FRANÇOIS GAGNON

A CLASS OF THEORY-DECIDABLE INFERENCE SYSTEMS

Toward a Decision Procedure for Structured Cryptographic Protocols

Mémoire présenté
à la Faculté des études supérieures de l'Université Laval
dans le cadre du programme de maîtrise en informatique
pour l'obtention du grade de Maître ès sciences, (M.Sc.)

FACULTÉ DES SCIENCES ET DE GÉNIE
UNIVERSITÉ LAVAL
QUÉBEC

FÉVRIER 2005

Résumé

Dans les deux dernières décennies, l'Internet a apporté une nouvelle dimension aux communications. Il est maintenant possible de communiquer avec n'importe qui, n'importe où, n'importe quand et ce, en quelques secondes. Alors que certains systèmes de communication distribués, comme le courriel, le chat, ..., sont plutôt informels et ne nécessitent aucune sécurité, d'autres comme l'échange d'informations militaires ou encore médicales, le commerce électronique, ..., sont très formels et nécessitent de très hauts niveaux de sécurité.

Pour atteindre les objectifs de sécurité voulus, les protocoles cryptographiques sont souvent utilisés. Cependant, la création et l'analyse de ces protocoles sont très difficiles. Certains protocoles ont été montrés incorrects plusieurs années après leur conception. Nous savons maintenant que les méthodes formelles sont le seul espoir pour avoir des protocoles parfaitement corrects. Ce travail est une contribution dans le domaine de l'analyse des protocoles cryptographiques de la façon suivante:

- Une classification des méthodes formelles utilisées pour l'analyse des protocoles cryptographiques.
- L'utilisation des systèmes d'inférence pour la modélisation des protocoles cryptographiques.
- La définition d'une classe de systèmes d'inférence qui ont une theorie décidable.
- La proposition d'une procédure de décision pour une grande classe de protocoles cryptographiques

Abstract

In the last two decades, Internet brought a new dimension to communications. It is now possible to communicate with anyone, anywhere at anytime in few seconds. While some distributed communications, like e-mail, chat, ..., are rather informal and require no security at all, others, like military or medical information exchange, electronic-commerce, ..., are highly formal and require a quite strong security.

To achieve security goals in distributed communications, it is common to use cryptographic protocols. However, the informal design and analysis of such protocols are error-prone. Some protocols were shown to be deficient many years after their conception. It is now well known that formal methods are the only hope of designing completely secure cryptographic protocols. This thesis is a contribution in the field of cryptographic protocols analysis in the following way:

- A classification of the formal methods used in cryptographic protocols analysis.
- The use of inference systems to model cryptographic protocols.
- The definition of a class of theory-decidable inference systems.
- The proposition of a decision procedure for a wide class of cryptographic protocols.

Avant-propos

I'd first like to thank my research director, Mohamed Mejri, for the last two years we spent together to obtain the results included in this thesis. Thank you for your guidance, for sharing with me a part of your passion and wisdom. Thank you for letting me explore interesting topics that were not directly related with my research area but have nevertheless been very enriching. Thank you for the many opportunities you provided me, Ottawa'2003 and Italy'2004 among many others.

Special thanks should go to my parents for without their unconditional support, nothing of this would have been possible. To my father with whom I played many chess games; to my mother who showed me the most important thing in life: "*never give up, always fight*"; to my sister who traced my way to université Laval; to my brother who helped me pass into manhood, to Denis with whom I spent so much of my time and with whom I had so much fun, thank you very much!

Thanks to my girlfriend for being there when I needed you the most, for sharing my dreams, for your encouragement, for your support and most of all for loving me. Thanks to all her family for opening me their home.

Among my colleagues at université Laval, I want to thank Claude Bolduc, Jean-Phillipe Gagnon, Hanane Houmani, Eric Lacoursière, Majoub Langar, Vincent Mathieu, Mohamed Mbarki, Jean-François Morneau and Maxime Morneau for many fruitful discussions. However, one deserve special acknowledgement for being a model in research as well as in teaching; for so many discussion about school, life, research, for helping me discover what I really wanted to do and how I should do it, for all this and so much more, thank you Hans Bherer.

I am grateful to my reviewers Dr. Nadia Tawbi and Dr. Kamel Adi who did a great job finding some mistakes as well as some parts that needed to be rewritten.

To my parents,
without whom nothing of this would have been possible.
To Catherine,
without whom this would have been much harder.
To Denis,
for it all started with our dreams back then.

Nothing great has ever been
accomplished without passion.

Contents

List of Tables

List of Figures

Introduction

Nowadays, it is essential to provide some security mechanisms in distributed systems. Many daily utilities rely upon either confidentiality, data integrity, authentication, ..., to achieve common tasks. Whether it be at a cash dispenser, when buying a book online with a credit card or while transferring a file from work to home, a user is entitled to be assured of a minimum of security. However, even a little security objective can amount a great deal of efforts. Many security mechanisms rely upon cryptographic protocols, but are those security protocols correct? What if there is a flaw no one knows about? And if someone knows about this flaw, what can he do?

Many formal methods have been used for the analysis of cryptographic protocols; some were even developed especially for this purpose. Among these, many are based on logics: translating a protocol as well as security properties into formulae, and trying to check if the properties are violated or if they hold. Model checking is also considered for the analysis of cryptographic protocols, however it often suffers from states space explosion. Process algebra is a recent addition to cryptographic protocol analysis and it seems very promising.

