\pm}, \ldots, u_{j-1}^{\pm}, u_j^{+}, u_{j+1}^{\pm}, \ldots, u_n^{\pm}]$ in $\mathcal{P}$ such that $s = \{SS'\tau \,\&\, \{IK'\tau\} \,\&\, [(u_1\tau)^{\pm}, \ldots, (u_{j-1}\tau)^{\pm}]\}$, and $s' = \{SS'\tau \,\&\, \{(u_j\tau){\in}\mathcal{I}, IK'\tau\} \,\&\, [(u_1\tau)^{\pm}, \ldots, (u_{j-1}\tau)^{\pm}, (u_j\tau)^{+}]\}$ and $(u_j\tau){\in}\mathcal{I}$ appears in $IK'\tau$. Since there exists a substitution θ' s.t $S' >^{\theta'} s'$, we consider the four applicable cases for substitution θ':

  - *Case a)* Both the strand and the intruder fact appear in $S'\theta'$ and thus we can perform a backwards narrowing step from S' with rule (3.3) to obtain a state S, i.e., $S' \rightsquigarrow S$. Since

there is no extra variable in the rule, we have that the same substitution θ' is valid for S and $S >^{\theta'} s$.

- *Case b)* The strand appears in $S'\theta'$ but not the intruder fact. We also perform a backwards narrowing step from S' with rule (3.3) to obtain a state S, i.e., $S' \leadsto_{\sigma} S$. But the variable IK in state S' gets instantiated $\sigma = \{IK \mapsto w{\in}\mathcal{I}, IK''\}$ in such a way that $w\theta' =_{E_{\mathcal{P}}} u_j\tau$. Since there is no extra variable in the rule, again $S >^{\theta'} s$.
- *Case c)* The intruder fact appears in $S'\theta'$ but not the strand. Here we perform a backwards narrowing step from S' with a rule in set (3.4) to obtain a state S, i.e., $S' \leadsto S$. This rule introduces a new strand into the symbolic state S, i.e., there is a substitution γ such that $[(u_1\gamma)^{\pm}, \ldots, (u_{j-1}\gamma)^{\pm} \mid (u_j\gamma)^{+}, (u_{j+1}\gamma)^{\pm}, \ldots, (u_n\gamma)^{\pm}]$ is a strand in S.

  Note that this new strand contains variables but there is a substitution θ such that $S >^{\theta} s$, since $[(u_1\gamma\theta)^{\pm}, \ldots, (u_{j-1}\gamma\theta)^{\pm}]$ corresponds to $[(u_1\tau)^{\pm}, \ldots, (u_{j-1}\tau)^{\pm}]$.
- *Case d)* The strand and the intruder fact do not appear in $S'\theta'$. This case is very simple, since θ' makes valid S' as a symbolic state of s, i.e., $S = S'$ and $S' >^{\theta'} s$.

- Given states s and s' such that $s \to s'$ using a rule in set (5.2), then there exist a substitution τ, variables SS' and IK', and a strand $[u_1^{\pm}, \ldots, u_{j-1}^{\pm}, u_j^{+}, u_{j+1}^{\pm}, \ldots, u_n^{\pm}]$ in $\mathcal{P}$ such that $s = \{SS'\tau \,\&\, \{IK'\tau\} \,\&\, [(u_1\tau)^{\pm}, \ldots, (u_{j-1}\tau)^{\pm}]\}$, and $s' = \{SS'\tau \,\&\, \{IK'\tau\} \,\&\, [(u_1\tau)^{\pm}, \ldots, (u_{j-1}\tau)^{\pm}, (u_j\tau)^{+}]\}$. Since there exists a substitution θ' s.t $S' >^{\theta'} s'$, we consider the two applicable cases for substitution θ':

  - *Case e)* The strand appears in $S'\theta'$ and thus we can perform a backwards narrowing step from S' with rule (3.2) to obtain a state S, i.e., $S' \leadsto S$. Since there is no extra variable in the rule, we have that the same substitution θ' is valid for S and $S >^{\theta'} s$.
  - *Case f)* The strand does not appear in $S'\theta'$. This case is very simple, since θ' makes valid S' as a symbolic state of s, i.e., $S = S'$ and $S' >^{\theta'} s$.

- Given states s and s' such that $s \rightarrow s'$ using a rule in set (5.3), then there exist a substitution τ, variables SS' and IK', and a strand $[u_1^{\pm}, \ldots, u_n^{\pm}]$ in $\mathcal{P}$ such that $s' = \{SS'\tau \,\&\, \{(u_1\tau) \in \mathcal{I}, IK'\tau\} \,\&\, [(u_1\tau)^{\pm}]\}$ and $(u_j\tau) \in \mathcal{I}$ does not appear in $IK'\tau$. This is similar to the case above of a rule in set (5.1).

- Given states s and s' such that $s \rightarrow s'$ using a rule in set (5.4), then there exist a substitution τ, variables SS' and IK', and a strand $[u_1^{\pm}, \ldots, u_n^{\pm}]$ in $\mathcal{P}$ such that $s' = \{SS'\tau \,\&\, \{IK'\tau\} \,\&\, [(u_1\tau)^{\pm}]\}$. This is similar to the case above of a rule in set (5.2).

- Given states s and s' such that $s \rightarrow s'$ using a rule in set (5.5), then there exist a substitution τ, variables SS' and IK', and a strand $[u_1^{\pm}, \ldots, u_{j-1}^{\pm}, u_j^{+}, u_{j+1}^{\pm}, \ldots, u_n^{\pm}]$ in $\mathcal{P}$ such that $s = \{SS'\tau \,\&\, \{(u_j\tau) \in \mathcal{I}, IK'\tau\} \,\&\, [(u_1\tau)^{\pm}, \ldots, (u_{j-1}\tau)^{\pm}]\}$, and $s' = \{SS'\tau \,\&\, \{(u_j\tau) \in \mathcal{I}, IK'\tau\} \,\&\, [(u_1\tau)^{\pm}, \ldots, (u_{j-1}\tau)^{\pm}, (u_j\tau)^{-}]\}$. Since there exists a substitution θ' s.t $S' >^{\theta'} s'$, we consider the four applicable cases for substitution θ':

  - *Case a)* Both the strand and the intruder fact appear in $S'\theta'$ and thus we can perform a backwards narrowing step from S' with rule (3.1) to obtain a state S, i.e., $S' \rightsquigarrow S$. Since there is no extra variable in the rule, we have that the same substitution θ' is valid for S and $S >^{\theta'} s$.
  - *Case b)* The strand appears in $S'\theta'$ but not the intruder fact. We also perform a backwards narrowing step from S' with rule (3.1) to obtain a state S, i.e., $S' \rightsquigarrow_{\sigma} S$. But the variable IK in state S' gets instantiated $\sigma = \{IK \mapsto w \in \mathcal{I}, IK''\}$ in such a way that $w\theta' =_{E_{\mathcal{P}}} u_j\tau$. Since there is no extra variable in the rule, again $S >^{\theta'} s$.
  - *Case c)* The intruder fact appears in $S'\theta'$ but not the strand. This case is very simple, since θ' makes valid S' as a symbolic state of s, i.e., $S = S'$ and $S' >^{\theta'} s$.
  - *Case d)* The strand and the intruder fact do not in $S'\theta'$. This case is very simple, since θ' makes valid S' as a symbolic state of s, i.e., $S = S'$ and $S' >^{\theta'} s$.

- Given states s and s' such that $s \rightarrow s'$ using a rule in set (5.6), then there exist a substitution τ, variables SS' and IK', and a strand $[u_1^{\pm}, \ldots, u_n^{\pm}]$ in $\mathcal{P}$ such that $s' = \{SS'\tau \,\&\, \{(u_1\tau) \in \mathcal{I}, IK'\tau\} \,\&\, [(u_1\tau)^-]\}$. This is similar to the case above of a rule in set (5.5).

This concludes the proof. □

The following theorem states that the symbolic reachability analysis is *complete* with respect to the forwards rewriting-based semantics, i.e., any concrete attack state s, matching an attack pattern S and reachable by the forwards semantics from a concrete initial state s_0 can be *found* by backwards symbolic reachability analysis from the attack pattern S. Its proof is a straightforward corollary of Lemma 5.9.

Theorem 5.10 (Completeness) *Given a protocol $\mathcal{P}$, two ground states s, s_0, a symbolic $\mathcal{P}$-state S, a substitution θ s.t. (i) s_0 is an initial state, (ii) $s_0 \rightarrow^n s$, and (iii) $S >^{\theta} s$ then there exist a symbolic initial $\mathcal{P}$-state S_0, two substitutions μ and θ', and $k \leqslant n$, s.t. $S \overset{k}{\leadsto}_{\mu} S_0$, and $S_0 >^{\theta'} s_0$.*

In the following we prove that the backwards rewrite theory $(\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, R_{B\mathcal{P}}^{-1})$ is *sound* with respect to the forward rewrite theory $(\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, R_{F\mathcal{P}})$. That is, we need to show that if we find a symbolic initial state S_0 from a symbolic attack pattern S then, for any concrete initial state s_0 such that $S_0 >^{\theta} s_0$, there is a reachable concrete attack state s such that $s_0 \rightarrow_s^*$ with $S >^{\theta'} s$. We first provide a lemma that says that for any backwards narrowing step there exist a corresponding sequence of forwards rewriting steps.

Lemma 5.11 *Given a protocol $\mathcal{P}$, two symbolic $\mathcal{P}$-states S, S', a ground state s and a substitution θ, if $S' \overset{\mu}{\leadsto} S$ and $S >^{\theta} s$, then there exist a state s' and a substitution θ' such that $s \rightarrow s'$, and $S' >^{\theta'} s'$.*

Proof. Since rewriting is simply a special case of narrowing, the proof of this lemma is simpler than that of Lemma 5.9. We take into account that $S >^{\theta} s$ implies the strand and intruder facts used in the narrowing step $S' \overset{\mu}{\leadsto} S$ are present in s.

- If we use rule (3.1) in $S' \overset{\mu}{\leadsto} S$, then there are associated rules in the sets (5.5) and (5.6).

- If we use rule (3.2) in $S' \overset{\mu}{\rightsquigarrow} S$, then there are associated rules in the sets (5.2) and (5.4).
- If we use rule (3.3) in $S' \overset{\mu}{\rightsquigarrow} S$, then there are associated rules in the sets (5.1) and (5.3).
- And if we use a rule in set (3.4) in $S' \overset{\mu}{\rightsquigarrow} S$, then there are associated rules in the sets (5.1) and (5.3).

Note that substitution θ' is just a restriction of substitution θ, since each backwards narrowing step instantiates some variable or add new terms (possibly with new variables) but never removes any term or variable already present. This concludes the proof. □

The following theorem is a straightforward corollary of Lemma 5.11. It proves that the symbolic reachability analysis is *sound* with respect to the forwards rewriting-based semantics,

Theorem 5.12 (Soundness) *Given a protocol $\mathcal{P}$, two symbolic $\mathcal{P}$-states S_0, S', an initial ground state s_0 and a substitution θ s.t. (i) S_0 is a symbolic initial state, and (ii) $S' \rightsquigarrow^* S_0$, and (iii) $S_0 >^\theta s_0$ then there exist a ground state s' and a substitution θ', s.t. (i) $s_0 \rightarrow^* s'$, and (ii) $S' >^{\theta'} s'$.*

5.5 Experimental Evaluation

We have performed several experiments to evaluate the feasibility of the rewriting-based forwards semantics defined in Section 5.3. We have used four protocols to perform five experiments: (i) the standard Needham-Schroeder protocol (NSPK) [Needham and Schroeder, 1978], (ii) a version of the Needham-Schroeder-Lowe protocol in which one of the concatenation operators is replaced by an exclusive-or, presented in [Sasse et al., 2010] (NSL-XOR), (iii) the Denning-Sacco Symmetric Key protocol [Denning and Sacco, 1981], and (iv) a protocol with Diffie-Hellman exponentiation. More specifically, we have verified both secrecy and authentication properties for NSPK, secrecy properties for XOR-NSL and Diffie-Hellman, and authentication properties for Denning-Sacco. Since the forwards rewrite-based semantics defined in this chapter does not include optimizations to reduce the search space and, therefore, is not

currently possible to obtain a finite search space, we have analyzed insecure protocols, i.e., protocols with known secrecy and/or authentication attacks. The specifications of these protocols can be found at:

`http://www.dsic.upv.es/~sescobar/Maude-NPA/forwards-semantics.html`

In Table 5.1 we compare the results obtained when analyzing the example protocols with the implementation in Maude of the forwards semantics defined in this paper, and the symbolic backwards reachability analysis performed by Maude-NPA. More specifically, in the second and third columns we show the number of rewrite or narrowing steps generated during the forwards or backwards search, respectively. Note that these results provide an experimental validation of the implementation of the forwards semantics defined in this chapter in Maude w.r.t. the symbolic backwards operational semantics of Maude-NPA, since for each protocol the forwards search found the same authentication or secrecy attacks that are found by Maude-NPA.

Results gathered in Table 5.1 show that the length of the forwards semantics analysis is longer than with Maude-NPA. The reason is that Maude-NPA performs very efficient state space reduction optimizations (see Chapter 4) that do not only reduce the search space in terms of the number of generated states, but also in terms of the analysis length. These state space reduction optimizations have not yet been added to the forwards semantics. For example, the Super-Lazy intruder optimization technique (see Chapter 4) allows Maude-NPA to detect that the intruder can trivially learn some messages, making it unnecessary to perform the analysis steps required to generate those messages. Instead, in the current forwards semantics it is necessary to perform each (rewriting) step to generate those messages making the analysis longer.

Table 5.2 gathers for each experiment the number of states generated during the first five steps of the forwards search. The reader can check that the number of generated states is the same for both experiments of the NSPK protocol, since the search space is the same and only the attack being searched is different. We used experimental heuristics to decrease the size of the state space. However systematic study of state space reduction techniques in the forwards semantics is left for future work.

Protocol	Length Forwards	Length Maude-NPA
NSPK-sec	9	7
NSPK-auth	9	7
NSL-XOR	13	8
Denning-Sacco	11	7
Diffie-Hellman	22	12

Table 5.1: Rewrite and Narrowing steps until finding the attack

Protocol	1	2	3	4	5
NSPK-sec	6	20	116	604	3026
NSPK-auth	6	20	116	604	3026
NSL-XOR	7	21	72	218	594
Denning-Sacco	7	28	132	596	2624
Diffie-Hellman	7	22	65	162	354

Table 5.2: States generated in each rewrite step

Summarizing, the results of our early experimental evaluation suggest that, even though its implementation is still at an early stage, the forwards semantics presented in this chapter is feasible and encouraging. However, much work needs to be done, specially with respect to the efficiency of the analysis.

5.6 Conclusions

We realized several benefits from developing the forwards semantics. One is that the forwards semantics can be *executed* and *model checked* in Maude. We have taken advantage of this feature to implement a prototype explicit-state cryptographic protocol model-checker in Maude that, like Maude-NPA, can be used to reason in the presence of different equational theories. Although we have not yet implemented any state space reduction techniques for it —so there is no current way of achieving termination— it has nevertheless been able to automatically find attacks on some simple protocols that use various equational theories. We sum-

marize the experimental results for this prototype in Section 5.5. This work also reduces the gap between the Maude-NPA and the realm of standard model checking, shedding some light on how its internal semantics and the logical reachability analysis correspond to an intuitive forward execution of a protocol with the intruder model.

The definition of this novel rewriting-based protocol analysis is relevant, since the new forwards semantics is directly implementable in rule-based programming languages such as Maude without any need for constraint solving or unification procedures as it is done in most current approaches (see [Chevalier and Rusinowitch, 2008]), including Maude-NPA, allowing us to explore applications such as the simulation of prototypes and reasoning about theories without the finite variant property, which we leave for future work.

Chapter 6

Sequential Protocol Composition in Maude-NPA

In this chapter we provide two different techniques for Maude-NPA to support *dynamic* sequential composition of protocols, i.e., protocols are specified in a modular way and can be composed when desired during the verification process. The first technique does not require modifying the tool, since the sequential composition is performed via a simple program transformation. The second technique consists of extending Maude-NPA with special transition rules, which allows a more efficient verification analysis. Below we explain in more detail the advantages of each technique w.r.t the other.

First, Section 6.1 motivates the analysis of protocol compositions. In Section 6.2 we present some motivating examples of sequential protocol composition, which will be used throughout this chapter as running examples. Section 6.3 provides a formal definition of sequential protocol composition in Maude-NPA. In Section 6.4 we describe protocol composition via protocol transformation, whereas in Section 6.5 we explain the direct implementation of protocol composition. In Section 6.6 we show the results of the experiments we have performed using our motivating protocol compositions and compare both techniques. Finally, Section 6.7 concludes this chapter.

These results have been published in [Escobar et al., 2010], and in [Santiago et al., 2014a].

6.1 Motivation

It is well known that many problems in the security of cryptographic protocols arise when the protocols are composed. This is true whether the composition is parallel, in which two different protocols are executed in an interleaved fashion, or sequential, in which one or more child protocols use information from executing a parent protocols. 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. Security of parallel composition can generally be achieved by avoiding ambiguity about which protocol a message belongs to (as in, e.g. [Guttman and Thayer, 2000; Ştefan Ciobâcă and Cortier, 2010]). The necessary conditions for security of sequential composition are harder to pin down, since they depend on the guarantees offered and needed by the particular protocols being analyzed.

The importance of understanding composition has long been acknowledged, and there are a number of logical systems that support it, as described in detail in Section 1.4. These logical systems and tools 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.

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 helps 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. For example, in

the Internet Key Exchange Protocol [Harkins and Carrel, 1998] 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. However, very little work has been done to address this problem. 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.

6.2 Examples of Sequential Protocol Compositions

In this section we provide several motivating examples of sequential composition. These examples give a flavor for the variants of sequential composition that are used in constructing cryptographic protocols. A single parent protocol instance can be composed with either many instances of a child protocol, or with only one such child instance. Likewise, parent protocol roles can determine child protocol roles, or child protocol roles can be unconstrained. In Section 6.2.1 we provide an example of a one-parent, one-child protocol composition, which appeared in [Guttman et al., 2008] and which is subject to an unexpected attack not noticed before; we also provide a corrected version that is proved secure. In Section 6.2.2 we provide an example of a one-parent, many-children protocol composition which is proved secure by our tool.

6.2.1 NSL Distance Bounding Protocol

In this example of a one-parent, one-child protocol composition, appeared in [Guttman et al., 2008], the participants first use NSL to agree on a secret nonce. We reproduce the NSL protocol below using textbook Alice-and-Bob notation where $A \rightarrow B : m$ means participant with name A sending the message m to the participant with name B:

1. $A \rightarrow B : \{N_A, A\}_{pk(B)}$

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

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 described below. This is a type of protocol, originally proposed by [Desmedt, 1988] for smart cards, which has received new interest in recent years for its possible application in wireless environments [Capkun and Hubaux, 2006]. 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 according to the following protocol:

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 self-cancellation property $X \oplus X = 0$ and the nilpotent property $X \oplus 0 = X$. 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. This is not the case for the NSL distance bounding protocol, which is subject to a form of what has come to be known as the Distance Hijacking Attack [Cremers et al., 2012] [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 Section 6.6, 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

[1] This is not meant as a denigration of [Guttman et al., 2008], whose main focus is on timing models in strand spaces, not the design of distance bounding protocols.

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

6.2.2 NSL Key Distribution Protocol

Our next example is a one parent, many children protocol 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 instances of the child protocol with the same master key.

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. This protocol is proved secure by our tool in Section 6.6.

6.3 Abstract Definition of Sequential Protocol Composition in Maude-NPA

Sequential composition of two protocols describes a situation in which one protocol (the *child*) can only execute after another protocol (the *parent*) has completed its execution, which allows the child protocol to use information generated during the execution of the parent protocol. The underlying idea of such a situation is that the end of the parent's pro-

tocol execution is *synchronized* with the beginning of the child's protocol execution. In Section 6.3.1 we first explain in detail the syntactic and semantic features necessary to express the synchronization among both protocols. Then, in Section 6.3.2 we provide an abstract definition of sequential composition of two or more protocols in Maude-NPA. Finally, in Section 6.3.3 we define a concrete execution model for the one-to-one and one-to-many protocol compositions by extending the basic Maude-NPA execution model. Throughout this chapter, we will refer to the syntax and semantics explained in this section as *abstract composition syntax and semantics*.

6.3.1 Input/Output Parameters and Roles

In this section we describe in more detail the new features we need to make explicit in each protocol to later define abstract sequential protocol compositions. Each strand in a protocol specification in the Maude-NPA is now extended with *input* 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. Note that we allow each honest or Dolev-Yao strand to be *labeled* (e.g. `NSL.init` or `NSL.resp`) to denote the "role" of that strand in the protocol, in contrast to the standard Maude-NPA syntax for strands. These strand labels play an important role in our protocol composition method.

Example 6.1 The following description of the NSL protocol contains more technical details than the informal description of NSL in Section 6.2. 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 $sk(A, M)$. The public/private encryption cancellation properties are described using the equations $pk(X, sk(X, Z)) = Z$ and $sk(X, pk(X, Z)) = Z$. The protocol $\mathcal{P}$ with two strands associated to the three protocol steps shown in Section 6.2.1 is described as follows:

$$\begin{aligned}(NSL.init) :: r :: [\{A, B\}, &+ (pk(B, n(A, r); A)),\\ &- (pk(A, n(A, r); N; B)),\\ &+ (pk(B, N)),\\ &\{A, B,\ n(A, r), N\}].\\ (NSL.resp) :: r :: [\{A, B\}, &- (pk(B, N; A)),\\ &+ (pk(A, N; n(B, r); B)),\\ &- (pk(B, n(B, r))),\\ &\{A, B,\ N, n(B, r)\}].\end{aligned}$$

■

Example 6.2 Similarly to the NSL protocol, there are several technical details missing in the previous informal description of the Distance Bounding (DB) protocol. The exclusive-or operator is $*$ and its equational properties are as described in Example 2.3 in Page 28. 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. The protocol $\mathcal{P}$ with two strands associated to the two protocol steps shown in Section 6.2.1 is described as follows:

$$\begin{aligned}(DB.init) :: r :: [\{A, B, N_A\}, &+ (n(B, r)), -(n(B, r) * N_A),\\ &\{A, B, N_A,\ n(B, r)\}].\\ (DB.resp) :: nil :: [\{A, B, N_A\}, &- (N_B), +(N_B * N_A),\\ &\{A, B, N_A,\ N_B\}].\end{aligned}$$

This protocol specification makes 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 6.3 The previous informal description of the Key Distribution (KD) protocol also lacks several technical details, which we supply here. 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. The protocol $\mathcal{P}$ with two strands associated to the KD protocol steps shown above is described as follows:

$$
\begin{aligned}
(KD.init) :: r :: [\{A, B, K\}, &+ (e(K, skey(A, r)),\\
&- (e(K, skey(A, r); N)), +(e(K, N)),\\
&\{A, B, K,\ skey(A, r), N\}].\\
(KD.resp) :: r :: [\{A, B, K\}, &- (e(K, SK)),\\
&+ (e(K, SK; n(B, r))), -(e(K, n(B, r))),\\
&\{A, B, K,\ SK, n(B, r)\}].
\end{aligned}
$$

■

In the rest of this chapter 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. Therefore, a strand is now a term of one of the following forms:

1. $[nil, \overrightarrow{M}, nil]$, i.e. a standard strand that cannot be connected to either a parent or a child strand,
2. $[\{\overrightarrow{I}\}, \overrightarrow{M}, nil]$, i.e. a *child strand* that can be connected to a parent strand,
3. $[nil, \overrightarrow{M}, \{\overrightarrow{O}\}]$, i.e. a *parent strand* that can be connected to a child strand,
4. $[\{\overrightarrow{I}\}, \overrightarrow{M}, \{\overrightarrow{O}\}]$, i.e. a strand that can be connected to both a parent and a child strand, or
5. $[\{\overrightarrow{I}\}, \{\overrightarrow{O}\}]$, i.e. a strand that can be connected to both a parent and a child strand, but without sending or receiving any message, called a *void strand*.

6.3.2 Strand and Protocol Composition

In this section we formally define sequential protocol composition in Maude-NPA. We first define the sequential composition of two strands, since this will help us to define sequential protocol composition in general. Intuitively, sequential composition of two strands describes a situation in which one strand (the *child*), can only execute after another strand (*the parent*) has completed its 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-specified way. Note that it may be possible for a single parent strand to have more than one child strand.

Definition 6.4 (Sequential Strand Composition) *Given two strands* $(a) :: \overrightarrow{r_a} :: [\{\overrightarrow{I_a}\}, \overrightarrow{M_a}, \{\overrightarrow{O_a}\}]$ *and* $(b) :: \overrightarrow{r_b} :: [\{\overrightarrow{I_b}\}, \overrightarrow{M_b}, \{\overrightarrow{O_b}\}]$ *that are properly renamed to avoid variable sharing, a sequential strand composition is a triple of the form* $(a, b, Mode)$ *where* a *and* b *denote the parent and child roles, respectively, and Mode is either 1-1 or 1-*, indicating a one-to-one or one-to-many composition. This triple satisfies the following conditions for consistency:*

1. *both* $\overrightarrow{O_a}$ *and* $\overrightarrow{I_b}$ *have the same length, i.e.* $\overrightarrow{O_a} = m_1, \ldots, m_n$ *and* $\overrightarrow{I_b} = m'_1, \ldots, m'_n$, *and*
2. *there exists at least one substitution* σ *such that* $\overrightarrow{O_a} =_{E_\mathcal{P}} \overrightarrow{I_b}\sigma$.

Example 6.5 Let us consider again our two examples of sequential protocol composition. The composition of the NSL initiator strand and the DB responder strand is specified by the triple $(NSL.init, DB.resp, 1\text{-}1)$, where both strands are as shown below:

$$
\begin{aligned}
(NSL.init) :: r :: [\{A, B\}, +(pk(B, n(A, r); A)),& \\
-(pk(A, n(A, r); N; B)), +(pk(B, N)),& \\
\{A, B,\ n(A, r)\}].& \\
(DB.resp) :: nil :: [\{A, B, N_A\}, -(N_B), +(N_B * N_A),& \\
\{A, B, N_A,\ N_B\}].&
\end{aligned}
$$

The composition of the NSL initiator strand with the KD responder strand is specified by the triple $(NSL.init, KD.resp, 1\text{-}*)$, where both strands are as shown below:

$$\begin{aligned}(NSL.init) :: r :: [\{A, B\}, &+ (pk(B, n(A, r); A)),\\ &- (pk(A, n(A, r); N; B)), +(pk(B, N)),\\ &\{A, B,\ h(n(A, r), N)\}].\\ (KD.resp) :: r :: [\{A, B, K\}, &- (e(K, SK)),\\ &+ (e(K, SK; n(B, r))), -(e(K, n(B, r))),\\ &\{A, B, K,\ SK, n(B, r)\}].\end{aligned}$$

such that the term $h(n(A, r), N)$ has the same sort as that of the input parameter K. ■

Intuitively, we can now define the sequential composition of two protocols as a set of sequential strand compositions.

Definition 6.6 (Sequential Composition of Two Protocols) *Given two protocols $\mathcal{P}_1$ and $\mathcal{P}_2$ that are properly renamed to avoid variable sharing, a sequential composition of both protocols, written $\mathcal{P}_1 ;_S \mathcal{P}_2$, is defined as a triple of the form $(\mathcal{P}_1, S, \mathcal{P}_2)$ where S denotes a set of strand compositions between a parent strand of $\mathcal{P}_1$ and a child strand of $\mathcal{P}_2$ of the form described in Definition 6.4. Note that the signature of such protocol composition is the union[2] of the signature of both protocols, i.e., $\Sigma_{\mathcal{P}_1;_S\mathcal{P}_2} = \Sigma_{\mathcal{P}_1} \cup \Sigma_{\mathcal{P}_2}$. Similarly, the set of equations specifying the algebraic properties of such protocol composition is the union of the equations of both protocols, i.e., $E_{\mathcal{P}_1;_S\mathcal{P}_2} = E_{\mathcal{P}_1} \cup E_{\mathcal{P}_2}$.*

Example 6.7 Let us consider again both the NSL and DB protocols and their composition. The composition of both protocols, which is an example of a one-to-one composition, is specified as follows, indicating that the initiator of NSL can be composed with the responder of DB and the responder of NSL with the initiator of DB:

[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 chapter.

$$NSL\ ;_S\ DB = (NSL, \{(NSL.init, DB.resp, 1{-}1), \\ (NSL.resp, DB.init, 1{-}1)\}, DB)$$

where the strands are as shown below:

$$\begin{aligned}
(NSL.init) :: r :: [\ &+(pk(B, n(A,r); A)), \\
&-(pk(A, n(A,r); N; B)), +(pk(B,N)), \{A, B, n(A,r)\}] \\
(NSL.resp) :: r :: [\ &-(pk(B, N; A)), \\
&+(pk(A, N; n(B,r); B)), -(pk(B, n(B,r))), \{A, B, N\}] \\
(DB.init) :: r :: \ &[\{A, B, N_A\}, +(n(B,r)), -(n(B,r) * N_A)] \\
(DB.resp) :: nil :: \ &[\{A, B, N_A\}, -(N_B), +(N_B * N_A)]
\end{aligned}$$

Note that we have removed irrelevant input and output parameters for clarity and simplicity. ■

Example 6.8 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, indicating that there are four possible compositions: the initiator of NSL composed with either the initiator or the responder of KD, and the responder of NSL composed with either the initiator or the responder of KD:

$$\begin{aligned}
NSL\ ;_S\ KD = (\ &NSL, (NSL.init, KD.init, 1{-}*), \\
&(NSL.init, KD.resp, 1{-}*), \\
&(NSL.resp, KD.init, 1{-}*), \\
&(NSL.resp, KD.resp, 1{-}*)\}, KD)
\end{aligned}$$

and the strands are as follows:

$$
\begin{aligned}
(NSL.init) :: r :: [\ &+(pk(B, n(A,r); A)), -(pk(A, n(A,r); N; B)),\\
&+(pk(B,N)), \{A, B, h(n(A,r), N)\}]\\
(NSL.resp) :: r :: [\ &-(pk(B, N; A)), +(pk(A, N; n(B,r); B)),\\
&-(pk(B, n(B,r))), \{A, B, h(N, n(B,r))\}]\\
(KD.init) :: r :: [&\{A, B, K\}, +(e(K, skey(A,r)),\\
&-(e(K, skey(A,r); N)), +(e(K,N))]\\
(KD.resp) :: r :: [&\{A, B, K\}, -(e(K, SK)), +(e(K, SK; n(B,r))),\\
&-(e(K, n(B,r))]
\end{aligned}
$$

such that terms *h(n(A,r),N)* and *h(N,n(B,r))* are of the same sort as variable K. Note that, again, we have removed irrelevant input and output parameters for clarity. ■

Remark 1 *As we shall see in Sections 6.4.2 and 6.5.4, composition via protocol transformation and composition via synchronization messages implement most of our abstract composition semantics. There is one important exception though in the case of one-to-many composition. In the abstract semantics there is nothing preventing a single instantiation of a parent role from having two or more children belonging to different roles, assuming both child roles are allowed by the specification. In composition via protocol transformation and composition via synchronization messages, a particular instantiation of a role can have children belonging to only one role, although that role may be one of any of the roles allowed by the specification. Thus, in the NSL-KD case, the case in which an instantiation of an NSL strand has both a KD-initiator child and a KD-responder child is never reachable via the protocol transformation or the synchronization message syntaxes and semantics. However, the case in which one instantiation of a parent NSL strand has a KD-initiator child, and another instantiation has a KD-responder child, may be reachable. This can be implemented by requiring that parents only compose with children of the sort that they have composed with before.*

In our proof of the soundness and completeness result in Sections 6.4.2 and 6.5.4, we thus make the further restriction on the abstract semantics that any instantiation of a strand only has children of a single role. We

do not make this restriction a permanent part of the definition of the abstract semantics however, since allowing greater freedom in the abstract semantics allows us to explore further options in the future.

Finally, we need to define the sequential composition of more than two protocols. Intuitively, the sequential composition of n protocols $\mathcal{P}_1, \ldots, \mathcal{P}_n$ is a set of two-protocol compositions, such that each protocol is composed with the previous protocol (except $\mathcal{P}_1$) and with the next protocol (except $\mathcal{P}_n$).

Definition 6.9 (Sequential Composition of n Protocols) *Given n protocols $\mathcal{P}_1, \ldots, \mathcal{P}_n$ that are properly renamed to avoid variable sharing, the sequential composition of all of them is denoted by:*

$$\mathcal{P}_1 \;;_{S_1} \mathcal{P}_2 \;;_{S_2} \mathcal{P}_3 \;;_{S_3} \ldots ;_{S_{n-2}} \mathcal{P}_{n-1} \;;_{S_{n-1}} \mathcal{P}_n$$

iff $\mathcal{P}_1 \;;_{S_1} \mathcal{P}_2$, $\mathcal{P}_2 \;;_{S_2} \mathcal{P}_3$, $\ldots$, $\mathcal{P}_{n-1} \;;_{S_{n-1}} \mathcal{P}_n$ are sequential protocol compositions as explained in Definition 6.6.

6.3.3 Abstract Operational Semantics

As explained in Section 3.4, 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 to the rewrite theory $(\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, R_{B\mathcal{P}})$ that characterizes the behavior of protocol $\mathcal{P}$ modulo the equations $E_{\mathcal{P}}$ for backwards execution in order to deal with protocol composition.

In the one-to-one composition, we add the state transition rules of Figure 6.1 to the rewrite theory $(\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, R_{B\mathcal{P}})$. Note that these transition rules are written in a forwards way but will be executed backwards, as the Maude-NPA's basic transition rules $R_{B\mathcal{P}}$. Rule (6.1) composes a parent and a child strand already present in the current state. Rule (6.2) adds, in a backwards execution, a parent strand to the current state and composes it with an existing child strand. For example, given the composition of the NSL initiator's strand with the DB responder's strand $(NSL.init, DB.resp, 1{-}1)$ where *NSL.init* and *DB.resp* were defined in Example 6.7 in Page 147, we add the following transition rule for Rule

For each one-to-one strand composition $(a, b, 1{-}1)$ with
strand $(a)[\overrightarrow{M_a}, \{\overrightarrow{O_a}\}]$ for protocol $\mathcal{P}_1$,
strand $(b)[\{\overrightarrow{I_b}\}, \overrightarrow{M_b}]$ for protocol $\mathcal{P}_2$,
and for each substitution σ s.t. $\overrightarrow{I_b}\sigma =_{E_\mathcal{P}} \overrightarrow{O_a}$
we add the following rules:

$$\begin{aligned} & SS \,\&\, (a)\ [\overrightarrow{M_a} \mid \{\overrightarrow{O_a}\}] \ \&\ (b)\ [nil \mid \{\overrightarrow{I_b}\sigma\}, \overrightarrow{M_b}\sigma] \,\&\, IK \\ \rightarrow\ & SS \,\&\, (a)\ [\overrightarrow{M_a}, \{\overrightarrow{O_a}\} \mid nil] \ \&\ (b)\ [\{\overrightarrow{I_b}\sigma\} \mid \overrightarrow{M_b}\sigma] \,\&\, IK \end{aligned} \tag{6.1}$$

$$\begin{aligned} & SS \,\&\, (a)\ [\overrightarrow{M_a} \mid \{\overrightarrow{O_a}\}] \ \&\ (b)\ [nil \mid \{\overrightarrow{I_b}\sigma\}, \overrightarrow{M_b}\sigma] \,\&\, IK \\ \rightarrow\ & SS \,\&\, (b)\ [\{\overrightarrow{I_b}\sigma\} \mid \overrightarrow{M_b}\sigma] \,\&\, IK \end{aligned} \tag{6.2}$$

Figure 6.1: Forward semantics for one-to-one composition

(6.1) where both the parent and the child strands are present and thus synchronized.

$$\begin{aligned} & (NSL.init) :: r :: \\ & [\, +(pk(B, n(A,r); A)), -(pk(A, n(A,r); N; B)), +(pk(B, N)) \mid \\ & \ \{A, B, n(A,r)\}\,] \,\& \\ & (DB.resp) :: nil :: \\ & [\, nil \mid \{A, B, n(A,r)\}, -(NB), +(NB * n(A,r))\,] \ \&\ SS \ \&\ IK \\ & \longrightarrow \\ & (NSL.init) :: r :: \\ & [\, +(pk(B, n(A,r); A)), -(pk(A, n(A,r); N; B)), +(pk(B, N)), \\ & \ \{A, B, n(A,r)\} \mid nil\,] \,\& \\ & (DB.resp) :: nil :: \\ & [\, \{A, B, n(A,r)\} \mid\ -(NB), +(NB * n(A,r))\,] \ \&\ SS \ \&\ IK \end{aligned}$$

One-to-many composition uses the rules in Figure 6.1 for the first child, plus an additional rule for subsequent children, described in Figure 6.2. Rule (6.3) composes a parent strand and a child strand but the

For each one-to-many strand composition $(a, b, 1{-}*)$ with
strand $(a)[\overrightarrow{M_a}, \{\overrightarrow{O_a}\}]$ for protocol $\mathcal{P}_1$,
strand $(b)[\{\overrightarrow{I_b}\}, \overrightarrow{M_b}]$ for protocol $\mathcal{P}_2$,
and for each substitution σ s.t. $\overrightarrow{I_b}\sigma =_{E_\mathcal{P}} \overrightarrow{O_a}$
we add one Rule (6.1), one Rule (6.2), and the following rule :

$$\begin{aligned} & SS \,\&\, (a)\ [\overrightarrow{M_a} \mid \{\overrightarrow{O_a}\}] \ \&\ (b)\ [nil \mid \{\overrightarrow{I_b}\sigma\}, \overrightarrow{M_b}\sigma] \,\&\, IK \\ \rightarrow\ & SS \,\&\, (a)\ [\overrightarrow{M_a} \mid \{\overrightarrow{O_a}\}] \ \&\ (b)\ [\{\overrightarrow{I_b}\sigma\} \mid \overrightarrow{M_b}\sigma] \,\&\, IK \end{aligned} \tag{6.3}$$

Figure 6.2: Forward semantics for one-to-many composition

bar in the parent strand is not moved, in order to allow further backwards child compositions. For example, given the composition of the NSL responder's strand with the KD initiator's strand ($NSL.resp, KD.init$,1-*) where $NSL.resp$ and $KD.init$ are as defined in Example 6.8 in Page 148 we add the following transition rule for Rule (6.3):

$$\begin{aligned} & (NSL.resp) :: r :: \\ & [\, -(pk(B, NA; A)), +(pk(A, NA; n(B, r); B)), -(pk(B, n(B, r))) \mid \\ & \ \{A, B, h(NA, n(B, r))\}\,] \ \& \\ & (KD.init) :: r' :: \\ & [\, nil \mid \{A, B, h(NA, n(B, r))\}, +(e(h(NA, n(B, r)), skey(A, r')), \\ & \ - (e(h(NA, n(B, r)), skey(A, r'); N)), +(e(h(NA, n(B, r)), N))\,] \\ & \&\ SS \ \&\ IK \\ & \longrightarrow \\ & (NSL.resp) :: r :: \\ & [\, -(pk(B, NA; A)), +(pk(A, NA; n(B, r); B)), -(pk(B, n(B, r))) \mid \\ & \ \{A, B, h(NA, n(B, r))\}, nil\,] \ \& \\ & (KD.init) :: r' :: \\ & [\, \{A, B, h(NA, n(B, r))\} \mid \ + (e(h(NA, n(B, r)), skey(A, r')), \\ & \ - (e(h(NA, n(B, r)), skey(A, r'); N)), +(e(h(NA, n(B, r)), N))\,] \\ & \&\ SS \ \&\ IK \end{aligned}$$

Thus, for a protocol composition $\mathcal{P}_1;_S \mathcal{P}_2$, the rewrite rules governing protocol execution are $R_{\mathcal{P}_1;_S\mathcal{P}_2} = \{(3.1), (3.2), (3.3)\} \cup \{(3.4)\} \cup \{(6.1), (6.2)\} \cup \{(6.3)\}$.

6.4 Protocol Composition via Protocol Transformation

In Section 6.3 we have provided an abstract syntax and a semantics for protocol composition in Maude-NPA. However, implementing this means a significant modification of Maude-NPA to support the new composition data type. In this section we show that sequential protocol composition can be implemented using communication between strands via messages sent over the Dolev-Yao channel. This approach, which will be referred to as *composition by protocol transformation*, does not require to modify the Maude-NPA since composition is performed by a simple protocol transformation, as explained in Section 6.4.1. In Section 6.4.2 we prove soundness and completeness of the composition by protocol transformation with respect to the abstract operational semantics of Section 6.3.3. We illustrate our results on our two running examples.

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

Given two protocols $\mathcal{P}_1$ and $\mathcal{P}_2$, its sequential composition performed via the protocol transformation given in Figure 6.3, written $\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)$, is a single, composed protocol specification where:

1. Sorts, symbols, and equational properties of both protocols are put together into a single specification. As explained in Footnote 6.6 in Page 147, we allow shared items but require the user to solve any possible conflict. A new sort Param is defined to denote input and output parameters. Strands of both protocols are transformed

$$\Phi(\mathcal{P}_1 \;;_S \mathcal{P}_2) = \begin{cases} \text{add strand } [\overrightarrow{M_a}, -(b(r)), +(a(r) \;.\; \dot{I}_b))], \text{ and} \\ \quad \text{strand } [+(b(r)), -(a(r) \;.\; \dot{I}_b), \overrightarrow{M_b}] \\ \quad \text{whenever} \\ \quad\quad (a, b, 1{-}1) \text{ in } \mathcal{P}_1 \;;_S \mathcal{P}_2, \\ \quad\quad \text{strand } (a)[\{\overrightarrow{I_a}\}, \overrightarrow{M_a}, \{\overrightarrow{O_a}\}] \text{ for protocol } \mathcal{P}_1, \\ \quad\quad \text{strand } (b)[\{\overrightarrow{I_b}\}, \overrightarrow{M_b}, \{\overrightarrow{O_b}\}] \text{ for protocol } \mathcal{P}_2, \\ \quad\quad \text{and } r \text{ is a fresh variable} \\ \text{add strand } [\overrightarrow{M_a}, +(a(r) \;.\; \dot{I}_b)], \text{ and} \\ \quad \text{strand } [-(a(r) \;.\; \dot{I}_b), \overrightarrow{M_b}] \\ \quad \text{whenever} \\ \quad\quad (a, b, 1{-}*) \text{ in } \mathcal{P}_1 \;;_S \mathcal{P}_2, \\ \quad\quad \text{strand } (a)[\{\overrightarrow{I_a}\}, \overrightarrow{M_a}, \{\overrightarrow{O_a}\}] \text{ for protocol } \mathcal{P}_1, \\ \quad\quad \text{strand } (b)[\{\overrightarrow{I_b}\}, \overrightarrow{M_b}, \{\overrightarrow{O_b}\}] \text{ for protocol } \mathcal{P}_2, \\ \quad\quad \text{and } r \text{ is a fresh variable} \end{cases}$$

Figure 6.3: Protocol Transformation

and added to this single specification as described in Figure 6.3. Note that this protocol transformation removes the strand labels since this information is now included in the input and output parameters.

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 identifier (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.

The rewrite theory defining the protocol execution of a protocol composition $\Phi(\mathcal{P}_1 \;;_S \mathcal{P}_2)$ is denoted by $\mathcal{R}_{\Phi(\mathcal{P}_1 \;;_S \mathcal{P}_2)}$.

As explained in Remark 1 in Page 149 in composition via protocol transformation we assume that in a one-to-many composition a particular instantiation of a parent role can have children belonging to only one role, although that role may be one of any of the roles allowed by the specification. Therefore, note that the protocol composition $(\mathcal{P}_1 \;;_S \mathcal{P}_2)$ may contain two or more strands specifying the same role which basically differ in the input message that corresponds to the composition identifier message sent by the parent strand.

Let us now illustrate this protocol transformation with our examples of protocol compositions.

Example 6.10 The transformed strands of the one-to-one protocol composition $NSL;_S DB$ of Example 6.7 are as shown below:

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

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

The transformed strands of the one-to-many composition $NSL;_S\, KD$ of Example 6.8 are as shown below:

```
--- NSL protocol
:: r , r# ::
[ nil | +(NSL-init),
        +(pk(B,n(A,r) ; A)) ,
        -(pk(A, n(A,r) ; NB ; B )),
        +(pk(B, NB)),
        +({NSL-init(r#) . A . B . h(n(A,r) , NB) }), nil ] &
:: r , r# ::
[ nil | +(NSL-resp),
        -(pk(B,NA ; A)),
        +(pk(A, NA ; n(B,r) ; B)),
        -(pk(B,n(B,r))),
        +({NSL-resp(r#) . A . B . h(NA , n(B,r))}), nil ] &

---- KD protocol
:: r' ::
[ nil | +(KD-init),
        -({NSL-init(r#) . A . B . MKe }),
        +(e(MKe, skey(A, n(A,r')))) ,
        -(e(MKe, skey(A, n(A,r')) ; N)),
        +(e(MKe, N)), nil] &
:: r' ::
```

```
[ nil | +(KD-resp),
        -({NSL-resp(r#) . A . B . MKe }),
        -(e(MKe, skey(A,NA'))),
        +(e(MKe, skey(A,NA') ; n(B,r'))),
        -(e(MKe, n(B,r'))), nil ]     &
 ---
:: r' ::
[ nil | +(KD-init),
        -({NSL-resp(r#) . A . B . MKe }),
        +(e(MKe, skey(B, n(B,r')))),
        -(e(MKe, skey(B, n(B,r')) ; N)),
        +(e(MKe, N)), nil ] &
:: r' ::
[ nil | +(KD-resp),
        -({NSL-init(r#) . A . B . MKe }),
        -(e(MKe, skey(B,NB') )),
        +(e(MKe, skey(B,NB') ; n(A,r'))),
        -(e(MKe, n(A,r'))),  nil ]
```

■

6.4.2 Soundness and Completeness of the Protocol Transformation

In this section we prove soundness and completeness of the composition by protocol transformation with respect to the abstract operational semantics of Section 6.3.3 subject to the assumption described in Remark 1, that any given instantiation of a parent role can have children of only one type of role, although it may have multiple choices for the one role. First, in Section 6.4.2.1 we relate protocol states using the protocol composition rewrite rules of Section 6.3.3 and protocol states in the transformed protocol composition. Then, in Section 6.4.2.2 we prove soundness and completeness of one backwards narrowing step using the rewrite theory associated to the transformed protocol (i.e., $\mathcal{R}_{\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)}$) w.r.t. one backwards narrowing step using the rewrite theory associated to the protocol composition of Section 6.3.3 (i.e., $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$). Finally, Section 6.4.2.3 extends the soundness and completeness results of Section 6.4.2.2 for one narrowing step to a complete backwards reachability analysis.

In the remainder of this chapter, when we can avoid confusion, a state *St* is called *valid according to a rewrite theory* $\mathcal{R}$ if it is a valid term of sort State with respect to the order-sorted signature of $\mathcal{R}$. We call a state *initial* if there are no backwards narrowing steps from it.

6.4.2.1 Relating States from Protocol Composition and Protocol Transformation

First, we must relate protocol states using the protocol composition rewrite rules of Section 6.3.3 and protocol states in the transformed protocol composition.

Definition 6.11 (Functions $trans_{\Phi}$ and inv_{Φ}) *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two protocols and $\mathcal{P}_1 ;_S \mathcal{P}_2$ their composition. Let $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$ be the rewrite theory associated in Section 6.3.3 to the protocol composition $\mathcal{P}_1 ;_S \mathcal{P}_2$ and $\mathcal{R}_{\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ be the rewrite theory associated in Section 6.4.1 to the transformed protocol. We define the following:*

1. *the relation $trans_{\Phi}$ between states valid according to the rewrite theory $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$ and states valid according to the rewrite theory $\mathcal{R}_{\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ as specified in Figure 6.4.*

2. *the function inv_{Φ} from states valid according to the rewrite theory $\mathcal{R}_{\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ into states valid according to the rewrite theory $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$ as specified in Figure 6.5.*

The following auxiliary results become crucial and ensure that there is an appropriate connection between states of both rewrite theories.

Lemma 6.12 *The function inv_{Φ} is total and relation $trans_{\Phi}$ is the inverse of inv_{Φ}. Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two protocols and $\mathcal{P}_1 ;_S \mathcal{P}_2$ their composition. Let $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$ be the rewrite theory associated in Section 6.3.3 to the protocol composition $\mathcal{P}_1 ;_S \mathcal{P}_2$ and $\mathcal{R}_{\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ be the rewrite theory associated in Section 6.4.1 to the transformed protocol.*

The function inv_{Φ} defined in Definition 6.11 defines a total function from terms of sort State *in $\mathcal{R}_{\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ back to terms of sort* State *in $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$. The function $trans_{\Phi}$ defined in Definition 6.11 is the inverse of inv_{Φ}.*

$$
trans_{\Phi}(St) = \begin{cases}
[+(b(r)), -(a(r) \,.\, \dot{I}_b), \overrightarrow{b_1} \mid \overrightarrow{b_2}] \;\&\; St' & \text{if } (b)[\{\overrightarrow{I_b}\}, \overrightarrow{b_1} \mid \overrightarrow{b_2}] \in St, \\
 & \quad (a, b, 1{-}1) \in \mathcal{P}_1 ;_S \mathcal{P}_2, \\
 & \quad trans_{\Phi}(St - (b)) = St' \\
[+(b(r)) \mid -(a(r) \,.\, \dot{I}_b), \overrightarrow{M_b}] \;\&\; St' & \text{if } (b)[\{\overrightarrow{I_b}\} \mid \overrightarrow{M_b}] \in St, \\
 & \quad (a, b, 1{-}1) \in \mathcal{P}_1 ;_S \mathcal{P}_2, \\
 & \quad trans_{\Phi}(St - (b)) = St' \\
[nil \mid +(b(r)), -(a(r) \,.\, \dot{I}_b), \overrightarrow{M_b}] \;\& & \\
[\overrightarrow{a_1} \mid \overrightarrow{a_2}, -(b(r)), +(a(r) \,.\, \dot{I}_b)] \;\&\; St' & \text{if } (b)[nil \mid \{\overrightarrow{I_b}\}, \overrightarrow{M_b}] \in St, \\
 & \quad (a)[\overrightarrow{a_1} \mid \overrightarrow{a_2}, \{\overrightarrow{O_a}\}] \in St, \\
 & \quad (a, b, 1{-}1) \in \mathcal{P}_1 ;_S \mathcal{P}_2, \\
 & \quad trans_{\Phi}(St - (a) - (b)) = St' \\
[-(a(r) \,.\, \dot{I}_b), \overrightarrow{b_1} \mid \overrightarrow{b_2}] \;\&\; St' & \text{if } (b)[\{\overrightarrow{I_b}\}, \overrightarrow{b_1} \mid \overrightarrow{b_2}] \in St, \\
 & \quad (a, b, 1{-}*) \in \mathcal{P}_1 ;_S \mathcal{P}_2, \\
 & \quad trans_{\Phi}(St - (b)) = St' \\
[nil \mid -(a(r) \,.\, \dot{I}_b), \overrightarrow{M_b}] \;\&\; St' & \text{if } (b)[\{\overrightarrow{I_b}\} \mid \overrightarrow{M_b}] \in St, \\
 & \quad (a, b, 1{-}*) \in \mathcal{P}_1 ;_S \mathcal{P}_2, \\
 & \quad trans_{\Phi}(St - (b)) = St', \\
 & \quad [\overrightarrow{M_a} \mid \; + (a(r) \,.\, \dot{I}_b)] \notin St' \\
[nil \mid -(a(r) \,.\, \dot{I}_b), \overrightarrow{M_b}] \;\& & \\
[\overrightarrow{a_1} \mid \overrightarrow{a_2}, +(a(r) \,.\, \dot{I}_b)] \;\&\; St' & \text{if } (b)[nil \mid \{\overrightarrow{I_b}\}, \overrightarrow{M_b}] \in St, \\
 & \quad (a)[\overrightarrow{a_1} \mid \overrightarrow{a_2}, \{\overrightarrow{O_a}\}] \in St, \\
 & \quad (a, b, 1{-}*) \in \mathcal{P}_1 ;_S \mathcal{P}_2, \\
 & \quad trans_{\Phi}(St - (a) - (b)) = St', \\
St & \text{otherwise}
\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'

Figure 6.4: Relation $trans_{\Phi}$ between states valid according to the rewrite theory $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$ and states valid according to the rewrite theory $\mathcal{R}_{\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)}$

$$
inv_{\Phi}(St) = \begin{cases}
(b)[\{\overrightarrow{I_b}\}, \overrightarrow{b_1} \mid \overrightarrow{b_2}] \;\&\; St' & \text{if } S_b = [+(b(r)), -(a(r) \,.\, \dot{I}_b), \overrightarrow{b_1} \mid \overrightarrow{b_2}] \in St, \\
& \quad (a, b, 1{-}1) \in \mathcal{P}_1 ;_S \mathcal{P}_2, \\
& \quad inv_{\Phi}(St - S_b) = St' \\
(b)[\{\overrightarrow{I_b}\} \mid \overrightarrow{M_b}] \;\&\; St' & \text{if } S_b = [+(b(r)) \mid -(a(r) \,.\, \dot{I}_b), \overrightarrow{M_b}] \in St, \\
& \quad (a, b, 1{-}1) \in \mathcal{P}_1 ;_S \mathcal{P}_2, \\
& \quad inv_{\Phi}(St - S_b) = St' \\
(b)[nil \mid \{\overrightarrow{I_b}\}, \overrightarrow{M_b}]\& & \\
(a)[\overrightarrow{a_1} \mid \overrightarrow{a_2}, \{\overrightarrow{O_a}\}] \;\&\; St' & \text{if } S_b = [nil \mid +(b(r)), -(a(r) \,.\, \dot{I}_b), \overrightarrow{M_b}] \in St \\
& \quad S_a = [\overrightarrow{a_1} \mid \overrightarrow{a_2}, -(b(r)), +(a(r) \,.\, \dot{I}_b)] \in St \\
& \quad (a, b, 1{-}1) \in \mathcal{P}_1 ;_S \mathcal{P}_2, \\
& \quad inv_{\Phi}(St - S_a - S_b) = St' \\
(b)[\{\overrightarrow{I_b}\}, \overrightarrow{b_1} \mid \overrightarrow{b_2}] \;\&\; St' & \text{if } S_b = [-(a(r) \,.\, \dot{I}_b), \overrightarrow{b_1} \mid \overrightarrow{b_2}] \in St \\
& \quad (a, b, 1{-}*) \in \mathcal{P}_1 ;_S \mathcal{P}_2, \\
& \quad inv_{\Phi}(St - S_b) = St' \\
(b)[\{\overrightarrow{I_b}\} \mid \overrightarrow{M_b}] \;\&\; St' & \text{if } S_b = [nil \mid -(a(r) \,.\, \dot{I}_b), \overrightarrow{M_b}] \in St, \\
& \quad [\overrightarrow{M_a} \mid \; + (a(r) \,.\, \dot{I}_b)] \notin St' \\
& \quad (a, b, 1{-}*) \in \mathcal{P}_1 ;_S \mathcal{P}_2, \\
& \quad inv_{\Phi}(St - S_b) = St', \\
(b)[nil \mid \{\overrightarrow{I_b}\}, \overrightarrow{M_b}]\& & \\
(a)[\overrightarrow{a_1} \mid \overrightarrow{a_2}, \{\overrightarrow{O_a}\}] \;\&\; St' & \text{if } S_b = [nil \mid -(a(r) \,.\, \dot{I}_b), \overrightarrow{M_b}] \in St, \\
& \quad S_a = [\overrightarrow{a_1} \mid \overrightarrow{a_2}, +(a(r) \,.\, \dot{I}_b)] \in St, \\
& \quad (a, b, 1{-}*) \in \mathcal{P}_1 ;_S \mathcal{P}_2, \\
& \quad inv_{\Phi}(St - S_a - S_b) = St' \\
St & \text{otherwise}
\end{cases}
$$

plus 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 6.5: Function inv_{Φ} mapping from states valid according to the rewrite theory $\mathcal{R}_{\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ onto states valid according to the rewrite theory $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$

Proof. By structural induction on the definitions of Figure 6.4 and 6.5. □

Note that inv_{Φ} is not injective, since two states that differ only by the fact that some strands have different child roles defined in the output parameters (respectively, different parent roles defined in the input parameters), will be mapped to the same state by inv_{Φ}. Thus its inverse $trans_{\Phi}$ is not a function.

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 $(a, b, 1{-}1)$, which is easily extensible to the whole protocol composition $\mathcal{P}_1 ;_S \mathcal{P}_2$.

Lemma 6.13 (Unique One-to-one Composition) *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two protocols and $\mathcal{P}_1 ;_S \mathcal{P}_2$ be a one-to-one composition. Let $\mathcal{R}_{\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ be the rewrite theory associated in Section 6.4.1 to the transformed protocol and $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$ be the rewrite theory associated in Section 6.3.3 to the protocol composition $\mathcal{P}_1 ;_S \mathcal{P}_2$. If St' is a protocol state valid according to the rewrite theory $\mathcal{R}_{\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)}$, 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 the same fresh variable ensures that there are no two children generating the same `r#`. □

6.4.2.2 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., $\mathcal{R}_{\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)}$) w.r.t. backwards narrowing using the rewrite theory associated to protocol composition (i.e., $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$). We write R^{-1} to denote the reverse form of all the rules in R.

Theorem 6.14 (One-step Soundness) *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two protocols and $\mathcal{P}_1 ;_S \mathcal{P}_2$ their composition. Let $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$ be the rewrite theory associated in Section 6.3.3 to the protocol composition $\mathcal{P}_1 ;_S \mathcal{P}_2$ subject to the restrictions of Remark 1 in Page 149, and $\mathcal{R}_{\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ be the rewrite theory associated in Section 6.4.1 to the transformed protocol. Let St'_1 and St'_2 be two protocol states valid according to the rewrite theory $\mathcal{R}_{\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)}$. If $St'_1 \leadsto_{\sigma, \mathcal{R}^{-1}_{\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)}, E_{\mathcal{P}_1 ;_S \mathcal{P}_2}} St'_2$, then either*

1. *there is a protocol state St_1 valid according to the rewrite theory $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$ and two substitutions ρ and ρ' such that $inv_\Phi(St'_1) = St_1$, $inv_\Phi(St'_2) = St_1\rho$, and $\sigma =_{E_{\mathcal{P}_1 ;_S \mathcal{P}_2}} \rho \circ \rho'$. or*

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

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 $\mathcal{R}^{-1}_{\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ corresponds to one rewrite step in $\mathcal{R}^{-1}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$. This case corresponds to two situations: (i) the standard uses of the rule-base semantics (i.e. Rules (3.1),(3.2), (3.3), and (3.4)) 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 a rule of the form Rule (3.4) in $\mathcal{R}_{\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ is applied in order to produce St'_2 by adding a strand $[\overrightarrow{M_a} \mid +(a(r).\dot{I}_b)]$ to St'_1 by unification with a term $(a(r) \ . \ \dot{I}_b) \in \mathcal{I}$ in the intruder knowledge of St'_1, this corresponds to the application of Rule (6.2) in $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$, in which the state St_2 is created by adding the strand $[\overrightarrow{M_a} \mid \{\overrightarrow{O_a}\}]$ 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 Rule (3.3) in $\mathcal{R}_{\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ to unify the term $+(a(r') \ . \ \dot{I}_b)$ of a strand $[\overrightarrow{M_a}, +(a(r')$

. $\dot{I}_b$) | nil] in St'_1 with a term $(a(r)$. $\dot{I}_b) \in \mathcal{I}$ in the intruder knowledge of St'_1, this corresponds to the creation of St_2 from St_1 via the application of Rule (6.1) in $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$, in which the output parameters $\{\overrightarrow{O_a}\}$ of the strand associated to protocol $\mathcal{P}_1$ are synchronized with the input parameters $\{\overrightarrow{I_b}\}$ of the strand associated to protocol $\mathcal{P}_2$. 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 (3.3) in $\mathcal{R}_{\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ to unify the term $+(b(r))$ of a strand $[+(b(r)) \mid -(a(r)$. $\dot{I}_b), \overrightarrow{M_b}]$ in St'_1 with a term $b(r) \in \mathcal{I}$ in the intruder knowledge of St'_1, this corresponds to the application of Rule (6.1) in $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$ to create St_2 from St_1, in which the output parameters $\{\overrightarrow{O_a}\}$ of the strand associated to protocol $\mathcal{P}_1$ are synchronized with the input parameters $\{\overrightarrow{I_b}\}$ of the strand associated to protocol $\mathcal{P}_2$. In this case, the substitution ρ above may be different from the identity, since the concrete unifier computed by Rule (6.1) was computed before, when the term $(a(r)$. $\dot{I}_b) \in \mathcal{I}$ was unified with a concrete parent strand.

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

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

2. In a one-to-one composition, the parent transition from strand $[\overrightarrow{M_a}, -(b(r)), +(a(r)$. $\dot{I}_b) \mid nil]$ in St'_1 to strand $[\overrightarrow{M_a}, -(b(r)) \mid +(a(r)$. $\dot{I}_b)]$ in St'_2. Here, state St'_1 has a state St_1 given by inv_{Φ}, St'_2 has a state St_2 given by inv_{Φ} 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 $[\overrightarrow{M_a}, -(b(r)) \mid +(a(r) \cdot \dot{I}_b)]$ to St'_1, yields St'_2. Here, state St'_1 has a state St_1 given by inv_{Φ}, St'_2 has a state St_2 given by inv_{Φ} 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 $[\overrightarrow{M_a}, -(b(r)) \mid +(a(r) \cdot \dot{I}_b)]$ in St'_1 to strand $[\overrightarrow{M_a} \mid -(b(r)), +(a(r) \cdot \dot{I}_b)]$ in St'_2. Here, states St'_1 and St'_2 have the same state St_1 given by inv_{Φ}.

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

This concludes the proof. □

Now, we consider completeness of one backwards narrowing step using the rewrite theory associated to protocol composition (i.e., $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$) w.r.t. backwards narrowing using the rewrite theory associated to the transformed protocol (i.e., $\mathcal{R}_{\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)}$). We write $St_1 \rightarrow_R^{1,2,4} St_2$ 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 6.15 (One-step Completeness) *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two protocols and $\mathcal{P}_1 ;_S \mathcal{P}_2$ their composition. Let $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$ be the rewrite theory associated in Section 6.3.3 to the protocol composition $\mathcal{P}_1 ;_S \mathcal{P}_2$ subject to the restrictions of Remark 1 in Page 149, and $\mathcal{R}_{\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ be the rewrite theory associated in Section 6.4.1 to the transformed protocol. Let St_1 and St_2 be two protocol states valid according to the rewrite theory $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$. If $St_1 \rightsquigarrow_{\sigma, \mathcal{R}^{-1}_{\mathcal{P}_1 ;_S \mathcal{P}_2}, E_{\mathcal{P}_1 ;_S \mathcal{P}_2}} St_2$, then there are two protocol states St'_1 and St'_2 valid according to the rewrite theory $\mathcal{R}_{\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ such that $trans_{\Phi}(St_1) = St'_1$, $trans_{\Phi}(St_2) = St'_2$, and $St'_1 \rightsquigarrow^{1,2,4}_{\sigma, \mathcal{R}^{-1}_{\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)}, E_{\mathcal{P}_1 ;_S \mathcal{P}_2}} St'_2$.*

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 6.14, we present the cases only in broad outline.

When the Rules (3.1),(3.2), (3.3), and (3.4) are applied to regular incoming and outcoming messages in the strands, we have just one narrowing step from state St'_1.

When we have a one-to-many composition and Rules (6.1), (6.2), or (6.3) are applied, we have two narrowing steps from state St'_1 as follows:

1. Rule (6.1) corresponds to an application of Rule (3.1) (accepting the input parameters of the child strand) followed by an application of Rule (3.3) (synchronizing the input of the child strand with the output parameters of the parent strand);

2. Rule (6.2) corresponds to an application of Rule (3.1) followed by an application of Rule (3.4) (introducing a new strand); and

3. Rule (6.3) corresponds to an application of Rule (3.3) (synchronizing the output parameters of the parent strand with the already accepted input parameters of the child strand). Given that our requirement that the restrictions of Remark 1 hold, Rule (6.3) can be applied if and only if the corresponding rule of the form (3.3) can be applied.

When we have a one-to-one composition and Rules (6.1) or (6.2) are applied, we have four narrowing steps from the state St'_1 as follows:

1. Rule (6.1) corresponds to an application of Rule (3.1) (accepting the input parameters of the child strand) followed by an application of Rule (3.3) (synchronizing the input of the child strand with the output parameters of the parent strand), and two further applications of Rules (3.1) and (3.3) for the $b(r)$ message; and

2. Rule (6.2) corresponds to an application of Rule (3.1) followed by an application of Rule (3.4) (introducing a new strand), and two further applications of Rules (3.1) and (3.3) for the $b(r)$ message.

This concludes the proof. □

6.4.2.3 Soundness and Completeness for Reachability Analysis

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

Theorem 6.16 (Reachability Soundness) *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two protocols and $\mathcal{P}_1 ;_S \mathcal{P}_2$ their composition. Let $\mathcal{R}_{\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ be the rewrite theory associated in Section 6.4.1 to the transformed protocol and $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$ be the rewrite theory associated in Section 6.3.3 to the protocol composition $\mathcal{P}_1 ;_S \mathcal{P}_2$, subject to the restrictions of Remark 1 in Page 149. If St'_1 and St'_2 are two protocol states valid according to the rewrite theory $\mathcal{R}_{\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ such that St'_2 is an initial state, and $St'_1 \leadsto^*_{\sigma, \mathcal{R}^{-1}_{\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)}, E_{\mathcal{P}_1 ;_S \mathcal{P}_2}} St'_2$, then there are two protocol states St_1 and St_2 valid according to the rewrite theory $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$ and two substitutions ρ and ρ' such that St_2 is an initial state, $inv_\Phi(St'_1) = St_1\rho$, $inv_\Phi(St'_2) = St_2$, $St_1 \leadsto^*_{\rho', \mathcal{R}^{-1}_{\mathcal{P}_1 ;_S \mathcal{P}_2}, E_{\mathcal{P}_1 ;_S \mathcal{P}_2}} St_2$, and $\rho' =_{E_{\mathcal{P}_1 ;_S \mathcal{P}_2}} \rho \circ \sigma$.*

Proof. By successive application of Theorem 6.14. Let us consider

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

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

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

Then, by Theorem 6.15, either

1. there is a protocol state St_1 valid according to the rewrite theory $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$ and a substitution ρ such that $inv_\Phi(St'_1) = St_1$, $inv_\Phi(St') = St_1\rho$ and σ_1 and ρ are $E_{\mathcal{P}_1 ;_S \mathcal{P}_2}$-compatible; or
2. there are two protocol states St_1 and St valid according to the rewrite theory $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$ and two substitutions ρ_1 and ρ'_1 such that $inv_\Phi(St'_1) = St_1\rho_1$, $inv_\Phi(St') = St$, $St_1 \leadsto_{\rho'_1, \mathcal{R}^{-1}_{\mathcal{P}_1 ;_S \mathcal{P}_2}, E_{\mathcal{P}_1 ;_S \mathcal{P}_2}} St$ 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 valid according to the rewrite theory $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$ and two substitutions $\widehat{\rho}$ and $\widehat{\rho}'$ such that St_2 is an initial state, $inv_\Phi(St') = \widehat{St}\widehat{\rho}$, $inv_\Phi(St'_2) = St_2$, $\widehat{St} \leadsto^*_{\widehat{\rho}', \mathcal{R}^{-1}_{\mathcal{P}_1 ;_S \mathcal{P}_2}, E_{\mathcal{P}_1 ;_S \mathcal{P}_2}} St_2$, and $\widehat{\rho}' = \widehat{\rho} \circ \sigma'$. Since $inv_\Phi(St') = St_1\rho$ and $inv_\Phi(St') = \widehat{St}\widehat{\rho}$, we can conclude by definition of the relation inv_Φ that there is a substitution σ'' s.t

$St_1 \rightsquigarrow^*_{\sigma'',\mathcal{R}^{-1}_{\mathcal{P}_1 ;_S \mathcal{P}_2},E_{\mathcal{P}_1 ;_S \mathcal{P}_2}} St_2$ 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 two protocol states $\widehat{St}$ and St_2 valid according to the rewrite theory $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$ and two substitutions $\widehat{\rho}$ and $\widehat{\rho}'$ such that St_2 is an initial state, $inv_\Phi(St') = \widehat{St}\widehat{\rho}$, $inv_\Phi(St'_2) = St_2$, $\widehat{St} \rightsquigarrow^*_{\widehat{\rho}',\mathcal{R}^{-1}_{\mathcal{P}_1 ;_S \mathcal{P}_2},E_{\mathcal{P}_1 ;_S \mathcal{P}_2}} St_2$, and $\widehat{\rho}' = \widehat{\rho} \circ \sigma'$. Here the conclusion follows because we can build the sequence

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

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

Theorem 6.17 (Reachability Completeness) *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two protocols and $\mathcal{P}_1 ;_S \mathcal{P}_2$ their composition. Let $\mathcal{R}_{\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ be the rewrite theory associated in Section 6.4.1 to the transformed protocol, and $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$ be the rewrite theory associated in Section 6.3.3 to the protocol composition $\mathcal{P}_1 ;_S \mathcal{P}_2$, subject to the restrictions of Remark 1 in Page 149. If St_1 and St_2 are two protocol states valid according to the rewrite theory $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$ such that St_2 is an initial state, and $St_1 \rightsquigarrow^*_{\sigma,\mathcal{R}^{-1}_{\mathcal{P}_1 ;_S \mathcal{P}_2},E_{\mathcal{P}_1 ;_S \mathcal{P}_2}} St_2$, then there are two protocol states St'_1 and St'_2 valid according to the rewrite theory $\mathcal{R}_{\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ such that St'_2 is an initial state, $trans_\Phi(St_1) = St'_1$, $trans_\Phi(St_2) = St'_2$, and $St'_1 \rightsquigarrow^*_{\sigma,\mathcal{R}^{-1}_{\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)},E_{\mathcal{P}_1 ;_S \mathcal{P}_2}} St'_2$.*

Proof. By successive application of Theorem 6.15. Let us consider

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

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

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

By Theorem 6.15, there are two protocol states St'_1 and St' valid according to the rewrite theory $\mathcal{R}_{\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ such that $trans_\Phi(St_1) = St'_1$,

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

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

□

Finally, we put everything together into one result.

Theorem 6.18 (Soundness and Completeness) *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two protocols and $\mathcal{P}_1 ;_S \mathcal{P}_2$ their composition, as defined in Section 6.3.2. Let $\mathcal{R}_{\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ be the composition rewrite theory defined above in Section 6.4.1, and let $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$ be the rewrite theory associated to the abstract composition semantic described in Section 6.3.3, subject to the restrictions of Remark 1 in Page 149.*

Given a state St valid according to $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$ and an initial state St_{ini} reachable from St by backwards narrowing in $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$, then there are two states St' and St'_{ini} valid according to $\mathcal{R}_{\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ such that (St, St'), $(St_{ini}, St'_{ini}) \in trans_{\Phi}$, and St'_{ini} is reachable from St' by backwards narrowing in $\mathcal{R}_{\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)}$.

Given a state St' valid according to $\mathcal{R}_{\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ and an initial state St'_{ini} reachable from St' by backwards narrowing in $\mathcal{R}_{\Phi(\mathcal{P}_1 ;_S \mathcal{P}_2)}$, then $inv_{\Phi}(St') = St$, $inv_{\Phi}(St'_{ini}) = St_{ini}$, and St_{ini} is reachable from St by backwards narrowing in $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$.

Proof. By Theorems 6.16 and 6.17. □

6.5 Protocol Composition via Synchronization Messages

In Section 6.4 we showed that sequential protocol composition can be implemented using communication between strands via messages sent over the Doler-Yao channel, by performing a protocol transformation. However, as we will show in Section 6.6 this had a serious impact on performance, as well as making it more difficult to write specifications and attack states.

In this section we give a direct implementation of a semantic for protocol composition slightly different to the abstract semantics of Section 6.3. There are two reasons for this, having to do with the fact that the rules in Figures 6.1 and 6.2 are parametrized by the strands in the two composed protocols. First of all, this means that implementing the rules would require a significant modification of Maude-NPA to support the new composition data type. Secondly, the fact that each strand composition produces a new rule means that the number of rewrite rules is significantly increased. Increasing the number of rewrite rules can affect efficiency, since each rewrite rule must be tried at each narrowing step. Here, we present a modified version of Maude-NPA in which composition is achieved via *synchronization messages* that are passed *directly* between a parent and child strand without going through the Dolev-Yao channel. Although, as in the case of composition with respect to protocol transformation, it is necessary to add new rewrite rules, the rules are very similar to those of the basic Maude-NPA semantics in Section 3.4, and require the addition of fewer parametrized rules than for protocol transformation. Communication between strands is achieved using only slight modifications of constructs already present in Maude-NPA.

In Section 6.5.1 we introduce the notion of *synchronization* of protocol strands, a key idea underlying sequential protocol composition. In Section 6.5.2 we explain in detail the new Maude-NPA syntax for the specification of protocol composition via synchronization messages. Section 6.5.3 provides detailed information about the operational semantics of this direct implementation of protocol composition in Maude-NPA. Throughout this chapter we will refer to these syntax and semantics as *composition via synchronization messages syntax and semantics*, respectively. Finally, Section 6.5.4 proves the soundness and completeness of

the semantics in Section 6.5.3 with respect to the abstract semantics in Section 6.3.3, assuming some restrictions on the states reachable via that semantics, thus proving that the semantics in Section 6.5.3 is a *correct* implementation of protocol composition in Maude-NPA. We use our two running examples (NSL-DB and NSL-KD) to illustrate our technique.

6.5.1 Synchronization Data Type Extension

As explained above, the underlying idea of a sequential protocol composition is that the end of the parent's protocol execution is *synchronized* with the beginning of the child's protocol execution. Since in Maude-NPA a protocol execution is denoted by a set of strands, we actually need to provide an infrastructure to express the notion of synchronization among strands, so that the strands of the parent protocol can be in fact "connected" with the strands of the child protocol.

Synchronization of strands can be achieved in Maude-NPA by extending its syntax to define a special type of message that we call *synchronization message*. Several sorts are added, Synch for the synchronization message, Role for user-definable constants denoting the roles in the protocol, RoleConnection for establishing which role strand is the parent and which role is the child, and Mode for choosing between one-to-one composition, denoted by constant `1-1`, and one-to-many composition, denoted by `1-*`. The synchronization message is defined by the following operator:

```
op {_;;_;;_} -> RoleConnection Mode Msg -> Synch .
```

The sort Role contains some constants defined by the user for role names, e.g. `NSL.init` or `NSL.resp`. The sort `RoleConnection` contains just one operator $\_\rightarrow\_$, so that $A \rightarrow B$ indicates that role A is the parent and role B is the child, e.g."`NSL.init -> DB.resp`". The synchronization message is indeed the third parameter, which is just a term of sort Msg, allowing the user to construct any message representing the information exchanged in the synchronization.

6.5.2 Syntax for Protocol Composition via Synchronization Messages

In this section we explain in detail how the Maude-NPA's syntax has been extended with synchronization messages (see Section 6.5.1) in order to support the input and output parameters of Section 6.3.1 and the abstract definition of protocol composition provided in Section 6.3.2. Synchronization messages are actually used to represent protocol compositions directly in the strand specification of the parent and child strands without any protocol transformation. A mapping from the notation for protocol composition of Section 6.3.2 into synchronization messages is described as follows.

Definition 6.19 (Strand Synchronization) *Given two protocols* $\mathcal{P}_1$ *and* $\mathcal{P}_2$ *and a strand composition of the form* $(a, b, Mode)$ *where* a *and* b *are role identifiers of strands in* $\mathcal{P}_1$ *and* $\mathcal{P}_2$*, respectively,* Mode *is 1-1 or 1-*, and the strands of* a *and* b *are of the form* (a) $[\overrightarrow{M}, \{o_1, \ldots, o_n\}]$ *and* (b) $[\{i'_1, \ldots, i'_n\}, \overrightarrow{M'}]$*, where* $i'_1, \ldots, i'_n$ *are the input parameters of* b*'s strand, and* $o_1, \ldots, o_n$ *are the output parameters of* a*'s strand, we define*

$$synch(a, b, Mode) = \left\{ \begin{array}{l} (a)\ [\overrightarrow{M}, \{a \rightarrow b\ ;;\ Mode\ ;;\ (o_1; \cdots; o_n)\}], \\ (b)\ [\{a \rightarrow b\ ;;\ Mode\ ;;\ (i'_1; \cdots; i'_n)\}, \overrightarrow{M'}] \end{array} \right\}$$

Definition 6.20 (Protocol Synchronization) *Given two protocols* $\mathcal{P}_1$ *and* $\mathcal{P}_2$ *that are properly renamed to avoid variable sharing, and a sequential protocol composition* $\mathcal{P}_1 ;_S \mathcal{P}_2 = (\mathcal{P}_1, S, \mathcal{P}_2)$ *where* S *denotes a set of strand compositions of the form* $(a, b, Mode)$*, the protocol synchronization, denoted* $sync(\mathcal{P}_1 ;_S \mathcal{P}_2)$ *is a single protocol which: (i) has signature* $\Sigma_{\mathcal{P}_1} \cup \Sigma_{\mathcal{P}_2} \cup \Sigma_{Synch}$*, where* Σ_{Synch} *is the new signature described in Section 6.5.1, (ii) the equational theory is* $E_{\mathcal{P}_1} \cup E_{\mathcal{P}_2}$ *and (iii) the set of strands is* $synch(S)$*, which is, by definition, the set of strands of the form* $synch(a, b, Mode)$ *for each strand composition* $(a, b, Mode)$ *in* S*.*

As explained in Remark 1 in Page 149, in composition via synchronization messages we assume that in a one-to-many composition a particular instantiation of a parent role can have children belonging to only one role, although that role may be one of any of the roles allowed by the specification. Therefore, note that in that case the protocol composition

$(\mathcal{P}_1 ;_S \mathcal{P}_2)$ may contain two or more strands specifying the same role, the only difference being that these strands may be given different parents or children in their synchronization messages. This is not a problem, since the names of the different parent and child roles disambiguate the different strands. However, for readability it may often be desirable to provide some "syntactic sugar" distinguishing the different role names from each other. We do this in the NSL-DB protocol defined below. One is simply to extend the identifiers of the different parents and/or children so that they become different. This is done for the child strands in NSL-KD, as shown below. The other is to use variables in place of constants, so that two or more strands playing the same role can be represented by a single strand having the same parents (or children). This is also more economical to specify, since the strand only has to be written once. This approach is used to specify the parent strands in NSL-KD, as shown below.

In the following we provide the specification of our two examples of protocol composition, namely the NSL Distance Bounding protocol (NSL-DB) and the NSL Key Distribution protocol (NSL-KD), presented in Sections 6.2.1 and 6.2.2, respectively, using the new synchronization message representation described above. 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 et al., 2008], there are two extra issues related to the DB protocol that must be considered: (i) the initiator of the previous protocol plays the role of the responder in DB and viceversa, and (ii) 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.

Example 6.21 We begin with our example of one-to-one protocol composition, i.e., the NSL-DB protocol. As explained in Section 6.2.1, the initiator of the DB protocol is always the child of the responder of the NSL protocol. The specification of the protocol strands using this syntax is as follows where the symbol $*$ denotes the exclusive-or operator:

```
*** NSL protocol
:: r ::
[ nil | +(pk(B, n(A,r) ; A)) ,
        -(pk(A, n(A,r) ; NB ; B )),
        +(pk(B, NB)),
        {NSL-init -> DB-resp ;; 1-1 ;; A ; B ; n(A,r)}, nil ] &
:: r ::
[ nil | -(pk(B,NA ; A)),
        +(pk(A, NA ; n(B,r) ; B)),
        -(pk(B,n(B,r))),
        {NSL-resp -> DB-init ;; 1-1 ;; A ; B ; NA}, nil ] &
*** Distance Bounding protocol
:: r' ::
[ nil | {NSL-resp -> DB-init ;; 1-1 ;;  A ; B ; NA},
        +(n(B,r')),
        -(NA * n(B,r')), nil] &
:: nil ::
[ nil |  {NSL-init -> DB-resp ;; 1-1 ;;  A ; B ; NA },
         -(N),
         +(NA * N), nil ]
```

■

Example 6.22 Let us now continue with our example of a one-to-many protocol composition, i.e., the NSL-KD protocol. As explained in Section 6.2.2, the initiator of the session key protocol can be the child of either the initiator or responder of the NSL protocol. These two possible compositions for each KD protocol need to be specified by different synchronization messages and, therefore, it is necessary to duplicate each strand, as shown below. In this case, we also change the names of the roles to improve the readability of the specification. The specification of the strands of the NSL-KD protocol using the syntax for protocol composition via synchronization messages is as follows:

```
*** NSL protocol
:: r ::
[ nil | +(pk(B, n(A,r) ; A)) ,
        -(pk(A, n(A,r) ; NB ; B )),
        +(pk(B, NB)),
```

```
        {NSL-init -> ROLE ;;
         1-* ;;  A ;  B ; h(n(A,r) , NB) },  nil ] &
:: r ::
[ nil | -(pk(B,NA ; A)),
        +(pk(A, NA ; n(B,r) ; B)),
        -(pk(B,n(B,r))),
        {NSL-resp -> ROLE ;;
         1-* ;; A ; B ; h(NA , n(B,r))}, nil ]  &
*** KD protocol
:: r' ::
[ nil | {NSL-init -> initA-kd ;; 1-* ;; A ; B ; MKe },
        +(e(MKe, skey(A, n(A,r')))) ,
        -(e(MKe, skey(A, n(A,r')) ; N)),
        +(e(MKe, N)), nil] &
:: r' ::
[ nil |  {NSL-resp -> respB-kd ;; 1-* ;; A ; B ;  MKe },
         -(e(MKe, skey(A, NA'))),
         +(e(MKe, skey(A, NA') ; n(B,r'))),
         -(e(MKe, n(B,r'))), nil ] &
----
:: r' ::
[ nil | {NSL-resp -> initB-kd ;; 1-* ;; A ; B ;  MKe },
        +(e(MKe, skey(B, n(B,r')))),
        -(e(MKe, skey(B, n(B,r')) ; N)),
        +(e(MKe, N)), nil ] &
:: r' ::
[ nil | {NSL-init -> respA-kd ;; 1-* ;; A ; B ; MKe },
        -(e(MKe, skey(B, NB'))),
        +(e(MKe, skey(B, NB') ; n(A,r'))),
        -(e(MKe, n(A,r'))), nil ]
```

where ROLE is a variable denoting a role. ■

6.5.3 Operational Semantics of Composition via Synchronization Messages

In Section 6.3.3 we provided an operational semantics based on extra transitions rules generated for each possible protocol composition and we differentiated between rules generated for one-to-one compositions and

rules generated for one-to-many compositions. However, in this section we propose a simplified version of that operational semantics, which we call *composition via message synchronization semantics*, so that now we just have two *generic transition rules* and a set of *generated transition rules* for each strand in the same spirit of Rule (3.1) and Rules (3.4).

The two generic transition rules for protocol composition via synchronization messages are described in Figure 6.6. Note that these transition rules are written in a *forwards* way but will be executed backwards, as the basic transition rules of Section 3.4 and the abstract composition semantics of Section 6.3.3. The first generic transition Rule (6.4) is applicable to both one-to-one compositions and one-to-many compositions. This rule achieves the synchronization between both strands by means of the synchronization message. The second generic Rule (6.5) is applicable only to one-to-many compositions and represents the synchronization of a parent and a child without disabling the synchronization message of the parent.

These two generic rules synchronize an output parameter of an existing parent strand with an input message of an existing child strand. Both strands must be present in the state. The difference between a one-to-one and one-to-many composition is that the output parameter of the parent strand is kept in the same position of the parent strand for further synchronizations with other children strands.

As it happens in the basic Maude-NPA operational semantics of Section 3.4, we generate extra transitions rules from strands, in this case for protocol composition, as shown in Figure 6.7. Transition rules of the form (6.6), when executed backwards, allow adding to the state a new parent strand, whose output parameters will be synchronized with the input parameters of an already existing child strand. Note that the generated transitions rules (6.6) are apply to *both* of the one-to-one or one-to-many composition cases. In each case, they describe a parent synchronizing with its first child.

For example, given the composition of the NSL initiator's strand and the DB responder's strand, where both strands were defined in Example 6.21, for Alice's strand

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

$$SS \,\&\, [L \mid \{\text{a} \rightarrow \text{b} \;;;\; \text{Mode} \;;; \text{Message}\}] \,\&\, [nil \mid \{\text{a} \rightarrow \text{b} \;;;\; \text{Mode} \;;; \text{Message}\}, L'] \,\&\, IK \rightarrow SS \,\&\, [L, \{\text{a} \rightarrow \text{b} \;;;\; \text{Mode} \;;; \text{Message}\} \mid nil] \,\&\, [\{\text{a} \rightarrow \text{b} \;;;\; \text{Mode} \;;; \text{Message}\} \mid L'] \,\&\, IK \tag{6.4}$$

$$SS \,\&\, [L \mid \{\text{a} \rightarrow \text{b} \;;;\; \texttt{1-*} \;;; \text{Message}\}] \,\&\, [nil \mid \{\text{a} \rightarrow \text{b} \;;;\; \texttt{1-*} \;;; \text{Message}\}, L'] \,\&\, IK \rightarrow SS \,\&\, [L \mid \{\text{a} \rightarrow \text{b} \;;;\; \texttt{1-*} \;;; \text{Message}\}] \,\&\, [\{\text{a} \rightarrow \text{b} \;;;\; \texttt{1-*} \;;; \text{Message}\} \mid L'] \,\&\, IK \tag{6.5}$$

where:

L, L' are variables of the sort for lists of input and output messages ($+m$,$-m$),

IK is a variable of the sort for sets of intruder facts ($m \in \mathcal{I}, m \notin \mathcal{I}$),

SS is a variable of the sort for sets of strands,

Message is a variable of sort Msg,

a, b are variables of sort Role, and

Mode is a variable of sort `Mode`

Figure 6.6: Generic forward transition rules for composition via synchronization messages

```
+(pk(B, NB)),
{NSL-init -> DB-resp ;; 1-1 ;; A ; B ; n(A,r)}, nil ] &
```

we add the following transition rule generated by Rules (6.6)

For each strand definition $[\overrightarrow{M_a}, \{\text{a} \rightarrow \text{b} \;;;\; \text{Mode} \;;; \text{Message}\}]$,
we add a rule of the form:

$$\begin{aligned} & SS \,\&\, [\overrightarrow{M_a} \mid \{\text{a} \rightarrow \text{b} \;;;\; \text{Mode} \;;; \text{Message}\}] \,\& \\ & \qquad [nil \mid \{\text{a} \rightarrow \text{b} \;;;\; \text{Mode} \;;; \text{Message}\}, L'] \,\& \, IK \\ \rightarrow & SS \,\&\, [\{\text{a} \rightarrow \text{b} \;;;\; \text{Mode} \;;; \text{Message}\} \mid L'] \,\& \, IK \end{aligned} \tag{6.6}$$

where:

L' are variables of the sort for lists of input and output messages $(+m, -m)$,
IK is a variable of the sort for sets of intruder facts $(m \in \mathcal{I}, m \notin \mathcal{I})$,
SS is a variable of the sort for sets of strands,
Message is a variable of sort Msg,
a, b are variables of sort Role, and
Mode is a variable of sort `Mode`

Figure 6.7: Generated forward transition rules for composition via synchronization messages

$$\begin{aligned} & :: r :: \\ & [\, nil, + (pk(B, n(A,r); A)), \\ & \qquad - (pk(A, n(A,r); NB; B)), \\ & \qquad + (pk(B, NB)), \\ & \qquad \mid \{NSL - init \rightarrow DB - resp;;1-1;;A;B;n(A,r)\}, nil\,] \,\& \\ & :: RR :: \\ & [\, nil \mid \{NSL - init \rightarrow DB - resp;;1-1;;A;B;n(A,r)\}, L\,] \,\& \\ & SS \,\&\, IK \\ & \quad \longrightarrow \\ & :: RR :: \\ & \quad [\, nil, \{NSL - init \rightarrow DB - resp;;1-1;;A;B;n(A,r)\} \mid L\,] \,\& \\ & SS \,\&\, IK \end{aligned}$$

Thus, for a protocol composition $\mathcal{P}_1;_S \mathcal{P}_2$, the rewrite rules governing protocol execution in composition via synchronization messages are $R_{synch(\mathcal{P}_1;_S\mathcal{P}_2)} = \{(3.1), (3.2), (3.3)\} \cup (3.4) \cup \{(6.4), (6.5)\} \cup (6.6)$.

Here, the reader can realize that this synchronization semantics for protocol composition contains two generic transition rules and one transition rule for each protocol composition, whereas the protocol composition presented in Section 6.3 produces several transition rules for each protocol composition. Indeed, this simpler semantics for protocol composition requires fewer rules distinguishing one-to-one and one-to-many compositions than the abstract semantics.

6.5.4 Soundness and Completeness

In this section we prove soundness and completeness of the operational semantics composition via synchronization messages presented in Section 6.5.3 with respect to the abstract compositional operational Semantics of Section 6.3.3, subject to the assumption described in Remark 1, that any given instantiation of a parent role can have children of only one type of role, although it may have multiple choices for the one role.

First, we must relate protocol states using the protocol composition rewrite rules of Section 6.3.3 and protocol states in the composition via synchronization messages.

Definition 6.23 (Functions *trans* and *inv*) *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two protocols and $\mathcal{P}_1 ;_S \mathcal{P}_2$ their composition. Let $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$ be the rewrite theory associated in Section 6.3.3 to the abstract protocol composition $\mathcal{P}_1 ;_S \mathcal{P}_2$ and $\mathcal{R}_{synch(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ be the rewrite theory associated in Section 6.5.3 to composition via synchronization messages. We define the following:*

1. *the relation trans between states valid according to the rewrite theory $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$ and states valid according to the rewrite theory $\mathcal{R}_{synch(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ as specified in Figure 6.8.*

2. *the function inv from states valid according to the rewrite theory $\mathcal{R}_{synch(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ into states valid according to the rewrite theory $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$ as specified in Figure 6.9.*

$$trans(St) = \begin{cases} (b)[\{a \to b\,;;\, Mode\,;;\, I_b^\sharp\},\ \overrightarrow{b_1} \mid \overrightarrow{b_2}] \ \&\ St' & \text{if } (b)[\{\overrightarrow{I_b}\}, \overrightarrow{b_1} \mid \overrightarrow{b_2}] \in St, \\ & \quad (a, b, Mode) \in \mathcal{P}_1 ;_S \mathcal{P}_2, \\ & \quad trans(St - (b)) = St' \\ (b)[nil \mid \{a \to b\,;;\, Mode\,;;\, I_b^\sharp\},\ \overrightarrow{b_1}] \ \&\ St' & \text{if } (b)[nil \mid \{\overrightarrow{I_b}\}, \overrightarrow{b_1}] \in St, \\ & \quad (a, b, Mode) \in \mathcal{P}_1 ;_S \mathcal{P}_2, \\ & \quad trans(St - (b)) = St' \\ (a)[\overrightarrow{a_1} \mid \overrightarrow{a_2}, \{a \to b\,;;\, Mode\,;;\, O_a^\sharp\}] \ \&\ St' & \text{if } (a)[\overrightarrow{a_1} \mid \overrightarrow{a_2}, \{\overrightarrow{O_a}\}] \in St, \\ & \quad (a, b, Mode) \in \mathcal{P}_1 ;_S \mathcal{P}_2, \\ & \quad trans(St - (a)) = St' \\ (a)[\overrightarrow{a_1}, \{a \to b\,;;\, Mode\,;;\, O_a^\sharp\} \mid nil] \ \&\ St' & \text{if } (a)[\overrightarrow{a_1}, \{\overrightarrow{O_a}\} \mid nil] \in St, \\ & \quad (a, b, Mode) \in \mathcal{P}_1 ;_S \mathcal{P}_2, \\ & \quad trans(St - (a)) = St' \\ St & \text{otherwise} \end{cases}$$

where $I^\sharp$ (resp. $O^\sharp$) is equal to $\overrightarrow{I}$ (resp. $\overrightarrow{O}$) by replacing the comma "," by a semicolon ";" to denote concatenation of input and output parameters, e. g. input parameters $\overrightarrow{I} = \{A\,,\, B\,,\, NA\}$ is written as the sequence $I^\sharp = A\,;\, B\,;\, NA$.

Figure 6.8: Relation *trans* between states valid according to the rewrite theory $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$ and states valid according to the rewrite theory $\mathcal{R}_{synch(\mathcal{P}_1 ;_S \mathcal{P}_2)}$

The following auxiliary results become crucial and ensure that there is an appropriate connection between states of both rewrite theories.

Lemma 6.24 *The function inv is total and relation trans is the inverse of inv. Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two protocols and $\mathcal{P}_1 ;_S \mathcal{P}_2$ their composition. Let $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$ be the rewrite theory associated in Section 6.3.3 to the protocol composition $\mathcal{P}_1 ;_S \mathcal{P}_2$ and $\mathcal{R}_{synch(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ be the rewrite theory associated in Section 6.5.3 to the composition via synchronization messages.*

The function inv defined in Definition 6.23 defines a total function from terms of sort State *in $\mathcal{R}_{synch(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ back to terms of sort* State *in $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$. The function trans defined in Definition 6.23 is the inverse of inv.*

Proof. By structural induction on the definitions of Figure 6.8 and 6.9. □

Note that *inv* is not injective, since two states that differ only by the fact that some strands have different child roles defined in the output

$$inv(St) = \begin{cases} (b)[\{\overrightarrow{I_b}\},\ \overrightarrow{b_1} \mid \overrightarrow{b_2}] \ \&\ St' & \text{if } (b)[\{a \rightarrow b\,;;\, Mode\,;;\, I_b^\sharp\}, \overrightarrow{b_1} \mid \overrightarrow{b_2}] \in St, \\ & \quad inv(St - (b)) = St' \\ (b)[nil \mid \{\overrightarrow{I_b}\},\ \overrightarrow{b_1}] \ \&\ St' & \text{if } (b)[nil \mid \{a \rightarrow b\,;;\, Mode\,;;\, I_b^\sharp\}, \overrightarrow{b_1}] \in St, \\ & \quad inv(St - (b)) = St' \\ (a)[\overrightarrow{a_1} \mid \overrightarrow{a_2}, \{\overrightarrow{O_a}\}] \ \&\ St' & \text{if } (a)[\overrightarrow{a_1} \mid \overrightarrow{a_2}, \{a \rightarrow b\,;;\, Mode\,;;\, O_a^\sharp\}] \in St, \\ & \quad inv(St - (a)) = St' \\ (a)[\overrightarrow{a_1}, \{\overrightarrow{O_a}\} \mid nil] \ \&\ St' & \text{if } (a)[\overrightarrow{a_1}, \{a \rightarrow b\,;;\, Mode\,;;\, O_a^\sharp\} \mid nil] \in St, \\ & \quad inv(St - (a)) = St' \\ St & \text{otherwise} \end{cases}$$

Figure 6.9: Function *inv* mapping from states valid according to the rewrite theory $\mathcal{R}_{synch(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ onto states valid according to the rewrite theory $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$

parameters (respectively, different parent roles defined in the input parameters), will be mapped to the same state by *inv*. Thus its inverse *trans* is not a function.

Let us now relate backwards narrowing steps using the rewrite theory associated to the composition via synchronization messages of Section 6.5.3 (i.e., $\mathcal{R}_{synch(\mathcal{P}_1 ;_S \mathcal{P}_2)}$) w.r.t. backwards narrowing using the rewrite theory associated to the abstract protocol composition of Section 6.3.3 (i.e., $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$). Note that in this case a backwards narrowing step performed with a rule of $\mathcal{R}_{synch(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ always corresponds to one backwards narrowing step with a rule of $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$, since no extra messages are introduced to synchronize parent and child strands.

Lemma 6.25 (One-step correspondence) *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two protocols and $\mathcal{P}_1 ;_S \mathcal{P}_2$ their composition. Let $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$ be the rewrite theory associated in Section 6.3.3 to the abstract protocol composition $\mathcal{P}_1 ;_S \mathcal{P}_2$, subject to the restrictions of Remark 1 in Page 149, and $\mathcal{R}_{synch(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ be the rewrite theory associated in Section 6.5.3 to composition via synchronization messages.*

If there are two protocol states St_1 and St_2 valid according to the rewrite theory $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$ and $St_1 \overset{1}{\leadsto}_{\rho, \mathcal{R}^{-1}_{\mathcal{P}_1 ;_S \mathcal{P}_2}, E_{\mathcal{P}_1 ;_S \mathcal{P}_2}} St_2$, then there exists St_1' and St_2' valid according to the rewrite theory $\mathcal{R}_{synch(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ such that $trans(St_1) = St_1'$, $trans(St_2) = St_2'$, and $St_1' \overset{1}{\leadsto}_{\sigma, \mathcal{R}^{-1}_{synch(\mathcal{P}_1 ;_S \mathcal{P}_2)}, E_{\mathcal{P}_1 ;_S \mathcal{P}_2}} St_2'$.

If there are two protocol states St_1' and St_2' valid according to the rewrite theory $\mathcal{R}_{synch(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ such that $inv(St_1') = St_1$, $inv(St_2') = St_2$,

and $St_1' \overset{1}{\leadsto}_{\sigma,\mathcal{R}^{-1}_{synch(\mathcal{P}_1 ;_S \mathcal{P}_2)},E_{\mathcal{P}_1 ;_S \mathcal{P}_2}} St_2'$*, then* $St_1 \overset{1}{\leadsto}_{\rho,\mathcal{R}^{-1}_{\mathcal{P}_1 ;_S \mathcal{P}_2},E_{\mathcal{P}_1 ;_S \mathcal{P}_2}} St_2$.

Proof. (Sketch) We prove the result by case analysis of the rewrite rules applicable to terms St_1 and St_2 (resp. St_1' and St_2').

(a) When the Rules (3.1),(3.2),(3.3), and (3.4) are applied to regular incoming and outcoming messages in the strands of St_1 (resp. St_1'), we have one narrowing step from the associated state St_1' (resp. St_1) applying the same type of rule.

(b) Rule (6.1) corresponds to an application of Rule (6.4) (synchronizing the input parameters of the child strand with the output parameters of the parent strand).

(c) Rule (6.2) corresponds to an application of Rule (6.6) (introducing a new parent strand and composing it with an existing child strand).

(d) Rule (6.3) corresponds to an application of a rule of the form (6.5) (synchronizing the output parameters of the parent strand with the already accepted input parameters of the child strand, but without moving the bar in the parent strand). Given that our requirement that the restrictions of Remark 1 hold, Rule (6.3) can be applied if and only if the corresponding rule of the form (6.5) can be applied. □

Finally, we can put everything together into the following result.

Theorem 6.26 (Soundness and Completeness) *Let* $\mathcal{P}_1$ *and* $\mathcal{P}_2$ *be two protocols and* $\mathcal{P}_1 ;_S \mathcal{P}_2$ *their composition, as defined in Section 6.3.2. Let* $\mathcal{R}_{synch(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ *be the rewrite theory associated to composition via synchronization messages defined above in Section 6.5.3, and let* $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$ *be the rewrite theory associated to the abstract protocol composition, as described in Section 6.3.3, subject to the restrictions of Remark 1 in Page 149.*

Given a state St valid according to $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$ *and an initial state* St_{ini} *reachable from St by backwards narrowing in* $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$*, then there are two*

states St' *and* St'_{ini} *valid according to* $\mathcal{R}_{synch(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ *such that* (St, St'), $(St_{ini}, St'_{ini}) \in trans$, *and* St'_{ini} *is reachable from* St' *by backwards narrowing in* $\mathcal{R}_{synch(\mathcal{P}_1 ;_S \mathcal{P}_2)}$.

Given a state St' *valid according to* $\mathcal{R}_{synch(\mathcal{P}_1 ;_S \mathcal{P}_2)}$ *and an initial state* St'_{ini} *reachable from* St' *by backwards narrowing in* $\mathcal{R}_{synch(\mathcal{P}_1 ;_S \mathcal{P}_2)}$, *then* $inv(St') = St$, $inv(St'_{ini}) = St_{ini}$, *and* St_{ini} *is reachable from* St *by backwards narrowing in* $\mathcal{R}_{\mathcal{P}_1 ;_S \mathcal{P}_2}$.

Proof. By successive applications of Lemma 6.25. □

6.6 Experimental Evaluation

In this section we further explore composition via protocol transformation versus composition via synchronization messages comparing them for ease of use and simplicity. Furthermore, we present some experimental results about the performance the two approaches. First, in Section 6.6.1 we show the attack for the NSL-DB explained in Section 6.2.1. Then we fix the NSL-DB protocol using a hash function, as explained in Section 6.2.1, and show that the protocol is verified as secure by our tool, i.e., the search space is finite and no attack is found. Moreover, in Section 6.6.2 we show that the NSL-KD protocol presented in Section 6.2.2 is also verified as secure by the Maude-NPA. Each time 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 regular execution proofs have not been included in this chapter, though they are available online (see below).

In Section 6.6.3 we provide more details of the experiments and compare the results obtained using both techniques. All the experiments, including the source Maude-NPA files and the generated outputs, can be found at:

`http://www.dsic.upv.es/~sescobar/Maude-NPA/composition.html`

6.6.1 The NSL-DB Protocol

We start with the NSL-DB protocol composition. As explained in Section 6.2.1, this protocol has an attack in which the honest principal B

thinks that he has heard from a principal D (who may or may not be honest), but who has actually heard from an honest principal A. This covers, for example, the case in which D is dishonest, and tries to pass on an honest principal's authenticated response as his own. This attack is represented in Maude-NPA by an attack state pattern, according to the protocol specification of Example 6.21, where: (i) the first strand is Alice talking to some principal C acting as NSL initiator and connecting to a DB responder, (ii) the second strand is Bob talking to some principal D acting as DB initiator and receiving data from NSL responder, and (iii) we include disequalities constraints for principal names, namely $a \neq D$ and $C \neq b$.

More specifically, the attack state pattern using the protocol transformation technique is as follows:

```
eq ATTACK-STATE(0)
=  :: r ::
   [ nil, +(NSL-init),
          +(pk(C,n(a,r) ; a)),
          -(pk(a, n(a,r) ; NC ; C )),
          +(pk(C, NC)) |
          -(DB-resp(r#)),
          +({NSL-init(r#) . a . C . n(a,r)}),  nil ]  &
   :: r', r# ::
  [ nil, +(DB-init(r#)),
         -({NSL-resp(r#) . D  . b . n(a,r) }),
         +(n(b,r')), -(n(b,r') *  n(a,r)) | nil ]
  || (a != D) , (C != b)
  || nil
  || nil
  || nil
 [nonexec] .
```

whereas in the composition via synchronization messages the attack pattern is as shown below:

```
eq ATTACK-STATE(0)
= :: r ::
  [ nil, +(pk(C,n(a,r) ; a)),
         -(pk(a, n(a,r) ; NC ; C)),
```

```
        +(pk(C, NC)) |
        {NSL-init -> DB-resp ;; 1-1 ;; a ; C ; n(a,r)}, nil] &
  :: r' ::
  [ nil, {NSL-resp -> DB-init ;; 1-1;; D ; b ; n(a,r)},
        +(n(b,r')), -(n(a,r) * n(b,r'))  | nil]
  || (a != D) , (C != b)
  || nil
  || nil
  || nil
 [nonexec] .
```

Here the reader can see that the attack state pattern for the transformed protocol is more complex and hence more error prone than the attack state pattern for composition via synchronization messages, since the introduction of fresh variables for protocol composition has to be done manually. Also, the attack state pattern looks more artificial in the protocol transformation because of the back and forth messages.

The backwards search from the the NSL-DB attack state using composition via synchronization messages finds an initial state from which it is reachable, and thus demonstrates a distance hijacking attack. The exchange of messages of this attack is as explained in Section 6.2.1 in Page 141. The backwards search from the corresponding attack state for protocol transformation does not terminate[3] due to a state space explosion, and no initial state is found up to the depth reached by the analysis. We then considered other attacks similar to the distance hijacking attack which however produced a smaller search space. In the following attack, we asked whether it is possible for an attacker to use an initiator A's nonce to participate in the distance-bounding part of the protocol without Alice having completed the corresponding NSL strand. This, besides being simpler, required only that the bar move one step forward in the NSL strand, and produced a smaller search space in which the protocol transformation version was able to find an attack, and to terminate on the corrected version of the protocol, giving us a better opportunity to compare the performance of the two approaches.

The attack state is given below:

[3]In [15] we reported termination, but this turned out to be a result of a bug in Maude-NPA's management of disequality constraints, which has since been corrected. The development of new semantics and implementation helped us to discover this bug.

```
eq ATTACK-STATE(1)
=  :: r ::
   [ nil, +(NSL-init),
          +(pk(C,n(a,r) ; a)) |
          -(pk(a, n(a,r) ; NC ; C )),
          +(pk(C, NC)),
          -(DB-resp(r#)),
          +({NSL-init(r#) . a . C . n(a,r)}),  nil ]  &
   :: r', r# ::
  [ nil, +(DB-init(r#)),
         -({NSL-resp(r#) . D  . b . n(a,r) }),
         +(n(b,r')), -(n(b,r') *  n(a,r)) | nil ]
  || (a != D) , (C != b)
  || nil
  || nil
  || nil
 [nonexec] .
```

As explained in Section 6.2.1, the distance hijacking attack can be avoided using a hash function. The protocol strands of the fixed version of the DB protocol following the protocol transformation are as follows:

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

whereas in the composition via synchronization messages the strands are of the form shown below:

```
:: r' ::
[ nil | {NSL-resp -> DB-init ;; 1-1 ;; A ; B ; NA},
        +(n(B,r')),
        -(h(A, NA) * n(B,r')), nil] &
:: nil ::
[ nil |  {NSL-init -> DB-resp ;; 1-1 ;; A ; B ; NA },
         -(N),
         +(h(A, NA) * N), nil ]
```

The previous property for the NSL-DB is specified in the new version of the protocol with the following attack state pattern using the protocol transformation:

```
eq ATTACK-STATE(0)
=  :: r ::
   [ nil, +(NSL-init),
          +(pk(C,n(a,r) ; a)),
          -(pk(a, n(a,r) ; NC ; C )),
          +(pk(C, NC)) |
          -(DB-resp(r#)),
          +({NSL-init(r#) . a . C . n(a,r)}),  nil ]  &
   :: r', r# ::
  [ nil, +(DB-init(r#)),
         -({NSL-resp(r#) . D  . b . n(a,r) }),
         +(n(b,r')), -(n(b,r') * h(D,n(a,r))) | nil ]
  || (a != D) , (C != b)
  || nil
  || nil
  || nil
 [nonexec] .
```

whereas in the composition via synchronization messages the attack state pattern is as follows:

```
eq ATTACK-STATE(0)
= :: r ::
  [ nil, +(pk(C, n(a,r) ; a)),
         -(pk(a, n(a,r) ; NC ; C)),
         +(pk(C, NC)) |
         {NSL-init -> DB-resp ;; 1-1 ;; a ; C ; n(a,r)}, nil] &
  :: r' ::
  [ nil, {NSL-resp -> DB-init ;; 1-1 ;; D ; b ; n(a,r)},
         +(n(b,r')), -(h(D, n(a,r)) * n(b,r')) | nil]
 || (a != D) , (C != b )
 || nil
 || nil
 || nil
[nonexec] .
```

The analysis of this protocol composition using the composition via synchronization messages, terminates finding no attack (see Table 6.1 below). Thus, the protocol is secure for such an attack state pattern. However, as in the case NSL-DB protocol, the analysis using the protocol transformation does not terminate due to state space explosion and, thus, the security of the protocol for such an attack state pattern cannot be proved. Therefore, we proceed in a similar way as we did in the NSL-DB protocol. That is, we considered an attack state pattern of the form:

```
eq ATTACK-STATE(1)
=  :: r ::
   [ nil, +(NSL-init),
          +(pk(C,n(a,r) ; a)) |
          -(pk(a, n(a,r) ; NC ; C )),
          +(pk(C, NC)),
          -(DB-resp(r#)),
          +({NSL-init(r#) . a . C . n(a,r)}),  nil ]  &
   :: r', r# ::
  [ nil, +(DB-init(r#)),
         -({NSL-resp(r#) . D  . b . n(a,r) }),
         +(n(b,r')), -(n(b,r') * h(D,n(a,r))) | nil ]
  || (a != D) , (C != b)
  || nil
  || nil
  || nil
 [nonexec] .
```

The analysis of the protocol using the protocol transformation terminates, finding no initial state from which this more specific attack state pattern is reachable. The same result is obtained when using composition via synchronization messages.

6.6.2 The NSL-KD Protocol

For the NSL-KD protocol presented in Section 6.2.2 we may wish to guarantee that a dishonest principal is not able to learn the secret key of an honest principal. This property is represented by an attack state, according to the protocol of Example 6.22, where the first strand is an initiator of the KD protocol generating the session key `skey(a,n(a,r'))`,

the second strand is a responder of the KD protocol using the same session key `skey(a,n(a,r'))`, and we ask whether the intruder can learn this session key by adding the fact `skey(a,n(a,r'))` to the intruder knowledge.

More specifically, in the protocol tranformation the attack pattern is of the following form:

```
eq ATTACK-STATE(0)
 = :: r' ::
    [ nil, +(KD-init),
           -({NSL-init(r#) . A . B . MKe }),
           +(e(MKe, skey(a, n(a,r')))) ,
           -(e(MKe, skey(a, n(a,r')) ;  n(b,r))),
           +(e(MKe,  n(b,r))) | nil]
    &
    :: r ::
    [ nil, +(KD-resp),
           -({NSL-resp(r#) . a . B . MKe }),
           -(e(MKe, skey(a, n(a,r')))),
           +(e(MKe, skey(a, n(a,r')) ; n(b,r))),
           -(e(MKe, n(b,r))) | nil ]
   || skey(a, n(a,r')) inI
   || nil
   || nil
   || nil
 [nonexec] .
```

whereas for the composition via synchronization message is specified as follows:

```
eq ATTACK-STATE(0)
 = :: r' ::
   [ nil, {NSL-init -> initA-kd ;; 1-* ;;  a ; b ; MKe },
          +(e(MKe, skey(a, r'))),
          -(e(MKe, skey(a,r') ; N)),
          +(e(MKe, N))  | nil]
    &
   :: r ::
   [ nil, {NSL-resp -> respB-kd ;; 1-* ;; a ; b ;  MKe },
          -(e(MKe, skey(a, r'))),
```

```
          +(e(MKe, skey(a, r') ; n(B,r))),
          -(e(MKe, n(b,r))) | nil ]
   || skey(a, r') inI
   || nil
   || nil
   || nil
[nonexec] .
```

Here again the reader can see that the attack state pattern for the transformed protocol lacks visual information about what is really happening, since we have two strands but each is for a different protocol composition. Instead, the attack state pattern for the composition via synchronization messages clearly shows that there are two different one-to-many strand compositions.

In this case, the desired property is satisfied by the NSL-KD, since the analysis terminates using both the protocol transformation and the composition via synchronization messages techniques, finding no initial state for the attack state pattern described above.

6.6.3 Performance Comparison

In this section we show in detail the results of the experiments explained in Sections 6.6.1 and 6.6.2. Table 6.1 gathers the results of the analysis of these protocol compositions, i.e., (i) the composition of the NSL and DB protocols (NSL-DB), (ii) the composition of the NSL and the fixed version of the DB protocol (NSL-DB-fix), and (iii) the composition of the NSL and the KD protocols (NSL-KD). Note that for the NSL-DB and NSL-DB-fix protocols we consider the two attack state patterns shown above: the more generic, denoted as "a0", e.g. NSL-DB-a0; and the more specific, denoted as "a1", e.g. NSL-DB-a1. For each protocol composition we provide the following information. For each technique, i.e., protocol transformation and composition via synchronization messages (referred as composition via DSM in the table header), the column "Sec?" shows whether the technique sucessfully proved the protocol composition is secure, i.e. Maude-NPA generated a finite search space finding no attacks, or insecure, i.e, Maude-NPA found an attack. When Maude-NPA did not obtain a concluding result, i.e., when the analysis did not terminate (e.g. because of an state space explosion) and no initial state was found up

	Protocol Transformation				Composition via DSM				
Protocol	Sec?	Finite?	Depth	States	Sec?	Finite?	Depth	States	DSM / PT
NSL-DB-a0	?	No	10	3434	No	Yes	16	1337	-
NSL-DB-a1	No	No	16	1526	No	Yes	13	259	0.17
NSL-DB-fix-a0	?	No	10	2650	Yes	Yes	19	1690	-
NSL-DB-fix-a1	Yes	Yes	17	273	Yes	Yes	16	103	0.38
NSL-KD	Yes	Yes	19	1192	Yes	Yes	16	517	0.44

Table 6.1: Experiments with sequential protocol compositions

to the depth reached by the analysis, we write in this column "?". The column "Finite?" indicates whether Maude-NPA generated a finite state search space or not, i.e. whether the analysis of such protocol composition terminated or not. The column "Depth" provides the depth of the analysis, i.e., the number of reachability steps performed by Maude-NPA until: (i) it generates a finite search space with no attacks in the case of a secure composition, (ii) it finds the attack in the case of an insecure composition, or (iii) the analysis finished before obtaining a concluding result; whereas the column "States" shows the total number of states generated during the analysis up to the indicated depth. For the composition via synchronization messages, the column "DSM / PT" shows the state space reduction as the number of states explored by the synchronization messages method (DSM) divided by the number of states explored by the protocol transformation method (PT). When Maude-NPA did not obtain concluding results using the protocol transformation technique we write "-" in this column.

As mentioned above, the results gathered in Table 6.1 show that the analysis of a protocol composition performed with the composition via synchronization messages has better performance than when it is carried out via protocol transformation. On the one hand, the protocol transformation fails to provide a conclusive result about the security of two of our experiments, namely the analysis of the NSL-DB and NSL-DB-fixed protocol compositions for the distance hijacking attack state pattern, whereas this problem does not occur with the composition via synchronization messages. On the other hand, the composition via synchronization messages generates a finite state search space in all cases, whereas with the protocol transformation this happens in only two cases.

Furthermore, when both techniques obtain concluding results, the

analysis performed with the composition via synchronization messages is more efficient and with a much smaller state space than when it is performed with the protocol transformation. The composition via synchronization messages also requires less reachability analysis steps than the protocol transformation. For example, the analysis of the NSL-DB composition with the protocol transformation terminates after 16 steps, whereas with the composition via synchronization messages it terminates after 13 steps. Moreover, the number of states generated with the composition via synchronization messages is also smaller than with the protocol transformation (a 0.33 factor on average). For example, the analysis of the NSL-DB composition with the protocol transformation generates 1526 states, whereas with the composition via synchronization messages generates 259 states, i.e. it reduces the search space to 17% of the search space generated with the protocol transformation. The reason for this behavior is not only because the depth of an analysis with synchronization messages is lower than with the protocol transformation, but also because the composition via synchronization messages does not require the exchange of extra messages to perform strand composition that is used in the protocol transformation, and thus avoids state space expansion due to interleaving.

6.7 Conclusions

In this chapter we have provided a framework to support *dynamic sequential protocol compositions* in Maude-NPA, i.e., protocols are specified in a modular way and can be composed when desired during the verification process. More specifically, we have presented two different techniques to support sequential protocol compositions: namely protocol composition via a protocol transformation, which does not require to modify the tool, and the direct implementation of protocol composition in Maude-NPA, which requires extending its operational semantics, but allows a more straightforward protocol specification. Moreover, we have performed several experiments to compare both techniques, and the results obtained show that the direct implementation allows a more efficient analysis of protocol compositions in Maude-NPA.

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 and Carrel, 1998] 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.

Chapter 7

Protocol Indistinguishability in Maude-NPA

In this chapter we explain how indistinguishability properties can be specified and verified in Maude-NPA. This kind of properties are used to verify whether an attacker who interacts with two different instances of a protocol involving different data is able or not to tell the difference.

First, in Section 7.1 we motivate the specification and verification of indistinguishability properties in Dolev-Yao models, and provide an overview of our approach. We then present our formal definition of indistinguishability in Section 7.2 using the fowards semantics given in Chapter 5 (see Section 5.3) as a reference model, and prove that it can be analyzed in Maude-NPA assuming that the equational theory used for $\mathcal{P}_1$ and $\mathcal{P}_2$ has a finite variant decomposition. Section 7.3 explains how to analyze indistinguishability properties in Maude-NPA. In Section 7.4 we discuss the results of the preliminary experiments we have performed to verify indistinguishability properties in Maude-NPA, following the method presented in this chapter. Finally, we summarize our conclusions in Section 7.5.

These results have been published in [Santiago et al., 2014b].

7.1 Motivation

Traditionally, properties proved about Dolev-Yao specifications of protocols fall into two classes: secrecy and correspondence. *Correspondence*

holds if, whenever certain actions have occurred, one can guarantee that certain other actions have occurred in a specified order; correspondence properties are thus used to reason about authentication. *Secrecy* (also called *simple secrecy*) holds for a term if the attacker never sees that term in the clear. This is a much weaker property than secrecy in the computational model, which usually relies on proving that an adversary cannot distinguish between two versions of the protocol: for example, one using one secret and one using another, or one using an encrypted secret and one using random data.

Recently the interest in formulating and applying indistinguishability properties for Dolev-Yao models has been growing. There are a number of reasons for this. The first is that cryptography has advanced to the point at which it is not only possible to provide computational proofs of security for algorithms, but for the protocols that use those algorithms as well. If Dolev-Yao tools can be extended to prove indistinguishability, this increases the likelihood that both approaches can be used together in an effective way to ensure protocol security. The second is that there is a growing class of privacy-protection protocols for which simple secrecy is clearly inadequate. Such protocols protect low-entropy data such as votes, medical records, or network routes; even partial leakage of this information could be harmful. The third is the result of recent work on automatic generation of cryptographic algorithms. In this work, multiple possible algorithms are generated out of a library of components and then checked for security. This may involve the use of Dolev-Yao like tools to weed out insecure algorithms or even verify the security of correct ones, as in [Barthe et al., 2013].

When a Dolev-Yao tool is used to check for subtle properties such as indistinguishability, it is important that it offers as detailed a picture of the properties of the cryptographic operations as possible. This is done by including information about the algebraic properties of the operations, that is, the equations obeyed by the function symbols. For example, if a cryptographic system uses exclusive-or, one should be able to take into account the associative-commutative, identity, and self-cancellation properties of exclusive-or. Such algebraic properties have been studied extensively in the literature, although there are still some classes of properties that are not that well understood.

At this point, there are four main problems being explored in relation

to Dolev-Yao indistinguishability. The first is how best to formulate in the Dolev-Yao model a property such as indistinguishability that in its broadest sense is *not* a reachability property. The second is, how to incorporate equational theories in this reasoning. The third is, how to increase the range and complexity of the types of protocols we can reason about. The fourth is, when and how to ensure that Dolev-Yao indistinguishability implies computational indistinguishability. In this chapter we address the first two of these problems, although we note that the second is closely related to the fourth, and can be used to facilitate its solution. In the following, we provide a high level explanation of how we address these two problems.

Formulating Indistinguishability in Dolev-Yao. We propose an intuitive notion of indistinguishability related to the notion of *uniformity* used in ProVerif [Blanchet et al., 2008] in that it is defined, not in terms of equivalence between two protocols, but of equivalence between *roles* of two protocols. In this case roles from the two protocol versions $\mathcal{P}_1$ and $\mathcal{P}_2$ are paired together and executed in a synchronous fashion (called a *synchronous product* in our case). We then define our notion as the conjunction of two more basic properties, namely *Indistinguishable Messages* (IM) and *Indistinguishable Attacker Event Sequences* (IAES).

Intuitively, the IM property says that the attacker, when performing the same actions for $\mathcal{P}_1$ and for $\mathcal{P}_2$, can never reach two corresponding stages in such action sequences such that it can learn the *same* message from $\mathcal{P}_1$ at both stages, but *different* messages from $\mathcal{P}_2$ at those same stages, or viceversa. The IM property assumes the existence of traces in both protocols and simply compares the messages. Thus, IM is closer to static equivalence than trace equivalence, where you need to prove the existence of such traces. The IAES property says that the attacker *can perform the same interaction steps* with $\mathcal{P}_1$ and $\mathcal{P}_2$, which requires a bisimulation between the two protocols. It ensures the existence of traces with the same behavior in both protocols. Indeed, IM does not imply IAES, since the latter is tested after the former is proved.

In our approach, we use an unmodified Dolev-Yao intruder with no ability to evaluate predicates. This is motivated by our preference to avoid increasing the complexity of Maude-NPA's Dolev-Yao model unless absolutely necessary, and thus to express our security requirements

in the original Maude-NPA framework. In particular, the IM property implicitly includes a test on equality, but it is expressed as a property of the attack state, not as an intruder predicate.

We prove a result with respect to the semantics of the Maude-NPA protocol analysis tool (see Chapter 3), showing that the conjunction of IM and IAES can be formulated in terms of reachability properties in Maude-NPA.

Incorporating Equational Theories. Our approach extends naturally to any algebraic theory E that can be decomposed as $E = E_0 \cup B$, with the equations E_0 oriented as rewrite rules modulo B, and the decomposition (B, E_0) satisfying the *finite variant* (FV) property. In this case we prove theorems showing that the IM and IAES properties can be checked by the Maude-NPA tool. The class of theories with finite variant decompositions contains a large number of theories of interest to cryptographic protocol analysis, including exclusive-or, Abelian groups, and a number of theories describing the properties of modular exponentiation. Thus our previous work on analysis of protocols modulo finite variant decompositions is *naturally* extended to the verification of indistinguishability under many possible equational theories.

Finally, let us illustrate the IM and IAES properties mentioned above with two examples.

Example 7.1 Consider two protocols $\mathcal{P}_1$ and $\mathcal{P}_2$ using the exclusive-or (XOR) operator "$\oplus$". Below we reproduce the exchange of messages of each protocol in Alice-Bob notation:

$$(\mathcal{P}_1)\ A \to B : m_1 \oplus m_1 \qquad\qquad (\mathcal{P}_2)\ A \to B : m_1 \oplus m_2$$

where m_1 and m_2 are two constants denoting different messages. Since the attacker can perform the XOR cancellation (i.e. $M \oplus M = 0$), and can generate the XOR unit element 0, then it can distinguish $\mathcal{P}_1$ and $\mathcal{P}_2$ by performing the following actions:

$$(\mathcal{P}_1)\ 1.\ A \to I(B) : m_1 \oplus m_1\ (=_{XOR}\ 0) \qquad (\mathcal{P}_2)\ 1.\ A \to I(B) : m_1 \oplus m_2$$

Thus these protocols do not satisfy the IM property, since in $\mathcal{P}_1$ the attacker generates 0 from two different action sequences, whereas in $\mathcal{P}_2$ it does not. ■

Example 7.2 Consider a protocol similar to the first step of the Encryption Key Exchange (EKE) protocol [Bellovin and Merritt, 1992] in which, unlike the original EKE, the attacker can distinguish whether a decryption succeeds or not. The algebraic properties of this protocol consist of the cancellation of encryption and decryption. In this protocol, Alice sends to Bob her name (A) concatenated with the encryption of her public key *pkey(A)* with a password *pw(A,B)* they have agreed on before.

$$A \to B : A\,;\, \{pkey(A)\}_{pw(A,B)}$$

Consider two cases in which the honest principals perform the same step shown above, but in $\mathcal{P}_1$ the attacker knows the right password *pw(A,B)*, whereas in $\mathcal{P}_2$ it knows a random password *pg(i)* [1]. The intruder can distinguish between $\mathcal{P}_1$ and $\mathcal{P}_2$ by performing the following actions, where steps 2, 3, and 4 denote deductions performed by the intruder:

($\mathcal{P}_1$) 1. $A \to I(B) : A\,;\, \{pkey(A)\}_{pw(A,B)}$	($\mathcal{P}_2$) 1. $A \to I(B) : A\,;\, \{pkey(A)\}_{pw(A,B)}$
($\mathcal{P}_1$) 2. $I : \{pkey(A)\}_{pw(A,B)}$	($\mathcal{P}_2$) 2. $I : \{pkey(A)\}_{pw(A,B)}$
($\mathcal{P}_1$) 3. $I : pw(A, B)$	($\mathcal{P}_2$) 3. $I : pg(i)$
($\mathcal{P}_1$) 4. I : *decryption succeeds*	($\mathcal{P}_2$) 4. I : *decryption fails*

That is, in step 4 in $\mathcal{P}_1$ the attacker can obtain the message *pkey(A)* in the clear, by decrypting the message sent in step 2 with the password it knows, i.e. *pw(A,B)*, whereas in $\mathcal{P}_2$ such decryption fails. Hence the protocols are not bisimilar and so fail to satisfy IAES. ■

7.2 Formal Definition of Indistinguishability in Maude-NPA

Intuitively, two protocols are indistinguishable if an intruder cannot tell the difference between them. In this section we provide the framework

[1] This is the approach followed by Lowe [2004]. However, there is also an *equivalent* version of the protocol specification, more in line with the convention used in cryptographic definitions of security, in which the attacker knows the same password *pw(A,B)* in both protocols and Alice uses a password *pw(A,B)* to encrypt her public key in $\mathcal{P}_1$ but a different password *pg(i)* in $\mathcal{P}_2$. Either of these can be expressed in our system.

that will allow the definition and verification of indistinguishability in Maude-NPA. In order to provide such framework, in Section 7.2.1 we first formalize, by the concept of a *protocol pairing*, the notion of pairs of protocols that are similar enough to each other so that the issue of their indistinguishability can arise. Then, in Section 7.2.2 we formalize the idea of *similar interactions* of the attacker with a protocol pairing $\mathcal{P}_1, \mathcal{P}_2$ by the concept of the *synchronous product* $\mathcal{P}_1 \otimes \mathcal{P}_2$. Finally, in Section 7.2.3 we define the indistinguishability of a protocol pairing in Maude-NPA as the conjunction of two simpler properties called IM and IAES.

7.2.1 Protocol Pairing

The notion of indistinguishability implies comparing two protocols $\mathcal{P}_1$ and $\mathcal{P}_2$ to ensure that an intruder *cannot distinguish* the behaviors of $\mathcal{P}_1$ and $\mathcal{P}_2$. In practical applications $\mathcal{P}_1$ and $\mathcal{P}_2$ are somewhat different *versions* of a given protocol with some significant *differences*. In this section, we formalize, by the concept of a *protocol pairing*, the notion of such pair of protocols in Maude-NPA.

Definition 7.3 (Protocol Pairing) *A* protocol pairing $\mathcal{P}_1, \mathcal{P}_2$ *is a pair of protocol specifications of the form* $((\Sigma_{\mathcal{P}_1}, E_{0_{\mathcal{P}_1}} \cup B_{\mathcal{P}_1}), HPS_{\mathcal{P}_1}, IS_{\mathcal{P}_1})$, $((\Sigma_{\mathcal{P}_2}, E_{0_{\mathcal{P}_2}} \cup B_{\mathcal{P}_2}), HPS_{\mathcal{P}_2}, IS_{\mathcal{P}_2})$, *where* $HPS_{\mathcal{P}_X}$ *and* $IS_{\mathcal{P}_X}$ *are the set of honest strands and the set of Dolev-Yao strands denoting the attacker's abilities of protocol* $\mathcal{P}_X$*, respectively, such that:*

1. $\mathcal{P}_1$ *and* $\mathcal{P}_2$ *share the same algebraic signature and equations, i.e.* $(\Sigma_{\mathcal{P}_1}, E_{0_{\mathcal{P}_1}} \cup B_{\mathcal{P}_1}) = (\Sigma_{\mathcal{P}_2}, E_{0_{\mathcal{P}_2}} \cup B_{\mathcal{P}_2}) = (\Sigma, E_0 \cup B)$ *having a decomposition* (Σ, B, E_0).

2. $HPS_{\mathcal{P}_1}$ *and* $HPS_{\mathcal{P}_2}$ *have strands for the same roles, with the same length and the same polarities* ($+$ *or* $-$) *at each position in the strand.*

3. $IS_{\mathcal{P}_1}$ *and* $IS_{\mathcal{P}_2}$ *have strands for the same operations, with the same length and the same polarities* ($+$ *or* $-$) *at each position in the strand.*

We assume both $HPS_{\mathcal{P}_1}$ and $HPS_{\mathcal{P}_2}$, and $IS_{\mathcal{P}_1}$ and $IS_{\mathcal{P}_2}$ have disjoint variables.

We will also require that strands in certain pairs both be identical up to change of variables, depending on the indistinguishability model used. For example, in the standard model based on cryptographic definitions of indistinguishability the same attacker interacts with two different versions of the protocol, so any two paired intruder strands must be identical up to change of variables. In Lowe's password guessing model the same attacker with different password guesses interacts with the same version of the protocol, so we require that any two paired strands be identical up to change of variables, except that describing the attacker's guess of the password.

The differences between $\mathcal{P}_1$ and $\mathcal{P}_2$ can be specified by having different messages in the same j-th strands of $\mathcal{P}_1$ and $\mathcal{P}_2$, at the same positions in the strands. Below we illustrate the notion of protocol pairing using the following example.

Example 7.4 Let us consider the two protocols $\mathcal{P}_1$ and $\mathcal{P}_2$ of Example 7.1 shown above. Note that both $\mathcal{P}_1$ and $\mathcal{P}_2$ share the same algebraic signature and equations (XOR), and have the same set of intruder strands, and differ only in the set of honest principal strands, as explained below. More specifically, the sets $HPS_{\mathcal{P}_1}$ and $HPS_{\mathcal{P}_2}$ of honest strands of $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively, are as follows:

$$HPS_{\mathcal{P}_1} = \{[\ (m_1 \oplus m_1)^+\ ]\} \qquad HPS_{\mathcal{P}_2} = \{[\ (m_1 \oplus m_2)^+\ ]\}$$

where m_1 and m_2 are two constants denoting different messages, and $\oplus$ is the exclusive-or operator. Therefore, $\mathcal{P}_1, \mathcal{P}_2$ is a protocol pairing. ■

7.2.2 Synchronous Product of Protocols

Given a protocol pairing $\mathcal{P}_1, \mathcal{P}_2$ as explained above, the analysis of its indistinguishability assumes that the attacker interacts in an analogous way with both protocols at each step. That is, if it performs an action a in $\mathcal{P}_1$, then it does so in $\mathcal{P}_2$ too. In this section we formalize the idea of *similar interactions* of the attacker with a protocol pairing $\mathcal{P}_1, \mathcal{P}_2$ by the concept of the *synchronous product* $\mathcal{P}_1 \otimes \mathcal{P}_2$. Intuitively, a synchronous

product is a new protocol obtained from a protocol pairing in which both protocols from the pairing are executed in a synchronous manner.

In order to provide a formal definition of a synchronous product of protocols, we first define the synchronous product of strands.

Definition 7.5 (Synchronous Product of Strands) *Given two strands* $Str_1 = [{m_1}^{\pm}, \ldots, {m_n}^{\pm}]$, *and* $Str_2 = [{m'_1}^{\pm}, \ldots, {m'_n}^{\pm}]$ *corresponding to the same protocol role or intruder action, and with the same polarities at each position in the strand, the synchronous product of* Str_1 *and* Str_2, *written* $Str_1 \otimes Str_2$, *is a strand of the form* $[(m_1 \otimes m'_1)^{\pm}, \ldots, (m_n \otimes m'_n)^{\pm}]$.

Let SS_1 *and* SS_2 *be two sets of strands that have* n *strands corresponding to the same protocol roles or intruder actions. The synchronous product of* SS_1 *and* SS_2, *written* $SS_1 \otimes SS_2$, *is a set of strands of the form* $\{Str_{1_i} \otimes Str_{2_j}\}_{0 \leqslant i,j \leqslant n}$, *such that* $Str_{1_i} \in SS_1$, $Str_{2_j} \in SS_2$, *and* Str_{1_i} *and* Str_{2_j} *correspond to the same protocol role or intruder action.*

Let us now define the synchronous product of protocols as follows.

Definition 7.6 (Synchronous Product of Protocols) *Given a protocol pairing* $\mathcal{P}_1, \mathcal{P}_2$, *its* synchronous product, *denoted by* $\mathcal{P}_1 \otimes \mathcal{P}_2$, *is a new protocol as explained below. Basically, the signature is extended with new sorts and symbols to support the specification of a pair of protocols.*

- *First, the theory decomposition* (Σ, B, E_0) *shared by* $\mathcal{P}_1$ *and* $\mathcal{P}_2$ *is renamed to* $(\hat{\Sigma}, \hat{B}, \widehat{E_0})$, *just by a renaming* $\mathsf{s} \mapsto \hat{\mathsf{s}}$ *(where* $\mathsf{s} \in \mathsf{S}$ *and* $\hat{\mathsf{s}} \in \hat{\mathsf{S}}$*), of the poset of sorts so that:* $\widehat{\mathsf{Msg}} = \mathsf{SingleMsg}$, *and* $\hat{\mathsf{s}} = \mathsf{s}$ *otherwise, and with* $\mathsf{s} < \mathsf{s}'$ *iff* $\hat{\mathsf{s}} < \hat{\mathsf{s}'}$. *The operators* $\hat{\Sigma}$ *are renamed accordingly, so that* $f : \mathsf{s_1} \cdots \mathsf{s_n} \to \mathsf{s}$ *is renamed to* $f : \hat{\mathsf{s}}_1 \cdots \hat{\mathsf{s}}_n \to \hat{\mathsf{s}}$, *and the equations* B *and* E_0 *are renamed to* $\hat{B}$ *and* $\widehat{E_0}$ *just by renaming the sorts of their variables by the mapping* $\mathsf{s} \mapsto \hat{\mathsf{s}}$.

- *A new sort* Msg *is added as the new top sort of the connected component for messages, so that* SingleMsg $<$ Msg.

- *A new operator* $\_ \otimes \_ : \mathsf{SingleMsg}\ \mathsf{SingleMsg} \to \mathsf{Msg}$ *is added to* $\Sigma_{\otimes}$.

- *Its protocol specification is the triple* $\mathcal{P}_1 \otimes \mathcal{P}_2 = ((\Sigma_{\otimes}, \widehat{E_0} \cup \hat{B}), HPS_{\mathcal{P}_1} \otimes HPS_{\mathcal{P}_2}, IS_{\mathcal{P}_1} \otimes IS_{\mathcal{P}_2})$, *where* $HPS_{\mathcal{P}_1}$, $HPS_{\mathcal{P}_2}$, $IS_{\mathcal{P}_1}$, *and* $IS_{\mathcal{P}_2}$ *are renamed to have disjoint variables.*

Therefore, if (Σ, B, E_0) *is the original theory decomposition, then the theory of the synchronous product is* $(\Sigma_{\otimes}, \widehat{B}, \widehat{E_0})$.

Example 7.7 For example, given the protocol pairing $\mathcal{P}_1, \mathcal{P}_2$ of Example 7.4, the synchronous product $\mathcal{P}_1 \otimes \mathcal{P}_2$ is specified as follows. In this protocol we only need sorts to denote messages. Therefore, the poset of sorts S is defined as $\mathsf{S} = \{\mathsf{Msg}, \mathsf{SingleMsg}\}$, such that $\mathsf{SingleMsg} < \mathsf{Msg}$.

The set of operators $\Sigma_{\otimes}$ contains the following operators:

$$\_ \otimes \_ : \mathsf{SingleMsg}\ \mathsf{SingleMsg} \rightarrow \mathsf{Msg}$$
$$\_ \oplus \_ : \mathsf{SingleMsg}\ \mathsf{SingleMsg} \rightarrow \mathsf{SingleMsg}\ \texttt{[assoc comm]}$$

where `[assoc comm]` denotes that the operator $\_ \oplus \_$ is associative and commutative. In this protocol we also define the following constants:

$$m_1 : \rightarrow \mathsf{SingleMsg}$$
$$m_2 : \rightarrow \mathsf{SingleMsg}$$
$$0 : \rightarrow \mathsf{SingleMsg}$$

where 0 denotes the XOR unit element. The equational theory of the synchronous product is specified as follows. First, the set of axioms $\widehat{B}$ consists of the associativity and commutativy of the $\oplus$ operator, whereas the set $\widehat{E_0}$ is a set of rewrite rules for the properties denoted by the equations below, which correspond to the identity and nilpotence properties of the $\oplus$ operator[2]:

$$M \oplus M = 0$$
$$M \oplus 0 = M$$
$$M \oplus M \oplus M' = M'$$

[2]Note that the first two equations are not *AC*-coherent, but adding the third equation is sufficient to recover that property (see Section 2.1).

where M is a variable of sort SingleMsg.

The set $HSP_{\mathcal{P}_1 \otimes \mathcal{P}_2}$ of protocol strands of the synchronous product $\mathcal{P}_1 \otimes \mathcal{P}_2$ is as follows:

$$HSP_{\mathcal{P}_1 \otimes \mathcal{P}_2} = \{ [\ ((m_1 \oplus m_1) \otimes (m_1 \oplus m_2))^+\ ] \}$$

The set $IS_{\mathcal{P}_1 \otimes \mathcal{P}_2}$ of intruder strands is as shown below:

$$\begin{aligned} IS_{\mathcal{P}_1 \otimes \mathcal{P}_2} = \{ & [\ (0 \otimes 0)^+\ ]\ \& \\ & [\ (M_1 \otimes M_1')^-, (M_2 \otimes M_2')^-, ((M_1 \oplus M_2) \otimes (M_1' \oplus M_2'))^+] \} \end{aligned}$$

denoting the intruder's capability to generate the XOR unit element 0, and to perform the XOR of two received messages, respectively. ■

The indistinguishability of a protocol pairing $\mathcal{P}_1, \mathcal{P}_2$ in Maude-NPA is characterized in terms of the synchronous product $\mathcal{P}_1 \otimes \mathcal{P}_2$, as we explain in Section 7.2.3 which is analyzed in Maude-NPA. Therefore, the equational theory of the synchronous product $\mathcal{P}_1 \otimes \mathcal{P}_2$ must have a finite variant decomposition. The following result states that, if the rewrite theory used by two protocols $\mathcal{P}_1$ and $\mathcal{P}_2$ has a finite variant decomposition, then so does the rewrite theory of $\mathcal{P}_1 \otimes \mathcal{P}_2$.

Theorem 7.8 *For a synchronous product $\mathcal{P}_1 \otimes \mathcal{P}_2$, the rewrite theory $(\Sigma_\otimes, \widehat{B}, \widehat{E_0})$ has the finite variant property iff the rewrite theory (Σ, B, E_0) does also.*

Proof. Since $(\widehat{\Sigma}, \widehat{B}, \widehat{E_0})$ is just a sort-renamed copy of (Σ, B, E_0) it has the FV property. Note that $(\widehat{\Sigma}, \widehat{B}, \widehat{E_0}) \subseteq (\Sigma_\otimes, \widehat{B}, \widehat{E_0})$ is a theory inclusion, and $\Sigma_\otimes$ terms are either $\widehat{\Sigma}$-terms, which have all a finite set of variants, or terms in $\mathcal{T}_{\Sigma_\otimes}(\mathcal{X}) - \mathcal{T}_{\widehat{\Sigma}}(\mathcal{X})$, which are either variables, whose only variant is itself, or terms of the form $t \otimes t'$, with $t, t' \in \mathcal{T}_{\widehat{\Sigma}}(\mathcal{X})$. Let $bd(t)$ and $bd(t')$ be the bounds (see [Escobar et al., 2012b]) on reduction sequences to their normal forms for t and t', respectively. Then $bd(t \otimes t') \leqslant bd(t) + bd(t')$, and thus $(\Sigma_\otimes, \widehat{B}, \widehat{E})$ has the FV property. □

Following Definition 5.4 in Page 122 the forward analysis of a synchronous product $\mathcal{P}_1 \otimes \mathcal{P}_2$ induces a transition system $\mathcal{L}_{\mathcal{P}_1 \otimes \mathcal{P}_2}$.

In the following, we define two projection functions that will be used below to connect the behavior of a synchronous product of protocols with the behavior of both protocols separately.

Definition 7.9 (Projection Functions) *Let Σ and $\Sigma_\otimes$ be as in Definition 7.6, and let $\hat{\mathcal{X}} = \{\mathcal{X}_{\hat{\mathsf{s}}}\}_{\hat{\mathsf{s}}\in\hat{\mathsf{S}}}$, and $\mathcal{X} = \{\mathcal{X}_{\mathsf{s}}\}_{\mathsf{s}\in\mathsf{S}}$. We then define functions $\pi_1, \pi_2 : \mathcal{T}_{\Sigma_\otimes}(\hat{\mathcal{X}}) \to \mathcal{T}_\Sigma(\mathcal{X})$ recursively as follows:*

$$\pi_1(x{:}\hat{\mathsf{s}}) = \pi_2(x{:}\hat{\mathsf{s}}) = x{:}\mathsf{s}$$
$$\pi_1(t_1 \otimes t_2) = t_1 \qquad \pi_2(t_1 \otimes t_2) = t_2$$
$$\pi_1(f(t_1,\ldots,t_n)) = f(\pi_1(t_1),\ldots,\pi_1(t_n)) \text{ for } f \neq \otimes$$
$$\pi_2(f(t_1,\ldots,t_n)) = f(\pi_2(t_1),\ldots,\pi_2(t_n)) \text{ for } f \neq \otimes$$

These projection functions are homomorphically extended to states, transitions and transition systems.

Proposition 7.10 *Given two protocols $\mathcal{P}_1$ and $\mathcal{P}_2$, and its synchronous product $\mathcal{P}_1 \otimes \mathcal{P}_2$ and their associated transition systems $\mathcal{L}_{\mathcal{P}_1}$, $\mathcal{L}_{\mathcal{P}_2}$, and $\mathcal{L}_{\mathcal{P}_1\otimes\mathcal{P}_2}$ as defined above, and the projection functions explained in Definition 7.9, $\pi_1 : \mathcal{L}_{\mathcal{P}_1\otimes\mathcal{P}_2} \to \mathcal{L}_{\mathcal{P}_1}$, and $\pi_2 : \mathcal{L}_{\mathcal{P}_1\otimes\mathcal{P}_2} \to \mathcal{L}_{\mathcal{P}_2}$ are both simulations.*

Proof. Easy from analysis of the rules $R_{F\mathcal{P}_1}$, $R_{F\mathcal{P}_2}$, and $R_{F(\mathcal{P}_1\otimes\mathcal{P}_2)}$. □

7.2.3 Indistinguishability in Maude-NPA

After explaining protocol pairing, synchronous product of protocols, and the projection functions, we define IM and IAES as follows.

Definition 7.11 (Indistinguishable Messages (IM)) *A protocol pairing $\mathcal{P}_1, \mathcal{P}_2$ with underlying equational theory $(\Sigma, E_0 \cup B)$ satisfies the* indistinguishable messages (IM) property *iff for any initial state St_0 of $\mathcal{L}_{\mathcal{P}_1\otimes\mathcal{P}_2}$, there exists no sequence of transitions $St_0 \to St_1 \cdots St_{n-1} \to St_n$ such that the intruder knowledge in St_n contains two facts $(m_1 \otimes m_2)\in\mathcal{I}$ and $(m_1' \otimes m_2')\in\mathcal{I}$ such that either (i) $m_1 =_E m_1'$ but $m_2 \neq_E m_2'$, or (ii) $m_2 =_E m_2'$ but $m_1 \neq_E m_1'$, where $E = E_0 \cup B$.*

Example 7.12 The protocol pairing $\mathcal{P}_1, \mathcal{P}_2$ of Example 7.1 in Page 196 does not satisfy the IM property. More specifically, there is an initial state of the form

$$\{\ [\ (0 \otimes (m_1 \oplus m_2))^+ \mid nil\ ]\ \&\ [\ (0 \otimes 0)^+ \mid nil\ ]\ \&$$
$$\{\ (0 \otimes (m_1 \oplus m_2)) \in \mathcal{I},\ (0 \otimes 0) \in \mathcal{I}\ \}\ \}$$

such that $0 \otimes (m_1 \oplus m_2) \neq_E 0 \otimes 0$. ■

Definition 7.13 Indistinguishable Attack Event Sequences (IAES) *Given $\mathcal{P}_1, \mathcal{P}_2$ a protocol pairing and two mappings $\pi_1 : \mathcal{L}_{\mathcal{P}_1 \otimes \mathcal{P}_2} \rightarrow \mathcal{L}_{\mathcal{P}_1}$ and $\pi_2 : \mathcal{L}_{\mathcal{P}_1 \otimes \mathcal{P}_2} \rightarrow \mathcal{L}_{\mathcal{P}_2}$, we say that $\mathcal{P}_1, \mathcal{P}_2$ have* indistinguishable attack event sequences (IAES) *iff π_1 and π_2 are bisimulations.*

Corollary 7.14 *Given $\mathcal{P}_1, \mathcal{P}_2$ a protocol pairing and two mappings $\pi_1 : \mathcal{L}_{\mathcal{P}_1 \otimes \mathcal{P}_2} \rightarrow \mathcal{L}_{\mathcal{P}_1}$ and $\pi_2 : \mathcal{L}_{\mathcal{P}_1 \otimes \mathcal{P}_2} \rightarrow \mathcal{L}_{\mathcal{P}_2}$, if π_1 and π_2 are bisimulations then protocols $\mathcal{P}_1$ and $\mathcal{P}_2$ are bisimilar.*

Example 7.15 The protocol pairing $\mathcal{P}_1, \mathcal{P}_2$ of Example 7.2 in Page 197 does not satisfy the IAES property. More specifically, π_1 and π_2 are not bisimulations, because in $\mathcal{P}_1$ the decryption performed in step 4 suceeds, whereas it fails in $\mathcal{P}_2$. ■

In Section 7.3 we explain in detail how the IAES and IM properties explained above are analyzed in practice in Maude-NPA.

7.3 Indistinguishability Verification in Maude-NPA

In this section we explain how the theoretical framework for indistinguishability verification presented in Section 7.2 can be implemented in Maude-NPA. In Section 5.3 we presented the rewriting-based forwards operational semantics of Maude-NPA, which is used as a reference model to define our notion of indistinguishability in Maude-NPA as the conjunction of two more basic properties, namely, *Indistinguishable Messages* (IM) and *Indistinguishable Attacker Event Sequences* (IAES), as

explained in Section 7.2. However, Maude-NPA performs a *backward narrowing-based reachability analysis* (see Chapter 3 for further details). In this section we prove a result with respect to the symbolic backwards semantics of the Maude-NPA protocol analysis tool presented in Section 3.4, showing that the conjunction of IM and IAES of a protocol pairing $\mathcal{P}_1, \mathcal{P}_2$ can be expressed as the unreachability of several different attack patterns for the synchronous product $\mathcal{P}_1 \otimes \mathcal{P}_2$.

In the following, for each property we describe a set of attack patterns in Maude-NPA's syntax denoting states in which such property is violated. The idea is that Maude-NPA performs a symbolic backwards analysis of those attack patterns. Intuitively, if Maude-NPA proves that at least one of such attack patterns is reachable from an initial state, then it proves that the protocol pairing violates the property and, therefore, the intruder can distinguish between both protocol variants. Instead, if the tool proves that no attack pattern is reachable (by generating a finite search space finding no initial state), then it proves that the protocol pairing satisfies the property. If a protocol pairing satisfies both IM and IAES, then the intruder *cannot* distinguish between both protocol variants.

The following theorem shows how to prove in Maude-NPA whether a given protocol pairing satisfies the IM property.

Theorem 7.16 *Let $\mathcal{P}_1, \mathcal{P}_2$ be a protocol pairing. Then $\mathcal{P}_1$ and $\mathcal{P}_2$ satisfy the IM property iff no initial state can be symbolically backwards reached from an attack state of $\mathcal{P}_1 \otimes \mathcal{P}_2$ of either one of the following forms:*

(1) $\{SS \;\&\; \{(m_1 \otimes m_2) \in \mathcal{I},\ (m_1 \otimes m_2') \in \mathcal{I}, (m_2 \neq_E m_2'), IK\}\}$, *or*

(2) $\{SS \;\&\; \{(m_1 \otimes m_2) \in \mathcal{I},\ (m_1' \otimes m_2) \in \mathcal{I}, (m_1 \neq_E m_1'), IK\}\}$

Proof. Immediate consequence of the soundness and completeness of the backwards operational semantics w.r.t. the forwards operational semantics (see Theorems 5.10 and 5.12 in Page 132) □

Theorem 7.17 *Let $\mathcal{P}_1, \mathcal{P}_2$ be a protocol pairing satisfying the IM property. Then $\mathcal{P}_1, \mathcal{P}_2$ satisfy the IAES property iff no initial state can be symbolically backwards reached from any attack state of either of the forms:*

(1) $\{\ SS\ \&\ [\ L\ \mid\ -(m_1 \otimes m_2),\ L'\ ]\ \&\ \{(m_1 \otimes m_2') \in \mathcal{I},\ (m_2 \neq_E m_2'),\ IK\}\}$, *or*

(2) $\{\ SS\ \&\ [\ L\ \mid\ -(m_1 \otimes m_2),\ L'\ ]\ \&\ \{(m_1' \otimes m_2) \in \mathcal{I},\ (m_1 \neq_E m_1'),\ IK\}\}$

Proof. ($\Rightarrow$) We reason by contradiction. Suppose $\mathcal{P}_1, \mathcal{P}_2$ satisfy IAES and an attack of type (1) exists. By the soundness and completeness of the backwards narrowing performed by Maude-NPA there is a ground substitution θ such that from an initial state of $\mathcal{P}_1 \otimes \mathcal{P}_2$ we can reach a ground state, via the forwards semantics, of the form $t = \{\widetilde{SS}\ \theta\ \&\ [L\theta]\ \&\ \{(m_1\theta \otimes m_2'\theta) \in \mathcal{I}, \widetilde{IK}\ \theta\}\}$, where $\widetilde{SS}$ are instances of *already executed* strand fragments in SS, and $\widetilde{IK}\ \theta$ are the instances of *positive* knowledge facts in IK, and with $m_2'\theta \neq_E m_2\theta$.

Because $(m_1\theta \otimes m_2\theta) \in \mathcal{I}$, $\mathcal{P}_1$ can make a transition

$$\pi_1(t) \rightarrow \{\pi_1(\widetilde{SS}\ \theta)\ \&\ [\pi_1(L\theta), -m_1\theta]\ \&\ \{m_1\theta \in \mathcal{I}, \pi_1(\widetilde{IK}\ \theta)\}\} \qquad (\dagger)$$

But, since π_1 is a bisimulation, this means that $\mathcal{P}_1 \otimes \mathcal{P}_2$ can make a transition

$$t \rightarrow \{SS\theta\ \&\ [L, -(m_1\theta \otimes m_2\theta)]\ \&\ \{(m_1\theta \otimes m_2\theta) \in \mathcal{I}, IK\}\} \qquad (\ddagger)$$

which is only possible if there is a fact $(m_1\theta \otimes m_2\theta) \in \mathcal{I}$ in IK, violating the IM assumption. The proof for an attack of type (2) is entirely similar.

($\Leftarrow$) Suppose no attacks of type (1) or (2) exist but, say $\pi_1 : \mathcal{P}_1 \otimes \mathcal{P}_2 \rightarrow \mathcal{P}_1$ (the case for π_2 is similar) is *not* a bisimulation. Thus there is a ground state t reachable from the initial state via the forwards semantics such that $\mathcal{P}_1$ can make a transition $\pi_1(t) \rightarrow u$ but there is no transition $t \rightarrow v$ in $\mathcal{P}_1 \otimes \mathcal{P}_2$ with $\pi_1(v) = u$. This can only happen for a *message receive* transition in a *user strand*. Therefore, the transition must be of the form ($\dagger$) above, but there is *no* transition of the form ($\ddagger$) above. Thus the received message $m_1\theta \in \mathcal{I}$ in $\pi_1(t)$ comes from a pair $(m_1\theta \otimes m_2'\theta) \in \mathcal{I}$ in t such that $m_2'\theta \neq_E m_2\theta$, contradicting the assumption that no attack of type (1) exists. $\square$

7.4 Experimental Evaluation

We have begun exploring the implementation of indistinguishability verification in Maude-NPA following the method presented in this chapter, and performed a preliminary evaluation. For example, we have proved in Maude-NPA the non-indistinguishability of some pairs of protocols such as the protocol involving an XOR operator of Example 7.1 and a complete version of the EKE protocol following the style of Example 7.2. We have analysed some other protocols but we were unable to achieve a finite state space due to state explosion. In the following, we describe in more detail the analyses of the pairs of protocols corresponding to Examples 7.1 and 7.2 in Maude-NPA.

The source files of the protocol specifications and the output of the analyses of these examples are available at:

`http://www.dsic.upv.es/~sescobar/Maude-NPA/indist.html`

Example 7.18 Let us first consider the pair of protocols $\mathcal{P}_1$ and $\mathcal{P}_2$ involving the XOR operator of Example 7.1 in Page 196, which violate the IM property. The analysis of $\mathcal{P}_1 \otimes \mathcal{P}_2$ in Maude-NPA proves that $\mathcal{P}_1$ and $\mathcal{P}_2$ violate the IM property. More specifically, when Maude-NPA searches backwards from the attack state shown below:

```
vars X Y Z : SingleMsg .
eq ATTACK-STATE(0)
 = empty
   || pair(X,Y) inI, pair(X,Z) inI, pair(X,Y) != pair(X,Z)
   || nil
   || nil
   || nil
[nonexec] .
```

it generates a finite search space finding an initial state after 2 backwards reachability steps. Below we show the output of this analysis:

```
==========================================
reduce in MAUDE-NPA : summary(0) .
rewrites: 54 in 0ms cpu (0ms real) (~ rewrites/second)
result Summary: States>> 1 Solutions>> 0
```

```
==========================================
reduce in MAUDE-NPA : summary(1) .
rewrites: 140387 in 588ms cpu (586ms real) (238738 rewrites/second)
result Summary: States>> 2 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(2) .
rewrites: 22778 in 128ms cpu (128ms real) (177942 rewrites/second)
result Summary: States>> 1 Solutions>> 1
```

The initial state found by Maude-NPA is as shown below:

```
< 1 . 1 . 2 > (
:: nil ::
[ nil | +(pair(0, 0)), nil]  &
:: nil ::
[ nil | +(pair(0, m1 * m2)), nil] )
||
pair(0, 0) !inI,
pair(0, m1 * m2) !inI
||
+(pair(0, m1 * m2)),
+(pair(0, 0))
||
nil
||
nil
```

where *0* is the XOR unit element. The initial state shown above corresponds to the exchange of messages explained in Example 7.1 in Page 196. ■

Let us now explain in more detail the specification and analysis of a complete version of the EKE protocol following the style of Example 7.2 in Page 197, which is an example of two protocols violating the IAES property.

Example 7.19 We consider a version of the Encryption Key Exchange (EKE) protocol [Bellovin and Merritt, 1992] in which, unlike the original version of the protocol, the attacker can distinguish whether a decryption succeeds or not. In the EKE protocol a party encrypts an ephemeral

(one-time) public key using a password, and sends it to a second party, who decrypts it and uses it to negotiate a shared key with the first party.

This protocol involves asymmetric encryption, using public and private keys, and two kinds of symmetric encryption using shared keys and passwords, respectively. A public key is represented as $pkey(A)$, whereas the private key corresponding to a public key $pkey(A)$, is represented as $inv(pkey(A))$. Asymmetric encryption of a message M with a public key is written as $penc(pkey(A), M)$, whereas asymmetric encryption with a private key is written as $pdec(inv(pkey(Ke)), M)$. Symmetric encryption/decryption with a shared key $skey(A, B)$ between two principals A and B, is written as $senc(skey(Ke), M)$ and $sdec(skey(A, B), M)$, respectively. Finally, symmetric encryption/decryption of M using a password $pw(A, B)$ agreed between two principals A and B, is denoted by $enc(pw(A, B), M)$ and $dec(pw(A, B), M)$, respectively.

Similarly to Example 7.2, we consider two protocols $\mathcal{P}_1$ and $\mathcal{P}_2$ in which the honest principals perform the same exchange of messages, and the difference is in the attacker's guess of the password, as explained below. The exchange of messages among the honest principals in both protocols is as follows:

1. $A \rightarrow B:\ A; enc(pw(A, B), pkey(A))$
2. $B \rightarrow A:\ enc(pw(A, B), penc(PKA, skey(A, B)))$
3. $A \rightarrow B:\ senc(skey(A, B), N_A)$
4. $B \rightarrow A:\ senc(skey(A, B), N_A; N_B)$
5. $A \rightarrow B:\ senc(skey(A, B), N_B)$

where A and B denote Alice and Bob's names, respectively, and N_A and N_B represent nonces generated by Alice and Bob, respectively.

The analysis of $\mathcal{P}_1 \otimes \mathcal{P}_2$ in Maude-NPA proves that $\mathcal{P}_1$ and $\mathcal{P}_2$ violate the IAES property. More specifically, when Maude-NPA searches backwards from the attack state shown below:

```
var Z : SingleMsg .
var PK : PKey .

eq ATTACK-STATE(0)
```

```
   = :: nil ::
     [ nil | -(pair(pw(a,b), Z)),
             -(pair(enc(pw(a,b),PK),enc(pw(a,b),PK))),
             +(pair(PK,PK)), nil ]
     || pair(pw(a,b), Z) inI, Z != pw(a,b)
     || nil
     || nil
     || nil
  [nonexec] .
```

it generates a finite search space finding an initial state after 1 backwards reachability step. Note that this attack pattern is more instantiated than the attack patterns of the form shown in Theorem 7.17 for IAES. However, it provides a faster analysis. Below we show the output of this analysis:

```
==========================================
reduce in MAUDE-NPA : summary(0) .
rewrites: 16689 in 72ms cpu (71ms real) (231778 rewrites/second)
result Summary: States>> 1 Solutions>> 0
==========================================
reduce in MAUDE-NPA : summary(1) .
rewrites: 7669886 in 7848ms cpu (7848ms real) (977243 rewrites/second)
result Summary: States>> 1 Solutions>> 1
```

The initial state found by Maude-NPA is as shown below:

```
< 1 . 1 > (
:: nil ::
[ nil |
   +(pair(pw(a, b),pg(i))), nil]  &
:: nil ::
[ nil |
   -(pair(pw(a, b), pg(i))),
   -(pair(enc(pw(a, b), #1:PKey), enc(pw(a, b), #1:PKey))),
   +(pair(#1:PKey, #1:PKey)), nil] )
||
pair(pw(a, b),pg(i)) !inI
||
+(pair(pw(a, b),pg(i)))
||
```

```
nil
||
nil
```

denoting that in $\mathcal{P}_1$ the intruder guessed the right password and, therefore, it can obtain the message `#1:PKey` in the clear, by decrypting message $\texttt{enc}(\texttt{pw}(\texttt{a},\texttt{b}), \#1 : \texttt{PKey})$, whereas in $\mathcal{P}_2$ it guessed a wrong password, and therefore, the decryption failed. ■

As explained before, we performed some other analyses, but we were unable to achieve a finite state space for those protocols. We have investigated reasons for non-termination due to state space explosion. One is that the attack states used are quite general. As future work, we plan to study sound and complete ways to replace them by more specific attack states, following the style of the attack state used in Example 7.19.

Another is that these analyses make a heavy use of inequalities modulo equational theories. Currently, in Maude-NPA inequality constraints are not checked until an initial state is reached. ProVerif, for example, includes methods for solving inequality constraints earlier (see [Blanchet et al., 2008]). Basically, this tool includes a predicate call "nounif" which allows the intruder to check if two terms do not unify. However, such inequatlity predicate is beyond unification, because it requires to compute all the possible negative cases of an equality predicate, that is, those evaluating to true, and those terms evaluating to false. Indeed, ProVerif's "nounif" predicate can only be applied to pairs of constructor symbols, and does not support AC operators. A similar procedure could be implemented in Maude-NPA but this implies computing all the variants of the inequality predicate. The problem is that inequality for AC symbols is not as easy to define as equality. Indeed, work on evaluating inequality constraints for theories with AC is beginning to appear (see e.g. [Gutiérrez et al., 2012]). We expect to explore this issue further in our future work.

7.5 Conclusions

We have formalized an intuitive notion of indistinguishability as the conjuntion of the IM and IAES properties. This is a bisimulation-type property. But it is significantly simplified by the fact that, since we model all

possible actions of the attacker by intruder strands and by the intruder knowledge, we do not need to consider *arbitrary contexts* —used, say, in the π-calculus to model the attacker— on which to place the protocols (which are modeled in our terms by the *honest principal strands*).

Our formalization of IM and IAES in terms of the synchronous product of two protocols in a pairing has been shown to be *checkable* automatically by Maude-NPA. This is a significant step-forward in indistinguishability research, because, for the first time, there is a tool that can perform such automatic checks *modulo* a very wide class of theories, namely, all theories with the finite variant property that can have axioms B such as AC or C, which include theories like Abelian Groups, several theories of homomorphic encryption, exclusive-or, and modular exponentiations essential for many privacy-preserving protocols. We have also illustrated with concrete examples how this kind of indistinguishability analysis can be performed by Maude-NPA.

Chapter 8

Asymmetric Unification: A New Unification Paradigm for Cryptographic Protocol Analysis

In this chapter we provide a tool-independent methodology for state exploration, called *contextual symbolic reachability analysis*. This methodology is based on unification and narrowing, and generates states that obey certain irreducibility constraints w.r.t. the protocol equational theory. Contextual symbolic reachability analysis also introduces a new type of unification mechanism, called *asymmetric unification*, in which any solution must leave the right side of the solution irreducible.

First, Section 8.1 motivates contextual symbolic reachability analysis, showing a problem that arises in cryptographic protocol analysis when the equational properties of the cryptosystem are taken into account: in many situations it is necessary to guarantee that certain terms generated during a state exploration are in *normal form* with respect to the equational theory. In Section 8.2 we introduce contextual symbolic reachability analysis and give a formal definition of asymmetric unification, illustrating their use in Maude-NPA. Section 8.3 outlines a general procedure for converting a symmetric algorithm to an asymmetric one, and applies it to exclusive-or with uninterpreted function symbols. In Section 8.4, we (i) show experiments illustrating the benefits, in Maude-NPA, of using

contextual symbolic reachability and asymmetric unification to integrate reachability analysis modulo exclusive-or with optimizations based on syntactic checks, and (ii) the experimental results on an implementation of the asymmetric XOR algorithm of Section 8.3 in Maude-NPA, comparing its performance with an asymmetric variant-based unification algorithm.

These results have been published in [Erbatur et al., 2012, 2013].

8.1 Motivation

In many cases, equational reasoning is integrated with syntactic reasoning. There are a number of reasons for doing this, but one reason is that optimizations that are done to eliminate redundant or nonsensical states may need to be done via syntactic checking, as in Maude-NPA. We illustrate the issues that can arise with the following protocol, which we will use as a running example. It uses an exclusive-or (XOR) operator $\oplus$, which is associative and commutative (AC) and self-canceling with identity 0, and a function pk, where $pk(A, X)$ stands for encryption of message X with A's (standing for Alice's) public key; below, B stands for Bob.

Example 8.1 Upon receiving the final message, Alice verifies that she received $X \oplus N_A$ for some X received in the first message $pk(A, X)$. The protocol is seen differently by Bob and Alice, as shown in the second and third columns.

Alice and Bob	Bob	Alice
1. $B \to A : pk(A, N_B)$	1. $B \to A : pk(A, N_B)$	1. $B \to A : pk(A, X)$
2. $A \to B : pk(B, N_A)$	2. $A \to B : pk(B, Z)$	2. $A \to B : pk(B, N_A)$
3. $B \to A : N_A \oplus N_B$	3. $B \to A : Z \oplus N_B$	3. $B \to A : N_A \oplus X$

■

We find an instance of the protocol from Alice's perspective by applying the substitution $X \mapsto N_A \oplus Y$ to achieve the left-hand column of Example 8.2. Maude-NPA could identify this instance as infeasible and discard it, since Alice cannot receive a message $N_A \oplus Y$ before she generates the nonce N_A.

Example 8.2 But further instantiating Y (perhaps as a result of further unifications elsewhere) to $N_A \oplus N_B$ causes problems.

Alice after $X \mapsto N_A \oplus Y$	Alice after $Y \mapsto N_A \oplus N_B$.
1. $B \to A : pk(A, N_A \oplus Y)$	1. $B \to A : pk(A, N_A \oplus N_A \oplus N_B) = pk(A, N_B)$
2. $A \to B : pk(B, N_A)$	2. $A \to B : pk(B, N_A)$
3. $B \to A : N_A \oplus N_A \oplus Y$	3. $B \to A : N_A \oplus N_A \oplus N_A \oplus N_B = N_A \oplus N_B$

■

This makes $N_A \oplus Y$ reduce to N_B and $N_A \oplus N_A \oplus Y$ reduce to $N_A \oplus N_B$, giving the right-hand side of Example 8.2: the intended legal execution of the protocol! Thus, Maude-NPA's syntactic check inadvertently could have ruled out a legal execution.

We avoid this problem as follows. We first decompose the XOR theory into (B, E_0), where B is the AC theory and E_0 is a set of rewrite rules for the properties $\{X \oplus 0 = X,\ X \oplus X = 0,\ X \oplus X \oplus Y = Y\}$. We then divide the possible instantiations of $\{pk(A, X), N_A \oplus X\}$ into two cases, each of which are constrained to remain irreducible under substitution. One is $\{pk(A, X), N_A \oplus X\}$, and the other is $\{pk(A, Y \oplus N_A), Y\}$ obtained by the substitution $X \mapsto Y \oplus N_A$. Every other reduced instantiation of $N_A \oplus X$ is an instance of either one or the other modulo AC. The case obtained by $X \mapsto Y \oplus N_A$ can now be safely deleted, because due to the irreducibility constraint that Y cannot contain N_A and 0, the N_A will never vanish from $N_A \oplus Y$ under any substitution.

This strategy works for several reasons. One is that Maude-NPA syntactic checks require that irreducibility constraints only be put on received messages. Another, and more important, is that the exclusive-or theory has the *finite variant property* [Comon-Lundh and Delaune, 2005] modulo AC. Thus, for every term s there is a finite set $s'_1, \ldots, s'_k$ of reduced instances of s such that any other reduced instance of s is equal modulo AC to a substitution instance of one of the s'_i. These two features mean that it is possible to integrate syntactic checks that are invariant under AC together with unification-based reachability modulo a richer theory, allowing us to improve efficiency without sacrificing soundness and completeness. Indeed, this is vital for Maude-NPA and other tools, because almost all of the checks used for optimization require the received messages to be in normal form.

Another capability that is needed for our strategy to work opens up a new area of research, namely, developing a sound and complete, tool-independent symbolic state exploration algorithm that preserves irreducibility constraints. In Maude-NPA state exploration is implemented via equational unification of sent messages with received messages, which means that the equational unification algorithm used should preserve the irreducibility of the received messages. Indeed, it was experimentation with a unification algorithm that did *not* have this property, the algorithm of [Liu and Lynch, 2011], that produced the example we described above. Variant narrowing unification (the algorithm currently used by Maude-NPA) has the properties that we need, but our search of the literature has produced no other examples. This has led us to define a class of unification algorithms known as *asymmetric unification algorithms* modulo a theory (B, E_0), which produce a most general set of unifiers which leave the right hand side irreducible. We have worked on techniques for converting standard equational unification algorithms into asymmetric algorithms, and have produced an asymmetric version of the exclusive-or algorithm in [Liu and Lynch, 2011].

We are not the only ones to use an approach that integrates syntactic and equational reasoning: this has also been done by other researchers for other reasons, as we have described in Section 1.2. However, most work in this area has concentrated on specific applications of this approach, and not on how to implement the approach itself. This chapter is devoted to providing a general procedure for doing this, called *contextual symbolic reachability analysis* modulo a theory (B, E_0), where E_0 is a set of rewrite rules. This employs a technique called *contextual unification* in which some subterms of the two terms being unified are constrained to be irreducible. In Maude-NPA these are input terms, which, since they are unified with output terms, create the opportunity for exploiting asymmetric unification. However, this is not the only way contextual symbolic reachability analysis could be implemented. For example, we could follow the approach of OFMC [Basin et al., 2005; Mödersheim and Viganò, 2009] which requires that both input and output terms are irreducible. Thus, our tool-independent framework should have many applications beyond Maude-NPA, allowing for experimentation with different techniques.

8.2 Contextual Symbolic Reachability Analysis

As we have explained in Section 8.1, the symbolic reachability approach presented in Section 2.2 does not really work in practice, since the particular way that a representative is chosen for each equivalence class may be crucial for the correct behavior, and in many cases the termination of a tool crucially depends on state space reduction techniques based on checking such representatives, as we illustrated for the case of nonces that *cannot* have been generated yet at a given point. Therefore, we now present a general, tool-independent framework for symbolic reachability analysis which refines narrowing modulo equations by imposing *irreducibility conditions* on representatives of equivalence classes. First, we give a way of imposing these irreducibility conditions on a rewrite theory, expressed by the notion of *contextual rewrite theory.*[1]

Definition 8.3 (Contextual Rewrite Theory) *A* contextual rewrite theory *is a tuple* $(\Sigma, B, E_0, T, \phi)$ *where:*

- $(\Sigma, E_0 \cup B, T)$ *is an order-sorted topmost rewrite theory,*
- (Σ, B, E_0) *is a decomposition of the equational theory* $(\Sigma, E_0 \cup B)$*, and*
- ϕ*, called the* irreducibility conditions*, is a function mapping each* $f \in \Sigma$ *to a set of its arguments, i.e.,* $\phi(f) \subseteq \{1, \ldots, ar(f)\}$*, where* $ar(f)$ *is the number of arguments of* f*. The set of* maximal irreducible positions *of a term* t *is denoted by* $\phi(t)$*.*

A term t *is called* ϕ, E_0, B-irreducible *(or just* ϕ*-irreducible) if for each* $p \in \phi(t)$*,* $t|_p{\downarrow_{E_0,B}} =_B t|_p$*, and* strongly ϕ-irreducible *if for any* E_0, B*-normalized substitution* σ*,* $t\sigma$ *is* ϕ*-irreducible.*

[1] Our use of "contextual" should be distinguished from : (i) "contextual rewriting," e.g., [Zhang and Remy, 1985], and (ii) "context-sensitive rewriting," e.g., [Lucas, 1998]. Our use is unrelated to contextual rewriting, which is a form of conditional rewriting with constraints, but is closely related to context-sensitive rewriting, where the rewritable argument positions of a function symbol f are specified by a function $\mu(f) \subseteq \{1, \ldots, ar(f)\}$ similar to our irreducibility conditions function $\phi(f) \subseteq \{1, \ldots, ar(f)\}$. However, ϕ-irreducibility is a *strictly stronger* requirement than μ-irreducibility when $\phi = \mu$.

Example 8.4 For the protocol of Example 8.1 the contextual rewrite theory $(\Sigma, B, E_0, T, \phi)$ is formed of T containing the reversed version of the generic rewrite rules (3.1)–(3.3) plus the rewrite rules for introducing new strands (rules (3.4)), i.e. $T = R_{B\mathcal{P}}^{-1}$, and the equational theory $(\Sigma, E_0 \cup B)$ for exclusive-or is decomposed into (Σ, B, E_0) where B and E_0 are as described in Example 2.3 in Page 28. The irreducibility conditions ϕ are imposed on two operators: $-(\_)$ for *input messages* in a strand, and $\_\in\mathcal{I}$ for each *positive fact* in the intruder knowledge. That is, $\phi(-(\_)) = \{1\}$, $\phi(\_\in\mathcal{I}) = \{1\}$, and $\phi(f) = \varnothing$ otherwise. ■

We extend the notion of a reachability goal to the contextual case.

Definition 8.5 (Contextual Reachability goal) *Given a contextual rewrite theory $(\Sigma, B, E_0, T, \phi)$, we define a contextual reachability goal as $t \overset{?}{\rightarrow}^*_{T,E_0,B,\phi} t'$, where $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})_\mathsf{s}$. We write $t \overset{?}{\rightarrow}^*_{\phi} t'$ when the theory is clear. A substitution σ is a* solution *of the contextual reachability goal $t \overset{?}{\rightarrow}^*_{T,E_0,B,\phi} t'$ iff there is a sequence $t\sigma \rightarrow_{T,(E_0\cup B)} u_1\sigma \rightarrow_{T,(E_0\cup B)} \cdots \rightarrow_{T,(E_0\cup B)} (u_{k-1})\sigma \rightarrow_{T,(E_0\cup B)} t'\sigma$ such that $t\sigma, u_1\sigma, \ldots, (u_{k-1})\sigma, t'\sigma$ are all ϕ, E_0, B-irreducible.*

As for reachability goals, a contextual version of narrowing provides a mechanism to find solutions to contextual reachability goals. However, we have to first define a new equational unification mechanism, called *contextual unification*, as the basis for contextual narrowing, where the $E_0 \cup B$-unification is extended to the contextual case, which has some asymmetry due to the irreducibility restrictions only on the right hand side.

Definition 8.6 (Contextual Unification) *Given a contextual rewrite theory $(\Sigma, B, E_0, T, \phi)$, a substitution σ is a* contextual E_0, B-unifier *of a set P of* contextual equations *of the form $P = \{t_1 =_{\downarrow_\phi} t'_1, \ldots, t_n =_{\downarrow_\phi} t'_n\}$ iff for every contextual equation $t_i =_{\downarrow_\phi} t'_i$ in P, the substitution σ is an $(E_0 \cup B)$-unifier of the equation $t_i = t'_i$ and, furthermore, $t'_i\sigma$ is ϕ,E_0,B-irreducible.*

A set of substitutions Ω is a complete set of contextual E_0, B-unifiers *of P, denoted by $CSU_{E_0,B,\phi}(P)$, iff: (i) every member of Ω is a contextual E_0, B-unifier of P, and (ii) for every contextual E_0, B-unifier θ of P there exists $\sigma \in \Omega$ such that $\sigma \sqsupseteq_B \theta$.*

Example 8.7 Consider the protocol of Example 8.1, and the state pattern shown below, with Alice and Bob's strands with the vertical bar at the end, and an empty intruder knowledge.

$$\{ :: r_2 :: [nil, -(pk(A,X)), +(pk(B,n(A,r_2))), -(X \oplus n(A,r_2)) \mid nil] \;\&\\ :: r_1 :: [nil, +(pk(A,n(B,r_1))), -(pk(B,Y)), +(Y \oplus n(B,r_1)) \mid nil] \;\&\\ SS \;\&\; \{IK\}\}$$

The contextual unification problem found by Maude-NPA is $t =_{\downarrow_\phi} t'$ where t is $\{SS \;\&\; [L, M^+ \mid L'] \;\&\; \{M{\in}\mathcal{I}, IK\}\}$ i.e., the right-hand side of Rule (3.3), and t' is the following state, found by Maude-NPA after one backwards narrowing step from the state pattern shown above.

$$\{ :: r_2 :: [nil, -(pk(A,X)), +(pk(B,n(A,r_2))) \mid -(X \oplus n(A,r_2)), nil] \;\&\\ :: r_1 :: [nil, +(pk(A,n(B,r_1))), -(pk(B,Y)), +(Y \oplus n(B,r_1)) \mid nil] \;\&\\ SS \;\&\; \{(X \oplus n(A,r_2)){\in}\mathcal{I}, IK\}\}$$

The two key terms are $Y \oplus n(B,r_1)$ and $X \oplus n(A,r_2)$. Note that term $X \oplus n(A,r_2)$ appears in two positions in t', under symbols $-(\_)$ and $\_{\in}\mathcal{I}$, both required to be irreducible by ϕ. The singleton most general contextual unifier is $\sigma_1 = \{Y \mapsto X \oplus n(B,r_1) \oplus n(A,r_2)\}$, whereas the substitution $\sigma_2 = \{X \mapsto Y \oplus n(B,r_1) \oplus n(A,r_2)\}$ is *not* a valid contextual unifier: term $X \oplus n(A,r_2)$ is under the irreducibility condition of symbol $-(\_)$ and the substitution σ_2 would make it reducible, whereas term $Y \oplus n(B,r_1)$ is under symbol $+(\_)$, which does not have any irreducibility condition and the substitution σ_1 makes it reducible. ■

Contextual unification can be reduced to the simpler notion of *asymmetric unification.*

Definition 8.8 (Asymmetric Unification) *Given a decomposition* (Σ, B, E_0) *of an equational theory* $(\Sigma, E_0 \cup B)$*, a substitution* σ *is an* asymmetric E_0, B-unifier *of a set* P *of* asymmetric equations $\{t_1 =_\downarrow t'_1, \ldots, t_n =_\downarrow t'_n\}$ *iff for every asymmetric equation* $t_i =_\downarrow t'_i$ *in* P*,* σ *is an* $(E_0 \cup B)$*-unifier of the equation* $t_i = t'_i$ *and* $(t'_i{\downarrow_{E_0,B}})\sigma$ *is in* E_0,B*-normal form.*

A set of substitutions Ω *is a* complete set of asymmetric E_0,B-unifiers *of* P *iff: (i) every member of* Ω *is an asymmetric* E_0,B*-unifier of* P*, and*

(ii) for every asymmetric E_0,B-unifier θ of P there exists a $\sigma \in \Omega$ such that $\sigma \sqsupseteq_B \theta$ (over $\mathcal{V}ar(P)$).

In the following, we always assume that in every asymmetric equation $t =_{\downarrow} t'$, t' is in normal form; otherwise, we can always normalize t'.

Example 8.9 Consider the asymmetric unification problem $Y \oplus n(B, r') =_{\downarrow} X \oplus n(A, r)$ arising in Example 8.7. Then, there is a most general $\oplus$-unifier $X \mapsto Y \oplus n(B, r') \oplus n(A, r)$. However, this is not an asymmetric unifier; but an equivalent $\oplus$-unifier is $Y \mapsto X \oplus n(B, r') \oplus n(A, r)$, which is the singleton most general asymmetric unifier. ■

For any $(E_0 \cup B)$-unifier θ of P and substitution τ, $\theta\tau$ is also an $(E_0 \cup B)$-unifier of P. But this is not necessarily the case for asymmetric E_0,B-unifiers.

Example 8.10 Consider Example 8.9 and the most general exclusive-or asymmetric unifier $Y \mapsto X \oplus n(B, r') \oplus n(A, r)$. If we apply the substitution $X \mapsto n(A, r)$ to the above unifier, the resulting substitution is no longer an asymmetric unifier of the original asymmetric unification problem. ■

Indeed, nothing can be said about the asymmetric unifiers of a problem from its set of unifiers. The unification problem could have a nonempty set of unifiers, whereas the asymmetric unification problem need not have any asymmetric unifier. Or, the unification problem could have a single most general unifier, whereas the asymmetric unification problem has exponentially many solutions, as illustrated using the following asymmetric unification problem:

$$x_1 \oplus \ldots \oplus x_n =_{\downarrow} a_1 \oplus \ldots a_n, \quad x_1 \oplus \ldots \oplus x_n =_{\downarrow} x_1 \oplus \ldots \oplus x_n$$

which has a single unifier $x_1 \mapsto x_2 \oplus ... \oplus x_n \oplus a_1 \oplus ... \oplus a_n$, and $n!$ asymmetric unifiers. The complexity and decidability results of asymmetric unification given in [Erbatur et al., 2013, Theorems 5 and 6] state that there are theories for which symmetric unification is decidable and asymmetric unification is undecidable

The reduction of contextual unification to the simpler asymmetric unification is provided by the following lemma.

Lemma 8.11 *Given a contextual rewrite theory* $(\Sigma, B, E_0, T, \phi)$ *and a set of contextual equations* $P = \{t_1 =_{\downarrow_\phi} t'_1, \ldots, t_n =_{\downarrow_\phi} t'_n\}$, σ *is a contextual* E_0, B*-unifier of* P *iff there is a substitution* θ *such that* θ *is an asymmetric* E_0, B*-unifier of* $\Gamma(P)$ *and* $\sigma =_E \theta|_{\mathcal{V}ar(P)}$, *where*

$$\begin{aligned}\Gamma(P) =& \{t_i =_\downarrow X, t'_i =_\downarrow X \mid t_i =_{\downarrow_\phi} t'_i \in P, X \textit{ fresh variable}\} \cup \\ & \{t'_i|_{p.j} =_\downarrow t'_i|_{p.j} \mid t_i =_{\downarrow_\phi} t'_i \in P, f \in \Sigma, p \in \mathcal{P}os_f(t'_i), j \in \phi(f)\}\end{aligned}$$

Proof. Immediate.

Using a contextual unification algorithm, we can modify the standard notion of narrowing so that it uses contextual unification to solve symbolic contextual reachability goals. Note that the following definition differs from the definition of narrowing modulo an equational theory given in Section 2.2 only in using contextual unification $CSU_{E_0,B,\phi}(l =_{\downarrow_\phi} t|_p)$ instead of regular unification $CSU_{E_0 \cup B}(l = t|_p)$ and carrying a set of irreducible terms Π passed to the contextual unification algorithm, where Π is the set of irreducible terms that have been computed earlier in the narrowing sequence.

Definition 8.12 (Contextual Narrowing modulo E_0, B**)** *Given a contextual rewrite theory* $(\Sigma, B, E_0, T, \phi)$, *the contextual narrowing relation modulo* E_0, B *on pairs* $\langle t, \Pi \rangle$ *for* t *a term and* Π *a set of irreducible terms is defined as* $\langle t, \Pi \rangle \overset{\sigma}{\leadsto}_{T,E_0,B,\phi} \langle t', \Pi\sigma \rangle$ *(or* $\overset{\sigma}{\leadsto}_\phi$ *if* T, E_0, B *are understood) iff there is* $p \in Pos_\Sigma(t)$, *a rule* $l \to r$ *in* T *such that* $\mathcal{V}ar(t) \cap (\mathcal{V}ar(l) \cup \mathcal{V}ar(r)) = \varnothing$, *a substitution* $\sigma \in CSU^V_{E_0,B,\phi}(P)$ *for* $P = \{l =_{\downarrow_\phi} t|_p\} \cup \{u =_{\downarrow_\phi} u \mid u \in \Pi\}$ *and a set* V *of variables containing* $\mathcal{V}ar(t)$, $\mathcal{V}ar(l)$, *and* $\mathcal{V}ar(r)$, *and* $t' = (t[r]_p)\sigma$.

The essential equivalence between contextual reachability analysis and standard narrowing-based reachability analysis is proved as follows: given a standard goal $t \overset{?}{\to}^*_{T,E_0 \cup B} t'$, *any* solution to it can be computed by contextual narrowing $\leadsto_{T,E_0,B,\phi}$ under some extra conditions involving *variants*. Let us motivate the issues involved by an example.

Example 8.13 Let us consider the protocol of Example 8.1 and a state pattern with Alice and Bob's strands with the vertical bar at the end, and the requirement that the intruder learns $n(A, r_2)$:

$$\{ :: r_2 :: [nil, -(pk(A, X)), +(pk(B, n(A, r_2))), -(X \oplus n(A, r_2)) \mid nil] \;\&$$
$$:: r_1 :: [nil, +(pk(A, n(B, r_1))), -(pk(B, Y)), +(Y \oplus n(B, r_1)) \mid nil] \;\&$$
$$SS \;\&\; \{n(A, r_2) \in \mathcal{I}, IK\}\}$$

This attack pattern should be possible in Maude-NPA by just applying the substitution $X \mapsto 0$, where 0 is the XOR unit element. However, the term $X \oplus n(A, r_2)$ becomes reducible under such substitution and the attack would not be reachable because of our irreducibility condition on $X \oplus n(A, r_2)$. To solve this problem, the key idea is that the pattern $X \oplus n(A, r_2)$ should be replaced by its *variants* before each contextual narrowing step, i.e., by the possible instance patterns of it which are irreducible, namely: (i) the pattern $X \oplus n(A, r_2)$ itself, (ii) the pattern Y, which is the normal form after applying substitution $X \mapsto Y \oplus n(A, r_2)$, (iii) the pattern 0, which is the normal form after applying substitution $X \mapsto n(A, r_2)$, and (iv) the pattern $n(A, r_2)$, which is the normal form after applying substitution $X \mapsto 0$. Only after replacement of the original term by these variants, can we impose the irreducibility conditions for reducing the search space. That is, for contextual reachability analysis, we need to first compute what we call the ϕ, E_0, B-variants of a term. ■

Definition 8.14 (ϕ, E_0, B-variants) *Given a contextual rewrite theory $(\Sigma, B, E_0, T, \phi)$, the set of E_0,B,ϕ-variants of a pair $\langle t, \Pi \rangle$ for t a term and Π a set of irreducible terms is defined as $[\![\langle t, \Pi \rangle]\!]^{\phi}_{E_0,B} = \{(t\sigma[v_1, \ldots, v_n]_{p_1,\ldots,p_n}, \sigma) \mid (g(v_1, \ldots, v_n), \sigma) \in [\![g(t|_{p_1}, \ldots, t|_{p_n})]\!]_{E_0,B} \wedge \forall u \in \Pi : u\sigma$ is ϕ, E_0, B-irreducible$\}$ where $\phi(t) = \{p_1, \ldots, p_n\}$ and g is an auxiliary function symbol not appearing in E_0 and B. For readability, we write $\langle t, \Pi \rangle \twoheadrightarrow^{\theta}_{E_0,B} \langle w, \overline{\Pi} \rangle$ to denote that $(w, \theta) \in [\![\langle t, \Pi \rangle]\!]^{\phi}_{E_0,B}$ and $\overline{\Pi} = \Pi\theta \cup \{w\}$.*

Example 8.15 Let us consider the state t' shown in Example 8.7:

$$\{ :: r_2 :: [nil, -(pk(A, X)), +(pk(B, n(A, r_2))) \mid -(X \oplus n(A, r_2)), nil] \;\&$$
$$:: r_1 :: [nil, +(pk(A, n(B, r_1))), -(pk(B, Y)), +(Y \oplus n(B, r_1)) \mid nil] \;\&$$
$$SS \;\&\; \{(X \oplus n(A, r_2)) \in \mathcal{I}, IK\}\}$$

We generate the four variants associated to $X \oplus n(A, r_2)$ in subterms rooted by $-(\_)$ and $\_\in\mathcal{I}$, since these are the symbols with irreducibility constraints: (i) the original one but with the assumption that X will

never contain either $n(A, r_2)$ or 0, (ii) the pattern $n(A, r_2)$ where $X \oplus n(A, r_2)$ has been collapsed into the nonce, (iii) the pattern Z where $X \oplus n(A, r_2)$ has been collapsed into a new variable Z by assuming $X \mapsto Z \oplus n(A, r_2)$, and (iv) the term 0 where $X \oplus n(A, r_2)$ has been collapsed into 0 by assuming $X \mapsto n(A, r_2)$:

$$\{ :: r_2 :: [nil, -(pk(A, X)), +(pk(B, n(A, r_2))) \mid -(X \oplus n(A, r_2)), nil] \;\&$$
$$:: r_1 :: [nil, +(pk(A, n(B, r_1))), -(pk(B, Y)), +(Y \oplus n(B, r_1)) \mid nil] \;\&$$
$$SS \;\&\; \{(X \oplus n(A, r_2)) \in \mathcal{I}, IK\}\}$$

$$\{ :: r_2 :: [nil, -(pk(A, 0)), +(pk(B, n(A, r_2))) \mid -(n(A, r_2)), nil] \;\&$$
$$:: r_1 :: [nil, +(pk(A, n(B, r_1))), -(pk(B, Y)), +(Y \oplus n(B, r_1)) \mid nil] \;\&$$
$$SS \;\&\; \{n(A, r_2) \in \mathcal{I}, IK\}\}$$

$$\{ :: r_2 :: [nil, -(pk(A, Z \oplus n(A, r_2))), +(pk(B, n(A, r_2))) \mid -(Z), nil] \;\&$$
$$:: r_1 :: [nil, +(pk(A, n(B, r_1))), -(pk(B, Y)), +(Y \oplus n(B, r_1)) \mid nil] \;\&$$
$$SS \;\&\; \{Z \in \mathcal{I}, IK\}\}$$

$$\{ :: r_2 :: [nil, -(pk(A, n(A, r_2))), +(pk(B, n(A, r_2))) \mid -(0), nil] \;\&$$
$$:: r_1 :: [nil, +(pk(A, n(B, r_1))), -(pk(B, Y)), +(Y \oplus n(B, r_1)) \mid nil] \;\&$$
$$SS \;\&\; \{0 \in \mathcal{I}, IK\}\}$$

The reader can check that only the variants of the terms in the intruder knowledge (which are indeed coming from messages of the form $-(M)$) are generated. ■

The key idea to achieve the desired semantic equivalence between contextual narrowing and ordinary narrowing is to precede each contextual narrowing step by a ϕ-variant computation step, which we prove in the following lemma.

Lemma 8.16 *Given a contextual rewrite theory $(\Sigma, B, E_0, T, \phi)$, and a term t, and a (possibly empty) set of irreducible terms Π, we have a narrowing step $t \overset{\theta}{\leadsto}_{T, E_0 \cup B} t'$ iff there are substitutions θ', θ'', τ and terms t'' and w such that $\langle t, \Pi \rangle \twoheadrightarrow^{\theta'}_{E_0, B} \langle w, \Pi' \rangle$, $\langle w, \Pi' \rangle \overset{\theta''}{\leadsto}_{T, E_0, B, \phi} \langle t'', \overline{\Pi'} \rangle$, $t' =_B t''\tau$, $\Pi' = \Pi \cup \{w\}$, and $\overline{\Pi'} = \Pi'\theta''$. Furthermore, $t\theta =_{E_0 \cup B} t\theta' \circ \theta'' \circ \tau$, and $t\theta \rightarrow^*_{E_0, B} w\theta'' \circ \tau$.*

Proof. The narrowing step $t \overset{\theta}{\leadsto}_{T,E_0\cup B} t'$ implies $\theta \in CSU_{E_0\cup B}(t = l)$ for some rule $l \to r \in T$. But then, $\theta \in CSU_{E_0\cup B}(t = l)$ iff there is $(u, \alpha) \in [\![\langle t, \Pi\rangle]\!]^{\phi}_{E_0,B}$ such that $(u, \alpha) \sqsupseteq_{E_0,B} (t, \theta|_{\mathcal{V}ar(t)})$, i.e., there is a substitution τ s.t. $(t\theta) \to^{*}_{E_0,B} u\tau$ and $(\theta{\downarrow_{E_0,B}})|_{\mathcal{V}ar(t)} =_B (\alpha \circ \tau)|_{\mathcal{V}ar(t)}$. Therefore, there is a substitution $\rho \in CSU^{V}_{E_0\cup B}(l =_{\downarrow_\phi} u)$ with $\mathcal{V}ar(u) \cup Ran(\alpha) \subseteq V$ such that $\rho|_{\mathcal{V}ar(u)} =_B \tau|_{\mathcal{V}ar(u)}$ and, thus, $\theta|_{\mathcal{V}ar(t)} =_{E_0\cup B} (\alpha \circ \rho)|_{\mathcal{V}ar(t)}$. □

Now, we provide the main result of this chapter.

Theorem 8.17 (Contextual Soundness and Completeness) *Given a contextual rewrite theory $(\Sigma, B, E_0, T, \phi)$, a reachability goal $t \overset{?}{\to}^{*} t'$, and a solution σ of it, there are a set of terms $u_1, \ldots, u_n, w_1, \ldots, w_{n+1}, t''$ and a set of substitutions $\theta_1, \ldots, \theta_{n+1}, \theta'_1, \ldots, \theta'_{n+1}$ such that*

$$\begin{aligned}
\langle t, \Pi_0\rangle &\twoheadrightarrow^{\theta_1}_{E_0,B} \langle w_1, \Pi_1\rangle \quad \overset{\theta'_1}{\leadsto}_{T,E_0,B,\phi} \langle u_1, \overline{\Pi_1}\rangle \\
&\twoheadrightarrow^{\theta_2}_{E_0,B} \langle w_2, \Pi_2\rangle \quad \overset{\theta'_2}{\leadsto}_{T,E_0,B,\phi} \langle u_2, \overline{\Pi_2}\rangle \\
&\quad\vdots \\
&\twoheadrightarrow^{\theta_n}_{E_0,B} \langle w_n, \Pi_n\rangle \quad \overset{\theta'_n}{\leadsto}_{T,E_0,B,\phi} \langle u_n, \overline{\Pi_n}\rangle \\
&\twoheadrightarrow^{\theta_{n+1}}_{E_0,B} \langle w_{n+1}, \Pi_{n+1}\rangle \overset{\theta'_{n+1}}{\leadsto}_{T,E_0,B,\phi} \langle t'', \overline{\Pi_{n+1}}\rangle
\end{aligned}$$

and also: (i) $\Pi_0 = \varnothing$, $\Pi_1 = \{w_1\}$, $\overline{\Pi_1} = \theta'_1(\Pi_1)$, $\Pi_2 = \theta_2(\overline{\Pi_1}) \cup \{w_2\}$, $\overline{\Pi_2} = \theta'_2(\Pi_2)$, …, $\Pi_{n+1} = \overline{\Pi_n} \cup \{w_{n+1}\}$, $\overline{\Pi_{n+1}} = \theta'_{n+1}(\Pi_{n+1})$, (ii) for each $i \in \{1, \ldots, n+1\}$, the term $w_i\theta'_i \circ \theta_{i+1} \circ \theta'_{i+1} \circ \cdots \circ \theta_{n+1} \circ \theta'_{n+1}$ is ϕ, E_0, B-irreducible, (iii) there is a substitution τ such that $\sigma =_B \theta_1 \circ \theta'_1 \circ \theta_2 \circ \theta'_2 \cdots \theta_{n+1} \circ \theta'_{n+1} \circ \tau$, and (iv) $t' =_B t''\tau$.

Conversely, any substitution σ for which there is a sequence as above satisfying conditions (i)-(iv) is a solution of $t \overset{?}{\to}^{*} t'$.

Proof. By successive application of Lemma 8.16.

Example 8.18 Continuing Example 8.15, we have four state patterns after variant generation. Contextual narrowing follows from the first variant state pattern as described in Example 8.19 below. The second variant state pattern will lead to an initial state where the intruder provides message $pk(A, 0)$. and the vertical bar of Bob's strand is at the beginning of the strand. And the third and the fourth variant state patterns will be

discarded by Maude-NPA, since they do not satisfy the syntactic check explained in the Motivation discarding states sending a nonce before it is generated. The state space reduction achieved in Maude-NPA is huge by using the irreducibility conditions on symbols $-(\_)$ and $\_\in\mathcal{I}$ and other state space reduction techniques based on such conditions (we further discuss experiments on this topic in Section 8.4.1). ■

Condition (ii) in Theorem 8.17 for terms w_i to be (ϕ-)irreducible after substitution application ensures that variants are not computed more than once for each irreducible subterm in term t or irreducible subterms introduced by right-hand sides of rules. This is very important to further reduce the search space.

Example 8.19 Let us consider the state t' shown in Example 8.7. After several variant generation and contextual narrowing steps using the reversed form of rewrite rules (3.1)–(3.4), the following state is found

$$\begin{aligned}
&\{ :: r_2 :: [nil, -(pk(A,X)), +(pk(B,n(A,r_2))) \mid -(X \oplus n(A,r_2)), nil] \;\&\\
&\quad :: r_1 :: [nil, +(pk(A,n(B,r_1))) \mid -(pk(B, X \oplus n(A,r_2) \oplus n(B,r_1))),\\
&\qquad\qquad\qquad\qquad +(X \oplus n(A,r_2) \oplus n(B,r_1) \oplus n(B,r_1)), nil]\\
&\;\&\; SS \;\&\; \{\; pk(B, X \oplus n(A,r_2) \oplus n(B,r_1)) \in \mathcal{I},\; (X \oplus n(A,r_2)) \notin \mathcal{I},\; IK \;\}\}
\end{aligned}$$

We can check that there is no contextual unifier that allows terms $pk(B, n(A, r_2))$ and $pk(B, X \oplus n(A, r_2) \oplus n(B, r_1))$ to be unifiable according to the reversed form of rewrite rule (3.3), since the second term is under a symbol with irreducibility restrictions and the substitution $X \mapsto n(B, r_1)$ would make it reducible.

However, another protocol session can be used, since the term $pk(B, X \oplus n(A, r_2) \oplus n(B, r_1))$ can be unified with term $pk(B, n(A', r_2'))$ coming from another session, using the contextual unifier $X \mapsto n(A', r_2') \oplus n(A, r_2) \oplus n(B, r_1)$. The resulting state is as follows

$$
\begin{array}{l}
\{ :: r_2' :: [nil, -(pk(A', X')) \mid +(pk(B, n(A', r_2'))), -(X' \oplus n(A', r_2')), nil] \ \& \\
\quad :: r_2 :: [nil, \ -(pk(A, n(A', r_2') \oplus n(A, r_2) \oplus n(B, r_1))), \\
\qquad\qquad +(pk(B, n(A, r_2))) \mid \\
\qquad\qquad -(n(A', r_2') \oplus n(A, r_2) \oplus n(B, r_1) \oplus n(A, r_2)), nil] \ \& \\
\quad :: r_1 :: [nil, \ +(pk(A, n(B, r_1))) \mid \\
\qquad\qquad -(pk(B, n(A', r_2') \oplus n(A, r_2) \oplus n(B, r_1) \oplus n(A, r_2) \oplus n(B, r_1))), \\
\qquad\qquad +(n(A', r_2') \oplus n(A, r_2) \oplus n(B, r_1) \oplus n(A, r_2) \\
\qquad\qquad\quad \oplus n(B, r_1) \oplus n(B, r_1), nil] \ \& \\
SS \ \& \ \{ \ \ pk(B, n(A', r_2') \oplus n(A, r_2) \oplus n(B, r_1) \oplus n(A, r_2) \oplus n(B, r_1)) \notin \mathcal{I}, \\
\qquad (n(A', r_2') \oplus n(A, r_2) \oplus n(B, r_1) \oplus n(A, r_2)) \notin \mathcal{I}, \ IK \ \}\}
\end{array}
$$

However, although the two contextual narrowing steps have computed contextual unifiers, the combination of both unifiers does not satisfy the irreducibility conditions of the original term $-(X \oplus n(A, r_2))$, since now it is reducible, i.e., the term $-(n(A', r_2') \oplus n(A, r_2) \oplus n(B, r_1) \oplus n(A, r_2))$ is reducible. Therefore, this narrowing sequence is discarded, since it does not fulfill the conditions for solutions of contextual reachability goals given in Theorem 8.17, further reducing search. ■

The question now arises of how to produce such asymmetric algorithms that improve upon the generic variant-based algorithm described above. We discuss one such approach in Section 8.3, using as example a special-purpose asymmetric unification algorithm for exclusive-or that has been used to perform the experiments presented in Section 8.4.

8.3 An Asymmetric Unification Algorithm for the Theory of Exclusive-OR with Uninterpreted Function Symbols

There are two metrics to be considered when optimizing asymmetric unification algorithms for cryptographic protocol analysis. One of course is speed of execution. The other is the size of the most general set of unifiers (mgu). Each such unifier results in the production of a new state, so minimizing the size of this set helps to keep the size of the state space down.

One way of minimizing both execution time and mgu size is to convert a symmetric algorithm that has already been optimized for these features. In that case, we need to keep unifiers produced by the original algorithm whenever possible. We outline a general approach and illustrate it for the exclusive-or (XOR) theory given in Section 8.1, together with uninterpreted function symbols, chosen because it is the simplest theory appearing in cryptographic protocol analysis that combines both cancellation rules and a non-trivial theory B in the decomposition (Σ, B, E_0)

Given a decomposition (Σ, B, E_0) of the equational theory (Σ, E), and an asymmetric unification problem $\Gamma = \{t_1 =_{\downarrow} t'_1, \ldots, t_n =_{\downarrow} t'_n\}$, the key steps of the approach are:

1. First compute a complete finite set S of B-unifiers using a finitary unification algorithm for B. If S is empty, then there are no asymmetric unifiers.

2. For each such unifier σ from the previous step, check whether every $t'_i\sigma$ is in E_0,B-normal form. All such unifiers are retained also as asymmetric unifiers.

3. For a unifier σ such that some $t'_i\sigma$ is not in E_0,B-normal form, compute an equivalent asymmetric unifier if possible.

4. If both of the previous steps fail, this implies that σ or its equivalents cannot be asymmetric unifiers in their full generality. However, there may be some instances obtained by instantiating variables in them which are asymmetric unifiers. A complete set of instances of a given unifier is generated by suitably instantiating variables. This step may be expensive, so it is employed only as a last resort (as demonstrated in Table 8.6 of Section 8.4.2 using unification problems manually chosen to stress this point). For each such instance the above steps are repeated.

We explain below how steps (1)–(4) yield an asymmetric unification algorithm for exclusive-or with uninterpreted symbols (XOR) from a symmetric one. Variables appearing in Γ are called *original variables* to distinguish them from new variables, called *support variables* by the inference rules. For a unification problem Γ and an XOR unifier σ we say

in the assignment $x \mapsto t \oplus T \in \sigma$ some original variable x has a *conflict* at some simple term t if

- there exists $u =\downarrow v[x \oplus s] \in \Gamma$ and
- there exists T' such that $s\sigma = t \oplus T'$

where s and t are *simple terms* (i.e., a term that does not have $\oplus$ as its outermost symbol) and T' might be empty. The significance of conflicts is that a substitution of x cannot include t as a subterm, in order to ensure the irreducibility of the right side of equations in Γ.

We present the algorithm as a collection of inference rules on a triple of sets:

$$\frac{\sigma \| \Upsilon \| \Delta}{\sigma' \| \Upsilon' \| \Delta'},$$

where σ is an XOR unifier of Γ, Υ is a set of *constraint pairs* in which each member has the form (v, s) (to mean that a substitution of v cannot include s as a subterm to ensure the irreducibility of the right side of equations in Γ), and Δ is a set of disequations of the form $s \oplus t \neq^? 0$, with s and t having the same topmost uninterpreted function symbol.

A complete set of XOR-unifiers is first generated using an XOR-unification algorithm. For each XOR unifier σ, the algorithm starts with a triple $\sigma \| \varnothing \| \varnothing$. The algorithm may generate numerous branches, some of which lead to a dead end because either (i) no inference rule is applicable or (ii) the candidate for an XOR unifier violates a constraint in the second component or a disequation in the third component. Different branches can generate equivalent asymmetric unifiers or asymmetric unifiers which are instances of other asymmetric unifiers.

We use the following notation. The result of applying a substitution θ to $\Upsilon = \{(v_1, s_1), \cdots, (v_n, s_n)\}$ is $\Upsilon\theta = \{(v_i, s_i\theta\downarrow) | (v_i, s_i) \in \Upsilon\}$; we will rewrite $(v_i, t_1 \oplus \cdots \oplus t_n)$ to $(v_i, t_1), \cdots, (v_i, t_n)$. A substitution δ satisfies Υ iff δ satisfies every constraint pair in Υ, i.e., given a pair $(v, s) \in \Upsilon$, δ satisfies (v, s) iff $v\delta \oplus s\delta$ is irreducible using E_0, B, were E_0 are the rewrite rules for the XOR theory given in Section 8.1,. If δ does not satisfy Υ, then δ violates Υ. Similarly, δ satisfies Δ iff δ satisfies every disequation $s \oplus t \neq 0 \in \Delta$, in other words $(s\delta \oplus t\delta)$ does not rewrite to 0.

8.3.1 The Inference System

All inference rules below are don't care nondeterministic rules. They are grouped as: **Splitting**, **Branching** and **Instantiation**. The algorithm runs in two phases. In the first phase, the **Splitting** and **Branching** rules are applied, attempting to generate an asymmetric XOR unifier equivalent to the original XOR unifier. The **Splitting** rule is applied as much as possible to (i) move all toplevel original variables out of the range of an XOR unifier, while (ii) eliminating conflicts between original variables and subterms with which they appear in t_i's in Γ. Once it is no longer applicable, an XOR unifier equivalent to the original unifier is constructed such that its range only includes new variables at top levels. Then, branching rules are repeatedly applied attempting to eliminate conflicts between support variables with other variables and nonvariable subterms The **Non-Variable Branching** rule, which eliminates a conflict between a support variable and a nonvariable subterm, is repeatedly applied first. This is followed by (i) the **Auxiliary Branching** rule and (ii) the **Variable Branching** rule. The last two rules may not eliminate any conflicts; however they are helpful later during the second phase. In this first phase, if any of the branches yields an asymmetric XOR unifier, the algorithm terminates; it is not necessary to consider other branches as all asymmetric XOR unifiers from various branches are equivalent.

If the first phase does not succeed in generating an equivalent asymmetric XOR unifier, all branches generated from the first phase must be considered in the second phase. Instantiation rules are now applied to generate instances of equivalent XOR unifiers. The **Decomposition Instantiation** rule generates instances of an XOR unifier so that the rules $x \oplus x \oplus y \mapsto y$ and $x \oplus x \mapsto 0$ are applicable, whereas the **Elimination Instantiation** rule generates instances by setting some support variables to 0. It is possible that an XOR unifier generated by the **Elimination Instantiation** rule is equivalent to the original XOR unifier (since it may have been generated by instantiating a support variable to 0 implying that it was unnecessary to introduce that support variable).

If along a branch, a result of **Decomposition Instantiation** is not an asymmetric XOR unifier, the algorithm moves again to the first phase and applies **Splitting**, since some of the original variables underneath interpreted function symbols may get elevated to the top level in substi-

tutions of original variables. **Elimination Instantiation** is repeatedly applied only after **Decomposition** cannot be applied any further. If the result is not an asymmetric XOR unifier, then the **Branching** rules are applied by returning to the first phase (**Splitting** is not applicable in this case).

8.3.2 The Splitting Rule

This rule transforms an XOR unifier σ into an equivalent XOR unifier σ' such that all the top variables in $Range(\sigma')$ are support variables.

$$\frac{[x \mapsto y \oplus S \oplus T] \cup \sigma \| \Upsilon \| \Delta}{([x \mapsto y \oplus S \oplus T] \cup \sigma) \circ \theta \| \Upsilon\theta \| \Delta\theta}$$

where $\theta = \{y \mapsto v \oplus S\}$ and v is a fresh support variable. The rule is applied only if (i) $x, y \in \mathcal{V}ar(\Gamma)$ and (ii) $y \notin \mathcal{V}ar(S)$.

Even though S and T can be chosen in any way, if x has a conflict at some simple term s in $S \oplus T$, then for efficiency in our implementation, we will put s into S, unless $y \in \mathcal{V}ar(s)$. After **Splitting** there will be no top level original variables in the range of σ. So from now on, we assume that all the top variables which appear in the range of σ are support variables.

8.3.3 The Branching Rules

The main objective in applying the two branching rules is to try to transform an XOR unifier into an equivalent one without conflicts.

Non-Variable Branching. This rule considers the case that some original variable x has a conflict at some non-variable simple term s.

$$\frac{\sigma \| \Upsilon \| \Delta}{\sigma \circ \theta \| (\Upsilon[v'/v] \cup (v', s))\theta \| \Delta\theta \quad \bigvee \quad \sigma \| \Upsilon \cup \{(v, s)\} \| \Delta\theta}$$

where there exists an assignment $[x \mapsto v \oplus s \oplus S] \in \sigma$ and $\theta = [v \mapsto v' \oplus s]$ with v' being a fresh support variable, under the conditions that x has a conflict at a simple nonvariable terms s in Γ where (i) $v \notin \mathcal{V}ar(s)$ and (ii) $(v, s) \notin \Upsilon$.

Above, $\Upsilon[v'/v]$ means: replace all occurrences of the variable v in the first component of every pair in Υ by the variable v'. The first branch is used when the conflict between x and s is successfully resolved using v by introducing a new support variable v'; the second branch is used when that is not possible, thus leading to an additional constraint (v, s) implying that v and s are in conflict.

Auxiliary Branching. This rule is applied when an original variable conflict with another original variable in Γ and their substitutions in an XOR unifier share a common part.

$$\frac{\sigma \| \Upsilon \| \Delta}{\sigma \circ \theta \| (\Upsilon[v'/v] \cup (v', s))\theta \| \Delta\theta \quad \bigvee \quad \sigma \| \Upsilon \cup \{(v, s)\} \| \Delta}$$

where $\theta = \{v \mapsto v' \oplus s\}$ with v' being a fresh support variable, and there exist two assignments $[x \mapsto v \oplus s \oplus S, y \mapsto v \oplus S']$ in σ. This rule is applied only if (i) x, y are in conflict in Γ, (ii) s is a simple non-variable term and $v \notin \mathcal{V}ar(s)$ and (iii) $(v, s) \notin \Upsilon$.

The additional simple nonvariable term s in the substitution for x in an XOR unifier is used to possibly eliminate the conflict with a new variable v', which stands for the common shared part of x and y. The reader will notice that unlike the **Non-Variable Branching** rule, both branches after this rule still have conflicts in the substitutions of x and y which are in conflict in Γ. So this rule does not solve the conflict directly; it is preparing for the instantiation part.

Variable Branching. This rule is similar to the **Auxiliary Branching** rule and is applied when two original variables x and y have a conflict in Γ and share a common support variable v_1 in their substitutions in an XOR unifier. The key difference from the **Auxiliary Branching** rule is that instead of the substitution for x having a simple nonvariable term that is not in conflict with v_1, it has another support variable v_2. The common support variable v_1 is then split into two parts: the common part of x and y, represented by v_{12}, and the remaining parts of x and y, represented by v_1' and v_2', respectively.

$$\frac{\sigma \| \Upsilon \| \Delta}{\sigma \circ \theta \| \Upsilon'\theta \| \Delta\theta \quad \bigvee \quad \sigma \| \Upsilon \cup \{(v_1, v_2)\} \| \Delta}$$

where σ includes $[x \mapsto v_1 \oplus v_2 \oplus S, y \mapsto v_1 \oplus S']$, $\theta = [v_1 \mapsto v_{12} \oplus v_1', v_2 \mapsto v_{12} \oplus v_2']$, v_{12}, v_1' and v_2' are fresh support variables, and $\Upsilon' = (\Upsilon[v_{12}/v_1][v_{12}/v_2] \cup \Upsilon[v_1'/v_1] \cup \Upsilon[v_2'/v_2] \cup \{(v_{12}, v_1'), (v_{12}, v_2'), (v_1', v_2'), (v_1', v_{12}), (v_2', v_{12}), (v_2', v_1')\}$. This rule is applied only if (i) x and y have a conflict in Γ and (ii) $(v_1, v_2) \notin \Upsilon$.

The first branch is the case when v_1 and v_2 have a common part, whereas the second branch is the case when v_1 and v_2 have nothing in common.

8.3.4 Instantiation Rules

The following instantiation rules are used for solving conflicts by instantiating support variables based on the equations $x+x \to 0$ and $x+0 \to x$

Decomposition Instantiation. This rule is used to solve the case that some original variable x has a conflict with a simple nonvariable term t.

$$\frac{\sigma \| \Upsilon \| \Delta}{\sigma \circ \theta_1 \| \Upsilon\theta_1 \| \Delta\theta_1 \quad \bigvee \cdots \bigvee \quad \| \sigma \circ \theta_n \| \Upsilon\theta_n \| \Delta\theta_n \quad \bigvee \quad \sigma \| \Upsilon \| \Delta''}$$

where there exists an assignment $[x \mapsto s \oplus t \oplus S]$ in σ, x has a conflict with a simple nonvariable subterm s in Γ and s and t have the same topmost uninterpreted symbol; $\{\theta_1, \cdots, \theta_n\}$ is a complete set of XOR unifiers of $s \stackrel{?}{=} t$ and $\Delta'' = \Delta \cup \{s \oplus t \neq^? 0\}$.

Elimination Instantiation. This rule is used to solve the case that some original variable x has a conflict at some support variable v.

$$\frac{[x \mapsto v \oplus S] \cup \sigma \| \Upsilon \| \Delta}{([x \mapsto S] \cup \sigma) \circ \theta \| \Upsilon\theta \| \Delta\theta}$$

where $\theta = \{v \mapsto 0\}$, x and y are in conflict in Γ for some y. The rule is applied only if $y\sigma = v \oplus S'$ with S' having at least one subterm.

Because v maps to 0, all pairs (v, s) in Υ will be removed from Υ.

The following result proves that the asymmetric unification algorithm for the theory of XOR theory presented in this section satisfies the desirable properties.

Theorem 8.20 (Soundness, completeness, and termination) [Liu, 2012] *The asymmetric unification algorithm described above is sound, terminating, and complete.*

8.4 Experimental Evaluation

In this section we present the result of the experimental evaluation we have performed comparing asymmetric unification with other standard unification techniques for two different purposes: as a mechanism used for cryptographic protocol analysis, and as a mechanism to solve unification problems. More specifically, in Section 8.4.1 we first compare the contextual symbolic reachability approach presented in Section 8.2 w.r.t other standard approaches, for the analysis of three cryptographic protocols involving the exclusive-or theory. Then, in Section 8.4.2 we compare the results obtained with an asymmetric XOR unification algorithm (implemented as explained in Section 8.3), and a variant-based XOR algorithm, when solving several unification problems arising in the analysis of the three protocols used for the experiments of Section 8.4.1.

We have used three protocols using exclusive-or: (i) the running protocol (RP) of Example 8.1, (ii) the Wired Equivalent Privacy Protocol (WEPP) of [MAC, 1999], and (iii) the TMN protocol of [Tatebayashi et al., 1990; Lowe and Roscoe, 1997].

8.4.1 Experiments of Contextual Symbolic Analysis of Cryptographic Protocols

We have performed several experiments to compare the contextual symbolic reachability approach presented in Section 8.2 with other approaches.

In Table 8.1, we compare the standard reachability analysis of Section 2.2, which uses the XOR unification algorithm developed in [Liu and Lynch, 2011], and the contextual reachability analysis of Section 8.2, which uses the asymmetric XOR unification algorithm of Section 8.3. We show the number of states generated from one level to the next one of the backwards reachability tree with the indicated number of steps as the maximum depth. We also include the execution time from one level to the next one. We write "timeout" when the tool did not finish within

states/seconds	1 step	2 steps	3 steps	4 steps	5 steps
RP - Standard	2/0.08	5/0.16	13/0.86	49/3.09	267/17.41
RP - Contextual	1/0.03	45/1.08	114/2.26	1175/37.25	13906/4144.30
WEPP - Standard	5/0.09	9/0.42	26/1.27	106/5.80	503/ 34.76
WEPP - Contextual	4/0.05	9/0.12	26/0.64	257/144.65	2454/612.08
TMN - Standard	5/0.11	15/ 0.55	99/3.82	469/ 25.68	timeout
TMN - Contextual	4/0.06	24/0.53	174/3.63	1079/170.29	9737/1372.55

Table 8.1: Experiments with standard reachability analysis using regular XOR unification algorithm vs contextual reachability analysis using asymmetric XOR unification algorithm. A pair n/t means: n = number of states, and t = time in seconds.

a time interval of two hours.

As shown in Table 8.1, contextual reachability analysis is not better than the standard reachability analysis because of variant generation, which creates many more states than may be necessary for rule application. However, although typically many more states are created, the use of variants and irreducibility constraints is crucial (as explained in the Motivation) for further optimizations of the search space, as shown in Table 8.2 below, which shows that contextual reachability analysis enables several Maude-NPA optimizations, including grammars (see Chapter 4 for details) and drastically reduces the search space.

For the three cryptographic protocols mentioned above, we are able to find the associated attacks in Table 8.2 below. Table 8.2 shows that, although, due to the extra computations needed for the optimization, the execution time without optimizations (-Opt) is sometimes better than with optimizations (+Opt), this only happens up to Step 3. The important point is that from Step 2 on, the total number of states is *drastically reduced* when optimizations are added (the only exception at Step 1 is RP, due to some differences on how variants are generated). In fact, the crucial point is not just the great reduction in the number of states, but the *finiteness of the analysis* for all the examples with optimization, whereas no such finiteness is even theoretically possible without optimizations. This is particularly important when an attack does *not* exist, since then finiteness of the analysis *proves* that the protocol is secure against such an attack. Therefore, the above performance results validate experimentally the main thesis of this chapter, namely that: (i)

states/seconds	1 step	2 steps	3 steps	4 steps	5 steps	Finite?
RP-Opt	1/0.03	45/1.08	114/2.26	1175/37.25	13906/4144.30	No, timeout with 6 steps
RP+Opt	4/0.59	7/0.59	7/1.92	7/1.89	7/3.02	Yes, at step 10
WEPP-Opt	4/0.05	9/0.12	26/0.64	257/144.65	2454/612.08	No, timeout with 7 steps
WEPP+Opt	2/0.36	2/0.20	1/0.80	2/1.42	1/0.03	Yes, at step 5
TMN-Opt	4/0.06	24/0.53	174/3.63	1079/170.29	9737/1372.55	No, timeout with 7 steps
TMN+Opt	3/0.42	6/9.85	9/1.78	9/4.43	8/3.20	Yes, at step 21

Table 8.2: Experiments for contextual reachability analysis using asymmetric XOR unification algorithm with and without optimizations

support of irreducibility conditions in symbolic reachability is essential for *effective* protocol analysis, since crucial optimizations depend on such conditions; and (ii) contextual reachability analysis supports irreducibility conditions in a sound and complete way and makes such optimizations possible.

The integration of this framework into Maude-NPA is still under testing and optimization is needed to increase performance. Indeed, the current experiments have been performed with a version of the contextual narrowing simpler than the conditions of Theorem 8.17 (irreducibility constraints on Π are not enforced), but is still valid for the benchmarked protocols, i.e., in these protocols, each strand contains only one expression using the XOR operator, and thus Π remains irreducible by default.

8.4.2 Experiments with Unification Problems Arising in Protocol Analysis

We implemented a variant-based algorithm for XOR and an algorithm produced by applying the procedure outlined in Section 8.3 to the special-purpose XOR algorithm of [Liu and Lynch, 2011] in Maude-NPA and experimentally compared their performance.

Tables 8.3, 8.4 and 8.5 gather the results of unification problems from the RP, WEPP, and TMN protocols, respectively. Table 8.6 gathers the results of some more complex problems manually defined by the authors

Unif. Problem	T. A-V	# A-V	T. D-A	# D-A	% T.	% #
$NS_1 \oplus NS_2 =\downarrow NS_3 \oplus N_A$	153	12	153	1	0	91
$NS_1 \oplus N_A =\downarrow NS_2 \oplus NS_3$	137	5	121	1	11	80
$NS_1 \oplus NS_2 =\downarrow NS_3 \oplus NS_4 \oplus NS_5$	286	54	116	1	59	98
$NS_1 \oplus NS_2 =\downarrow NS_3 \oplus NS_4 \oplus N_A$	159	36	115	1	27	97
$NS_1 \oplus NS_2 =\downarrow N_A$	127	4	114	1	10	75
$NS_1 \oplus NS_2 =\downarrow null$	128	1	105	1	17	0
$NS_1 \oplus NS_2 =\downarrow null \oplus NS_3$	130	7	105	1	20	85

Table 8.3: Unification Problems in RP protocol.

Unif. Problem	T. A-V	# A-V	T. D-A	# D-A	% T.	% #
$M_1 \oplus M_2 =\downarrow M_3 \oplus pair(V_1, M_4)$	51	12	44	1	13	91
$pair(V, rc4(V_1, kAB) \oplus ([N_A, c(N_A)])) =\downarrow pair(V_1, M_1)$	30	1	29	1	3	0
$M_1 \oplus M_2 =\downarrow M_3 \oplus V_1$	33	12	32	1	3	91
$M_1 \oplus M_2 =\downarrow M_3 \oplus ([N_1, c(N_2)])$	34	12	30	1	11	91
$M_1 \oplus M_2 =\downarrow M_3 \oplus pair(V_1, pair(V_2, M_4))$	36	12	30	1	16	91

Table 8.4: Unification Problems in WEPP protocol.

to stress the algorithms. Here each unification problem combines several subproblems, shown below the table. The RP, WEPP and TMN protocols were used in the experiments presented in 8.4.1, in order to compare the contextual symbolic reachability approach presented in that paper with other approaches. However, the experiments presented in this Section are more focused on concrete unification problems that occur during the analysis of these protocols and the efficiency of asymmetric unification algorithms when solving them in terms of number of unifiers and execution time.

In each table the second and third columns show, respectively, the execution time (in milliseconds) and the number of unifiers obtained using the asymmetric variant-based unification algorithm. The fourth and fifth columns show, respectively, the execution time (in milliseconds) and the number of unifiers obtained using the special-purpose asymmetric unification algorithm for exclusive-or. Finally, the two last columns present a percentage that reflects the performance improvement of the special-purpose asymmetric unification algorithm with respect to the asymmetric variant-based algorithm in terms of execution time and number of unifiers obtained, respectively.

On the average the special-purpose asymmetric unification algorithm

Unif. Problem	T. A-V	# A-V	T. D-A	# D-A	% T.	% #
$M_1 \oplus M_2 =\downarrow M_3 \oplus M_4$	115	18	105	1	8	94
$M_1 \oplus M_2 =\downarrow M_3 \oplus M_4 \oplus M_5$	5749	1	74	1	98	0
$M_1 \oplus M_2 =\downarrow M_3 \oplus pair(M_4, M_5)$	71	12	71	1	0	91
$pair(M_1, M_2) =\downarrow pair(M_3, M_4)$	65	1	70	1	-1	0
$M_1 \oplus M_2 =\downarrow pair(M_3, M_4)$	67	4	71	1	0	91
$M_1 \oplus M_2 =\downarrow null \oplus M_3$	66	7	70	1	-6	85

Table 8.5: Unification Problems in TMN protocol.

Unif. Problem	T. A-V	# A-V	T. D-A	# D-A	% T.	% #
$SP4 \wedge SP1 \wedge SP2$	422	4	68	3	83	25
$SP5 \wedge SP1 \wedge SP2$	408	24	131	7	67	70
$SP6 \wedge SP1 \wedge SP2$	416	100	491	15	-18	85
$SP7 \wedge SP1 \wedge SP2$	454	360	3732	31	-722	91
$SP8 \wedge SP1 \wedge SP2 \wedge SP3$	151387	3	47	1	99	66
$SP9 \wedge SP1 \wedge SP2 \wedge SP3$	153913	33	80	3	99	66
$SP10 \wedge SP1 \wedge SP2 \wedge SP3$	154137	201	157	7	99	96
$SP11 \wedge SP1 \wedge SP2 \wedge SP3$	154534	1053	349	15	99	98
$SP12 \wedge SP1 \wedge SP2 \wedge SP3$	160114	5073	829	31	99	99

Table 8.6: Other Unification Problems

SP1 $= M_1 \oplus M_2 =\downarrow M_1 \oplus M_2$
SP2 $= M_1 \oplus M_3 =\downarrow M_1 \oplus M_3$
SP3 $= M_1 \oplus M_4 =\downarrow M_1 \oplus M_4$
SP4 $= M_1 \oplus M_2 \oplus M_3 =\downarrow a \oplus b$
SP5 $= M_1 \oplus M_2 \oplus M_3 =\downarrow a \oplus b \oplus c$
SP6 $= M_1 \oplus M_2 \oplus M_3 =\downarrow a \oplus b \oplus c \oplus d$
SP7 $= M_1 \oplus M_2 \oplus M_3 =\downarrow a \oplus b \oplus c \oplus d \oplus e$
SP8 $= M_1 \oplus M_2 \oplus M_3 \oplus M_4 =\downarrow a$
SP9 $= M_1 \oplus M_2 \oplus M_3 \oplus M_4 =\downarrow a \oplus b$
SP10 $= M_1 \oplus M_2 \oplus M_3 \oplus M_4 =\downarrow a \oplus b \oplus c$
SP11 $= M_1 \oplus M_2 \oplus M_3 \oplus M_4 =\downarrow a \oplus b \oplus c \oplus d$
SP12 $= M_1 \oplus M_2 \oplus M_3 \oplus M_4 =\downarrow a \oplus b \oplus c \oplus d \oplus e$

is about 8% faster than the variant-based one, and generates about 71% fewer unifiers. Note, however, that in many cases the reduction in the number of unifiers is more than 90%. Moreover the asymmetric variant-based unification algorithm does not provide a minimal set of unifiers, whereas the special-purpose asymmetric algorithm does in all our examples. Indeed, all the asymmetric unification problems extracted from protocols have a singleton most general asymmetric unifier, as shown in Tables 8.3, 8.4, and 8.5. However, as shown in Table 8.6, the special-purpose algorithm can sometimes be slower than the variant-based one, even when it generates a smaller most general set of asymmetric unifiers. The reason is that the post-processing step of the algorithm explained in Section 8.3 in which appropriate asymmetric unifiers are only instances of the computed unifiers is sometimes very expensive.

8.5 Conclusions

In this chapter we have defined contextual symbolic reachability analysis, a tool-independent methodology for state exploration, based on unification and narrowing that allows to guarantee that certain terms generated during a state exploration are in *normal form* with respect to the equational theory. Moreover, we have defined asymmetric unification, a new unification paradigm with irreducibility constraints on one side of the problem and shown how asymmetric unification arises in a natural way when analyzing cryptographic protocols. We have also outlined an approach for converting symmetric algorithms to asymmetric ones and applied it to an exclusive-or algorithm. Finally, we have performed several experiments in order to (i) show the benefits of using contextual symbolic reachability and asymmetric unification in Maude-NPA, and (ii) compare the performance of Maude-NPA when using both, the asymmetric XOR algorithm of Section 8.3, and an asymmetric variant-based unification algorithm. The experimental results obtained are encouraging, not only for increasing speed but for reducing the number of unifiers.

Chapter 9

Conclusion

In this thesis we have tackled problems regarding the three main pillars of protocol verification: modelling capabilities, verifiable properties, and efficiency. We have investigated advanced features in the analysis of cryptographic protocols regarding these three main pillars tailored to the Maude-NPA tool. Indeed, all the techniques presented in this thesis were implemented in Maude-NPA and their feasability has been validated via experimental evaluation. In the following, we conclude this thesis and point out the directions for future work amongst the different topics presented.

First, we have studied theoretical aspects such as the definition of Maude-NPA's forwards operational semantics in Chapter 5, and the formalization of contextual symbolic reachability analysis and asymmetric unification, in Chapter 8.

The rewriting-based forwards operational semantics described in Chapter 5 is directly implementable in rule-based programming languages such as Maude, without any need for constraint solving or unification procedures as it is done in most current approaches, allowing us to explore applications such as the simulation of prototypes and reasoning about theories without the finite variant property. This forwards semantics reduces the gap between the Maude-NPA and the realm of standard model checking, shedding some light on how its internal semantics and the logical reachability analysis correspond to an intuitive forward execution of a protocol with the intruder model. This opens up several research directions: the integration of Maude-NPA state space reduc-

tion techniques into the forwards semantics, clarification of the relation of equational theories in the forward semantics, and investigation of how standard model-checking techniques can improve the protocol analysis in the forwards semantics. Also, there is a vast literature in term rewriting and tree automata on forwards and backwards reachability analysis that should be very useful for improving the forwards analysis.

The contextual symbolic reachability analysis explained in Chapter 8 defines a methodology for state exploration that allows to guarantee that certain generated terms are in normal form with respect to the equational theory. These ideas have been applied to define asymmetric unification, where the right-hand sides of a unification problem are enforced to be irreducible. Incorporating such irreducibility constraints allows us to obtain a more efficient equational unification procedure. Since both contextual symbolic reachability analysis and asymmetric unification are tool-independent methodologies, they can be applied to any other tool performing symbolic reachability analysis, modulo equational theories. Furthermore, the benefits of asymmetric unification can be applied to the rest of the topics discussed in this thesis. For example, adapting Maude-NPA's state space reduction techniques to consider irreducibility constraints might probably discard many more states and, therefore, generate smaller state spaces. This seems to be specially interesting for the verification of indistinguishability properties, where equational unification and inequalities modulo equational theories are heavily used.

Second, we have extended Maude-NPA capabilities in two ways. On the one hand, the tool now handles a wider class of protocols since it supports the specification and analysis of sequential protocol compositions in a modular and natural style, as explained in Chapter 6. On the other hand, indistinguishability properties can now be specified and analyzed in Maude-NPA as explained in Chapter 7.

Regarding the protocol composition presented in Chapter 6, property composition, called *additive* composition in [Datta et al., 2003], seems to be an interesting research direction. This would give us the benefits of both model checking (for finding errors and debugging), and logical derivations (for building complex systems out of properties of simple components), allowing to switch between one and the other as needed. Furthermore, we have discovered that sequential protocol composition is a key idea for several other applications in protocol specification such as

protocol branching, secure communication channels, group protocols and protocols with global state memory. Our belief is that these applications can be supported in Maude-NPA. We have performed a preliminary study of these applications but we leave for future work a deeper investigation on these topics.

Our work on protocol indistinguishability of Chapter 7 is a significant step-forward in indistinguishability research, because, for the first time, there is a tool that can perform such automatic checks modulo a very wide class of theories, namely, all theories with the finite variant property, which include theories like Abelian groups, several theories of homomorphic encryption, exclusive-or, and modular exponentiations essential for many privacy-preserving protocols. Much work remains ahead in the definition and verification of indistinguishability in Maude-NPA. First of all, our indistinguishability notion can be further strengthened in various ways that should also be formalized and mechanized in Maude-NPA. For example, our notion of indistinguishability can be extended to cover other issues that may allow the intruder to distinguish two protocols, such as the sorts of the messages. Second, an interesting line of future work in this sense is the handling of protocols with branching, which would allow Maude-NPA to check indistinguishability properties for a wider class of protocols. Third, a more detailed comparison with other indistinguishability notions should be carried out. This is not entirely trivial, since they depend on different models. Fourth, much more experimentation is needed to test the capacity of Maude-NPA to prove indistinguishability for many protocol pairings, and to improve such a capacity.

Finally, in Chapter 4 we have also advanced the research in improving the efficiency of protocol analyses by developing several state space reduction techniques in Maude-NPA. However, more techniques can be developed, e.g. a partial order reduction technique in the standard model-checking style has not been defined in Maude-NPA yet. Moreover, the existing state space reduction techniques can be specialized for protocol composition and indistinguishability properties, thus allowing a more efficient analysis of a wider class of protocols and security properties. Such specialized state space reduction techniques would probably mitigate the state space explosion experienced in the verification of indistinguishability properties for some protocol pairings.

Bibliography

[MAC, 1999] *IEEE 802.11 Local and Metropolitan Area Networks: Wireless LAN Medium Access Control (MAC) and Physical (PHY) Specifications*. 1999.

[Abadi and Cortier, 2006] Martín Abadi and Véronique Cortier. Deciding knowledge in security protocols under equational theories. *Theoretical Computer Science*, 367(1-2):2–32, 2006.

[Abadi and Fournet, 2001] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In Chris Hankin and Dave Schmidt, editors, *POPL*, pages 104–115. ACM, 2001. ISBN 1-58113-336-7.

[Amadio and Lopez, 2000] R. Amadio and V. Lopez. On the reachability problem in cryptographic protocols. In *Concur '00*, pages 380–394. Springer, 2000.

[Anantharaman et al., 2010] Siva Anantharaman, Hai Lin, Christopher Lynch, Paliath Narendran, and Michaël Rusinowitch. Cap unification: application to protocol security modulo homomorphic encryption. In Dengguo Feng, David A. Basin, and Peng Liu, editors, *ASIACCS*, pages 192–203. ACM, 2010. ISBN 978-1-60558-936-7.

[Anlauff et al., 2006] M. Anlauff, D. Pavlovic, R. Waldinger, and 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.

[Arapinis et al., 2012] Myrto Arapinis, Sergiu Bursuc, and Mark Dermot Ryan. Reduction of equational theories for verification of trace equiv-

alence: Re-encryption, associativity and commutativity. In Pierpaolo Degano and Joshua D. Guttman, editors, *POST*, volume 7215 of *Lecture Notes in Computer Science*, pages 169–188. Springer, 2012. ISBN 978-3-642-28640-7.

[Armando et al., 2005] Alessandro Armando, David A. Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, Paul Hankes Drielsma, Pierre-Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, Sebastian Mödersheim, David von Oheimb, Michaël Rusinowitch, Judson Santiago, Mathieu Turuani, Luca Viganò, and Laurent Vigneron. The AVISPA tool for the automated validation of internet security protocols and applications. In *CAV*, pages 281–285, 2005.

[Armando et al., 2014] Alessandro Armando, Roberto Carbone, and Luca Compagna. Satmc: A SAT-based model checker for security-critical systems. In Erika Ábrahám and Klaus Havelund, editors, *TACAS*, volume 8413 of *Lecture Notes in Computer Science*, pages 31–45. Springer, 2014. ISBN 978-3-642-54861-1.

[Arora and Turuani, 2009] Charu Arora and Mathieu Turuani. Validating integrity for the ephemerizer's protocol with cl-atse. In Véronique Cortier, Claude Kirchner, Mitsuhiro Okada, and Hideki Sakurada, editors, *Formal to Practical Security*, volume 5458 of *Lecture Notes in Computer Science*, pages 21–32. Springer, 2009. ISBN 978-3-642-02001-8.

[Baader and Schulz, 1992] Franz Baader and Klaus U. Schulz. Unification in the union of disjoint equational theories: Combining decision procedures. In Deepak Kapur, editor, *CADE*, volume 607 of *Lecture Notes in Computer Science*, pages 50–65. Springer, 1992. ISBN 3-540-55602-8.

[Baader and Snyder, 2001] Franz Baader and Wayne Snyder. Unification theory. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 445–532. Elsevier and MIT Press, 2001. ISBN 0-444-50813-9, 0-262-18223-8.

[Barthe et al., 2013] Gilles Barthe, Juan Manuel Crespo, Benjamin Grégoire, César Kunz, Yassine Lakhnech, Benedikt Schmidt, and San-

tiago Zanella Béguelin. Fully automated analysis of padding-based encryption in the computational model. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM Conference on Computer and Communications Security*, pages 1247–1260. ACM, 2013. ISBN 978-1-4503-2477-9.

[Basin et al., 2005] D. Basin, S. Mödersheim, and L. Viganò. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 4(3):181–208, 2005.

[Basin et al., 2013] David Basin, Cas Cremers, and Catherine Meadows. Model checking security protocols, 2013. To appear in Handbook of Model Checking, Springer, available at `http://www.cs.ox.ac.uk/people/cas.cremers/publications/`.

[Basin et al., 2003] David A. Basin, Sebastian Mödersheim, and Luca Viganò. An on-the-fly model-checker for security protocol analysis. In Einar Snekkenes and Dieter Gollmann, editors, *ESORICS*, volume 2808 of *Lecture Notes in Computer Science*, pages 253–270. Springer, 2003. ISBN 3-540-20300-1.

[Baudet, 2005] Mathieu Baudet. Deciding security of protocols against off-line guessing attacks. In Vijay Atluri, Catherine Meadows, and Ari Juels, editors, *ACM Conference on Computer and Communications Security*, pages 16–25. ACM, 2005. ISBN 1-59593-226-7.

[Baudet et al., 2013] Mathieu Baudet, Véronique Cortier, and Stéphanie Delaune. YAPA: A generic tool for computing intruder knowledge. *ACM Trans. Comput. Log.*, 14(1):4, 2013.

[Bellovin and Merritt, 1992] S. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Proceedings of the 1992 Symposium on Research in Security and Privacy*, pages 72–84. IEEE, 1992.

[Blanchet, 2001] Bruno Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14)*, pages 82–96, Cape Breton, Nova Scotia, Canada, June 2001. IEEE Computer Society.

[Blanchet et al., 2008] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated verification of selected equivalences for security protocols. *J. Log. Algebr. Program.*, 75(1):3–51, 2008.

[Boichut et al., 2004] Y. Boichut, P.-C. Héam, O. Kouchnarenko, and F. Oehl. Improvements on the Genet and Klay technique to automatically verify security protocols. In *Proceedings of Automated Verification of Infinite States Systems (AVIS'04)*. ENTCS, 2004.

[Bursuc and Comon-Lundh, 2009] Sergiu Bursuc and Hubert Comon-Lundh. Protocol security and algebraic properties: Decision results for a bounded number of sessions. In Ralf Treinen, editor, *RTA*, volume 5595 of *Lecture Notes in Computer Science*, pages 133–147. Springer, 2009. ISBN 978-3-642-02347-7.

[Canetti et al., 2002] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, pages 494–503, 2002.

[Capkun and Hubaux, 2006] S. Capkun and J. P. Hubaux. Secure positioning in wireless networks. *IEEE Journal on Selected Areas in Communication*, 24(2), February 2006.

[Cervesato et al., 2005] I. Cervesato, C. Meadows, and D. Pavlovic. An encapsulated authentication logic for reasoning about key establishment protocols. In *IEEE Computer Security Foundations Workshop, 2005*, 2005.

[Chadha et al., 2012] Rohit Chadha, Ştefan Ciobâcă, and Steve Kremer. Automated verification of equivalence properties of cryptographic protocols. In Helmut Seidl, editor, *ESOP*, volume 7211 of *Lecture Notes in Computer Science*, pages 108–127. Springer, 2012. ISBN 978-3-642-28868-5.

[Cheval et al., 2010] Vincent Cheval, Hubert Comon-Lundh, and Stéphanie Delaune. Automating security analysis: symbolic equivalence of constraint systems. In Jürgen Giesl and Reiner Haehnle, editors, *Proceedings of the 5th International Joint Conference on Automated Reasoning (IJCAR'10)*, volume 6173 of *Lecture Notes in Artificial Intelligence*, pages 412–426, Edinburgh, Scotland, UK, July

2010. Springer-Verlag. doi: 10.1007/978-3-642-14203-1_35. URL http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/CCD-ijcar10.pdf.

[Cheval et al., 2011] Vincent Cheval, Hubert Comon-Lundh, and Stéphanie Delaune. Trace equivalence decision: negative tests and non-determinism. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *ACM Conference on Computer and Communications Security*, pages 321–330. ACM, 2011. ISBN 978-1-4503-0948-6.

[Cheval et al., 2013] Vincent Cheval, Véronique Cortier, and Antoine Plet. Lengths may break privacy - or how to check for equivalences with length. In *CAV*, pages 708–723, 2013.

[Chevalier and Rusinowitch, 2008] Yannick Chevalier and Michaël Rusinowitch. Hierarchical combination of intruder theories. *Inf. Comput.*, 206(2-4):352–377, 2008.

[Chevalier et al., 2007] Yannick Chevalier, Denis Lugiez, and Michaël Rusinowitch. Verifying cryptographic protocols with subterms constraints. In *LPAR*, LNCS vol. 4790, pages 181–195. Springer, 2007.

[Chevalier et al., 2008] Yannick Chevalier, Ralf Küsters, Michaël Rusinowitch, and Mathieu Turuani. Complexity results for security protocols with Diffie-Hellman exponentiation and commuting public key encryption. *ACM Trans. Comput. Log.*, 9(4), 2008.

[Clarke et al., 2000] Edmund M. Clarke, Somesh Jha, and Wilfredo R. Marrero. Verifying security protocols with brutus. *ACM Trans. Softw. Eng. Methodol.*, 9(4):443–487, 2000.

[Clarkson and Schneider, 2010] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010.

[Clavel et al., 2007] M. Clavel, F. Durán, S. Eker, J. Meseguer, P. Lincoln, N. Martí-Oliet, and C. Talcott. *All About Maude – A High-Performance Logical Framework*. Springer LNCS Vol. 4350, 2007.

[Comon-Lundh and Delaune, 2005] Hubert Comon-Lundh and Stéphanie Delaune. The finite variant property: How to get rid

of some algebraic properties. In Jürgen Giesl, editor, *Term Rewriting and Applications, 16th International Conference, RTA 2005, Nara, Japan, April 19-21, 2005, Proceedings*, volume 3467 of *Lecture Notes in Computer Science*, pages 294–307. Springer, 2005.

[Comon-Lundh and Shmatikov, 2003] Hubert Comon-Lundh and Vitaly Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *LICS*, pages 271–. IEEE Computer Society, 2003. ISBN 0-7695-1884-2.

[Comon-Lundh et al., 2011] Hubert Comon-Lundh, Stéphanie Delaune, and Jonathan Millen. Constraint solving techniques and enriching the model with equational theories. In Véronique Cortier and Steve Kremer, editors, *Formal Models and Techniques for Analyzing Security Protocols*, volume 5 of *Cryptology and Information Security Series*, pages 35–61. IOS Press, 2011. URL `http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/CDM-fmtasp11.pdf`.

[Cortier and Delaune, 2009a] Véronique Cortier and Stéphanie Delaune. Safely composing security protocols. *Formal Methods in System Design*, 34(1):1–36, 2009a.

[Cortier and Delaune, 2009b] Véronique Cortier and Stéphanie Delaune. A method for proving observational equivalence. In *CSF*, pages 266–276, 2009b.

[Cremers, 2006] C. J. F. Cremers. *Scyther - Semantics and Verification of Security Protocols*. PhD thesis, Eindhoven University of Technology, 2006.

[Cremers, 2008a] Cas J. F. Cremers. The Scyther tool: Verification, falsification, and analysis of security protocols. In *CAV*, pages 414–418, 2008a.

[Cremers, 2008b] Cas J. F. Cremers. Unbounded verification, falsification, and characterization of security protocols by pattern refinement. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM Conference on Computer and Communications Security*, pages 119–128. ACM, 2008b. ISBN 978-1-59593-810-7.

[Cremers et al., 2012] Cas J. F. Cremers, Kasper Bonne Rasmussen, and Srdjan Capkun. Distance hijacking attacks on distance bounding protocols. In *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*. The Internet Society, 2012. URL `http://www.internetsociety.org/distance-hijacking-attacks-distance-bounding-protocols`.

[Ştefan Ciobâcă and Cortier, 2010] Ştefan Ciobâcă and Véronique Cortier. Protocol composition for arbitrary primitives. In *CSF*, pages 322–336. IEEE Computer Society, 2010. ISBN 978-0-7695-4082-5.

[Ştefan Ciobâcă et al., 2012] Ştefan Ciobâcă, Stéphanie Delaune, and Steve Kremer. Computing knowledge in security protocols under convergent equational theories. *J. Autom. Reasoning*, 48(2):219–262, 2012.

[Datta et al., 2003] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. Secure protocol composition. In *Proc. Mathematical Foundations of Programming Semantics*, volume 83 of *Electronic Notes in Theoretical Computer Science*, 2003.

[Denning and Sacco, 1981] Dorothy E. Denning and Giovanni Maria Sacco. Timestamps in key distribution protocols. *Commun. ACM*, 24 (8):533–536, 1981.

[Desmedt, 1988] 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 et al., 2007] S. Doghim, J. Guttman, and F. J. Thayer. Searching for Shapes in Cryptographic Protocols. In *TACAS 2007*. Springer LNCS 4424, March 2007.

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

[Durán and Meseguer, 2010] Francisco Durán and José Meseguer. A Maude coherence checker tool for conditional order-sorted rewrite theories. In Peter Csaba Ölveczky, editor, *WRLA*, volume 6381 of *Lecture Notes in Computer Science*, pages 86–103. Springer, 2010. ISBN 978-3-642-16309-8.

[Durgin et al., 2001] N. Durgin, J. Mitchell, and 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.

[Durgin et al., 2004] N.A. Durgin, P.D. Lincoln, J.C. Mitchell, and A. Scedrov. Multiset rewriting and the complexity of bounded security. *Journal of Computer Security*, pages 677–722, 2004.

[Erbatur et al., 2012] Serdar Erbatur, Santiago Escobar, Deepak Kapur, Zhiqiang Liu, Christopher Lynch, Catherine Meadows, José Meseguer, Paliath Narendran, Sonia Santiago, and Ralf Sasse. Effective symbolic protocol analysis via equational irreducibility conditions. In Sara Foresti, Moti Yung, and Fabio Martinelli, editors, *ESORICS*, volume 7459 of *Lecture Notes in Computer Science*, pages 73–90. Springer, 2012. ISBN 978-3-642-33166-4.

[Erbatur et al., 2013] Serdar Erbatur, Santiago Escobar, Deepak Kapur, Zhiqiang Liu, Christopher Lynch, Catherine Meadows, José Meseguer, Paliath Narendran, Sonia Santiago, and Ralf Sasse. Asymmetric unification: A new unification paradigm for cryptographic protocol analysis. In Maria Paola Bonacina, editor, *CADE*, volume 7898 of *Lecture Notes in Computer Science*, pages 231–248. Springer, 2013. ISBN 978-3-642-38573-5.

[Escobar and Meseguer, 2007] Santiago Escobar and José Meseguer. Symbolic model checking of infinite-state systems using narrowing. In *Proceedings of the 18th International Conference on Rewriting Techniques and Applications (RTA'07)*, volume 4533 of *Lecture Notes in Computer Science*, pages 153–168, 2007.

[Escobar et al., 2006] Santiago Escobar, Catherine Meadows, and José Meseguer. A rewriting-based inference system for the NRL Protocol

Analyzer and its meta-logical properties. *Theor. Comput. Sci.*, 367(1-2):162–202, 2006. doi: 10.1016/j.tcs.2006.08.035. URL `http://dx.doi.org/10.1016/j.tcs.2006.08.035`.

[Escobar et al., 2008] Santiago Escobar, Catherine Meadows, and José Meseguer. State space reduction in the Maude-NRL Protocol Analyzer. In Sushil Jajodia and Javier López, editors, *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 et al., 2009a] Santiago Escobar, Catherine Meadows, and José Meseguer. Maude-NPA: Cryptographic protocol analysis modulo equational properties. In *Foundations of Security Analysis and Design V, FOSAD 2007/2008/2009 Tutorial Lectures*, volume 5705 of *Lecture Notes in Computer Science*, pages 1–50. Springer, 2009a.

[Escobar et al., 2009b] Santiago Escobar, Catherine Meadows, and Jose Meseguer. Maude-NPA manual version 2.0. `http://maude.cs.uiuc.edu/tools/Maude-NPA/`, 2009b.

[Escobar et al., 2010] Santiago Escobar, Catherine Meadows, José Meseguer, and Sonia Santiago. Sequential protocol composition in Maude-NPA. In Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou, editors, *ESORICS*, volume 6345 of *Lecture Notes in Computer Science*, pages 303–318. Springer, 2010. ISBN 978-3-642-15496-6.

[Escobar et al., 2011] Santiago Escobar, Deepak Kapur, Christopher Lynch, Catherine Meadows, José Meseguer, Paliath Narendran, and Ralf Sasse. Protocol analysis in Maude-NPA using unification modulo homomorphic encryption. In *PPDP*, pages 65–76, 2011.

[Escobar et al., 2012a] Santiago Escobar, Catherine Meadows, and José Meseguer. *Maude-NPA, version 2.0.* University of Illinois at Urbana-Champaign, 2012a. Available at `http://maude.cs.uiuc.edu/tools/Maude-NPA`.

[Escobar et al., 2012b] Santiago Escobar, Ralf Sasse, and José Meseguer. Folding variant narrowing and optimal variant termination. *J. Log. Algebr. Program.*, 81(7-8):898–928, 2012b.

[Escobar et al., 2014a] Santiago Escobar, Catherine Meadows, José Meseguer, and Sonia Santiago. State space reduction in the Maude-NRL Protocol Analyzer. *Inf. Comput.*, 238:157–186, 2014a. doi: 10.1016/j.ic.2014.07.007. URL `http://dx.doi.org/10.1016/j.ic.2014.07.007`.

[Escobar et al., 2014b] Santiago Escobar, Catherine Meadows, José Meseguer, and Sonia Santiago. A rewriting-based forwards semantics for Maude-NPA. In *proc. I Symposium and Bootcamp on the Science of Security (HotSoS 2014). IEEE Digital Library*, 2014b. To appear.

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

[Gong and Syverson, 1998] L. Gong and P. Syverson. Fail-stop protocols: An approach to designing secure protocols. In R. K. Iyer, M. Morganti, W. K. Fuchs, and V. Gligor, editors, *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.

[Groß and Mödersheim, 2011] Thomas Groß and Sebastian Mödersheim. Vertical protocol composition. In *CSF*, pages 235–250. IEEE Computer Society, 2011. ISBN 978-1-61284-644-6.

[Gutiérrez et al., 2012] Raúl Gutiérrez, José Meseguer, and Camilo Rocha. Order-sorted equality enrichments modulo axioms. In Franciso Durán, editor, *WRLA*, volume 7571 of *Lecture Notes in Computer Science*, pages 162–181. Springer, 2012. ISBN 978-3-642-34004-8.

[Guttman, 2001] J. Guttman. Security protocol design via authentication tests. In *Proc. Computer Security Foundations Workshop*. IEEE Computer Society Press, 2001.

[Guttman et al., 2008] J. D. Guttman, J. C. Herzog, V. Swarup, and 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/`.

[Guttman and Thayer, 2000] Joshua D. Guttman and F. Javier Thayer. Protocol independence through disjoint encryption. In *CSFW*, pages 24–34. IEEE Computer Society, 2000. ISBN 0-7695-0671-2.

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

[Huima, 1999] Antti Huima. Efficient infinite-state analysis of security protocols. In *Proc. FLOCÕ99 Workshop on Formal Methods and Security Protocols (FMSPÕ99)*, 1999.

[Jouannaud and Kirchner, 1986] Jean-Pierre Jouannaud and Hélène Kirchner. Completion of a set of rules modulo a set of equations. *SIAM J. Comput.*, 15(4):1155–1194, 1986.

[Kremer and Ryan, 2005] Steve Kremer and Mark D. Ryan. Analysing the vulnerability of protocols to produce known-pair and chosen-text attacks. In Riccardo Focardi and Gianluigi Zavattaro, editors, *Proceedings of the 2nd International Workshop on Security Issues in Coordination Models, Languages and Systems (SecCo'04)*, Electronic Notes in Theoretical Computer Science, pages 84–107, London, UK, May 2005. Elsevier Science Publishers. doi: 10.1016/j.entcs.2004.11.043. URL `http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/Kremer-secco04.pdf`.

[Küsters and Truderung, 2009] Ralf Küsters and Tomasz Truderung. Using ProVerif to Analyze Protocols with Diffie-Hellman Exponentiation. In *Proceedings of the 22nd IEEE Computer Security Foundations Symposium, CSF 2009, Port Jefferson, New York, USA, July 8-10, 2009*, pages 157–171, 2009. doi: 10.1109/CSF.2009.17. URL `http://doi.ieeecomputersociety.org/10.1109/CSF.2009.17`.

[Küsters and Truderung, 2011] Ralf Küsters and Tomasz Truderung. Reducing protocol analysis with XOR to the XOR-free case in the Horn theory based approach. *J. Autom. Reasoning*, 46(3-4):325–352, 2011. doi: 10.1007/s10817-010-9188-8. URL `http://dx.doi.org/10.1007/s10817-010-9188-8`.

[Liu, 2012] Zhiqiang Liu. *Dealing Efficiently with Exclusive OR, Abelian Groups and Homomorphism in Cryptographic Protocol Analysis*. PhD thesis, Clarkson University, 2012.

[Liu and Lynch, 2011] Zhiqiang Liu and Christopher Lynch. Efficient general unification for xor with homomorphism. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *CADE*, volume 6803 of *Lecture Notes in Computer Science*, pages 407–421. Springer, 2011. ISBN 978-3-642-22437-9.

[Lowe, 1996] 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.

[Lowe, 2004] Gavin Lowe. Analysing protocol subject to guessing attacks. *Journal of Computer Security*, 12(1):83–98, 2004.

[Lowe and Roscoe, 1997] Gavin Lowe and A. W. Roscoe. Using CSP to detect errors in the TMN protocol. *IEEE Trans. Software Eng.*, 23(10): 659–669, 1997.

[Lucas, 1998] S. Lucas. Context-sensitive computations in functional and functional logic programs. *J. Functl. and Log. Progr.*, 1(4):446–453, 1998.

[Lynch and Meadows, 2005] Christopher Lynch and Catherine Meadows. On the relative soundness of the free algebra model for public key encryption. *Electr. Notes Theor. Comput. Sci.*, 125(1):43–54, 2005.

[Meadows, 1996a] C. Meadows. The NRL Protocol Analyzer: An overview. *Journal of logic programming*, 26(2):113–131, 1996a.

[Meadows, 1996b] Catherine Meadows. Language generation and verification in the NRL Protocol Analyzer. In *Ninth IEEE Computer Security Foundations Workshop, March 10 - 12, 1996, Dromquinna Manor, Kenmare, County Kerry, Ireland*, pages 48–61. IEEE Computer Society, 1996b.

[Meier et al., 2013] Simon Meier, Benedikt Schmidt, Cas Cremers, and David A. Basin. The Tamarin prover for the symbolic analysis of security protocols. In Natasha Sharygina and Helmut Veith, editors, *CAV*, volume 8044 of *Lecture Notes in Computer Science*, pages 696–701. Springer, 2013. ISBN 978-3-642-39798-1.

[Merritt, 1984] Michael Merritt. *Cryptographic Protocols*. PhD thesis, Georgia Institute of Technology, 1984.

[Meseguer, 1992] J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.

[Meseguer, 1997] José Meseguer. Membership algebra as a logical framework for equational specification. In Francesco Parisi-Presicce, editor, *WADT*, volume 1376 of *Lecture Notes in Computer Science*, pages 18–61. Springer, 1997. ISBN 3-540-64299-4.

[Meseguer and Thati, 2007] José Meseguer and Prasanna Thati. Symbolic reachability analysis using narrowing and its application to verification of cryptographic protocols. *Higher-Order and Symbolic Computation*, 20(1-2):123–160, 2007.

[Millen, 2003] Jonathan K. Millen. On the freedom of decryption. *Inf. Process. Lett.*, 86(6):329–333, 2003.

[Millen and Shmatikov, 2001] Jonathan K. Millen and Vitaly Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In Michael K. Reiter and Pierangela Samarati, editors, *ACM Conference on Computer and Communications Security*, pages 166–175. ACM, 2001. ISBN 1-58113-385-5.

[Millen et al., 1987] Jonathan K. Millen, Sidney C. Clark, and Sheryl B. Freedman. The Interrogator: Protocol security analysis. *IEEE Trans. Software Eng.*, 13(2):274–288, 1987.

[Mitchell et al., 1997] J. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using Murphi. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 1997.

[Mödersheim, 2007] Sebastian Mödersheim. *Models and methods for the automated analysis of security protocols*. PhD thesis, ETH Zurich, 2007.

[Mödersheim and Viganò, 2009] Sebastian Mödersheim and Luca Viganò. The open-source fixed-point model checker for symbolic analysis of security protocols. In Alessandro Aldini, Gilles Barthe, and Roberto

Gorrieri, editors, *FOSAD*, volume 5705 of *Lecture Notes in Computer Science*, pages 166–194. Springer, 2009. ISBN 978-3-642-03828-0.

[Mödersheim et al., 2010] Sebastian Mödersheim, Luca Viganò, and David A. Basin. Constraint differentiation: Search-space reduction for the constraint-based analysis of security protocols. *Journal of Computer Security*, 18(4):575–618, 2010.

[Needham and Schroeder, 1978] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, 1978.

[Newcomb and Lowe, 2005] Tom Newcomb and Gavin Lowe. A computational justification for guessing attack formalisms. Technical report, Oxford University Computing Laboratory, 2005.

[Paulson, 1998] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1-2): 85–128, 1998.

[Rusinowitch and Turuani, 2001] Michael Rusinowitch and Mathieu Turuani. Protocol insecurity with a finite number of sessions and composed keys is NP-complete. In 14^{th} *IEEE Computer Security Foundations Workshop*, pages 174–190, 2001.

[Ryan and Schneider, 1998] Peter Y. A. Ryan and Steve A. Schneider. An attack on a recursive authentication protocol. a cautionary tale. *Inf. Process. Lett.*, 65(1):7–10, 1998.

[Santiago et al., 2014a] Sonia Santiago, Santiago Escobar, Catherine Meadows, and José Meseguer. Sequential protocol composition in Maude-NPA. *Journal of Computer Security*, 2014a. Under review.

[Santiago et al., 2014b] Sonia Santiago, Santiago Escobar, Catherine Meadows, and José Meseguer. A formal definition of protocol indistinguishability and its verification using maude-npa. In Sjouke Mauw and Christian Damsgaard Jensen, editors, *Security and Trust Management - 10th International Workshop, STM 2014, Wroclaw, Poland, September 10-11, 2014. Proceedings*, volume 8743 of *Lecture Notes in Computer Science*, pages 162–177. Springer, 2014b. ISBN 978-3-319-11850-5.

[Sasse et al., 2010] Ralf Sasse, Santiago Escobar, Catherine Meadows, and José Meseguer. Protocol analysis modulo combination of theories: A case study in Maude-NPA. In Jorge Cuéllar, Javier Lopez, Gilles Barthe, and Alexander Pretschner, editors, *STM*, volume 6710 of *Lecture Notes in Computer Science*, pages 163–178. Springer, 2010. ISBN 978-3-642-22443-0.

[Schmidt-Schauß, 1989] Manfred Schmidt-Schauß. Unification in a combination of arbitrary disjoint equational theories. *J. Symb. Comput.*, 8 (1/2):51–99, 1989.

[Shmatikov and Stern, 1998] V. Shmatikov and U. Stern. Efficient finite-state analysis for large security protocols. In *11th Computer Security Foundations Workshop — CSFW-11*. IEEE Computer Society Press, 1998.

[Stubblebine and Meadows, 2000] Stuart Stubblebine and Catherine Meadows. Formal characterization and automated analysis of known-pair and chosen-text attacks. *IEEE Journal on Selected Areas in Communications*, 18(4):571–581, 2000.

[Tatebayashi et al., 1990] Makoto Tatebayashi, Natsume Matsuzaki, and David Newman. Key distribution protocol for digital mobile communication systems. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO'89 Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 324–334. Springer Berlin / Heidelberg, 1990. ISBN 978-0-387-97317-3. URL `http://dx.doi.org/10.1007/0-387-34805-0_30`.

[TeReSe, 2003] TeReSe, editor. *Term Rewriting Systems*. Cambridge University Press, Cambridge, 2003.

[Thati and Meseguer, 2007] P. Thati and J. Meseguer. Symbolic reachability analysis using narrowing and its application verification of cryptographic protocols. *J. Higher-Order and Symbolic Computation*, 20(1–2): 123–160, 2007.

[Turuani, 2006] Mathieu Turuani. The CL-Atse protocol analyser. In Frank Pfenning, editor, *RTA*, volume 4098 of *Lecture Notes in Computer Science*, pages 277–286. Springer, 2006. ISBN 3-540-36834-5.

[Viry, 2002] Patrick Viry. Equational rules for rewriting logic. *Theor. Comput. Sci.*, 285(2):487–517, 2002.

[Weidenbach, 1999] Christoph Weidenbach. Towards an automatic analysis of security protocols in first-order logic. In Harald Ganzinger, editor, *CADE*, volume 1632 of *Lecture Notes in Computer Science*, pages 314–328. Springer, 1999. ISBN 3-540-66222-7.

[Yang et al., 2014] Fan Yang, Santiago Escobar, Catherine Meadows, José Meseguer, and Paliath Narendran. Theories of homomorphic encryption, unification, and the finite variant property. In *proc. PPDP*, 2014. to appear.

[Zhang and Remy, 1985] Hantao Zhang and Jean-Luc Remy. Contextual rewriting. In *RTA*, pages 46–62, 1985.