Most of the formal methods used for the analysis of cryptographic protocols follow this pattern: first, a mathematical model is used to represent a cryptographic protocol and another to represent security properties. Then, they try to check whether or not the modeled cryptographic protocol satisfies the modeled security properties. Formal methods used for the analysis of cryptographic protocols can usually be classified in three groups: the first one consists of methods trying to prove that a protocol is correct (that it respects a security requirement); the second group comprise the methods trying to prove that a protocol is not correct (that it does not respect a security requirement); and the last one is composed of methods aiming to decide whether a protocol is correct or not. Since establishing the correctness of a cryptographic protocol is undecidable, in the general case, very few formal methods belong to the last group.

A possible model for a cryptographic protocol (and widely used to model many other

problems) is an inference system, where security properties are modeled as formulae to be proven using the inference system. It is already known that inference systems are good models for cryptographic protocols; good because they form a good representation of the behavior of a protocol and because extracting an inference system from a given cryptographic protocol can be fully automatic.

In general, inference systems suffer from a decidability problem. That is, it is not possible, in general, to decide if a given well-formed formula is provable in a given inference system. As a consequence of this problem, it is not yet possible to decide the correctness of a protocol using an inference system model.

In this thesis, we address these two problems: the decidability of basic security properties for a class of cryptographic protocols and the decidability of a class of inference systems.

This thesis is structured as follows: The first chapter is an introduction to cryptography and cryptographic protocols. It is intended for a beginner in this subject and might be skipped entirely (except for Section 1.4 where our notation is introduced) by an expert reader. Chapter 2 provides a classification of some well known formal methods used for the analysis of cryptographic protocols. Its purpose is two-fold: on one hand, it serves as a state of the art for protocol analysis, and on the other hand it helps to situate this thesis among other similar works. Chapter 3 is an introduction to inference systems and their decidability problem. It also presents a process transforming an inference system into an equivalent, but decidable[1], one. Chapter 4 addresses the major problem of the transformation process presented in Chapter 3, non-termination. A class of inference system will be defined such that the transformation process is proved to always terminate on them. The last chapter uses the transformation process to build a decision procedure for the correctness of a class of cryptographic protocols. Finally, a comparison between our results and some similar ones as well as a discussion about possible future work will serve as a conclusion.

Chapter 4 and 5 hold the results of this thesis. As far as we know, these results are new.

[1]The result of the transformation will not automatically be a decidable inference system, but with some care it will.

Chapter 1

Cryptographic Protocols

Abstract

This chapter aims to introduce the basic concepts of cryptography, cryptographic protocols, security properties and flaws that could arise in cryptographic protocols. A notation about cryptographic protocols that will be used throughout this thesis will also be presented.

1.1 Introduction

The extremely fast rise of Internet and distributed systems in the last decade brought the opportunity to exchange information with anyone, anywhere at anytime. While some kinds of information exchange require a very low level of security, others can only be used if one can guarantee a very secure exchange. The latter comprise, among other things, electronic commerce, medical information exchanges and military conversations. To achieve a basic security goal on a basic task, one uses cryptography, but to ensure that more elaborate security properties are not violated during a complex process, one needs to use cryptographic protocols.

The following sections give an introduction to the world of cryptography and cryptographic protocols. The rest of this chapter is structured as follows: Section 1.2 introduces basic concepts in cryptography; Section 1.3 describes some common security properties often required; in Section 1.4 we develop the formalism used to represent cryptographic protocols as well as some important concepts and finally Section 1.5

shows some examples of flaws in cryptographic protocols.

1.2 Cryptography

Cryptology, which comes from Greek *kryptos* "hidden" and *logos* "word", is considered as the science studying principles and techniques used to transform information into an unintelligible form that is difficult for an unauthorized person to read, while still allowing the intended reader to recover the original message. Cryptography, together with cryptanalysis, form the two opposite branches of cryptology. On one hand, cryptographers try to develop new encryption/decryption techniques based on mathematics, linguistics, physics, etc... to assure that messages can be exchanged in a secret way. On the other hand, cryptanalysts aim to read encrypted messages when they are not authorized to do so. Since cryptography is of limited importance in this thesis, we address only some technical points here. The reader interested in a deeper history of cryptology is referred to [62, 79].

1.2.1 Cryptosystem

Informally, a cryptographic system (or cryptosystem for short) is a pair of functions: one used to encrypt messages and the other to decrypt encrypted messages, both using a key. More formally, in [21], *Denning* defines a cryptosystem with five components ($\mathcal{M}$, $\mathcal{C}$, $\mathcal{K}$, E,D) and some requirements on the behavior of these components:

- A set of messages $\mathcal{M}$ (clear text space).
- A set of encrypted messages $\mathcal{C}$ (cipher text space).
- A set of keys $\mathcal{K}$.
- An encryption function $E : \mathcal{M} \times \mathcal{K} \rightarrow \mathcal{C}$.
- A decryption function $D : \mathcal{C} \times \mathcal{K} \rightarrow \mathcal{M}$.
- $\forall k \in \mathcal{K}, \forall m \in \mathcal{M} : \exists m' \in \mathcal{C}$ such that $E(m, k) = m'$
- $\forall k \in \mathcal{K}, \forall m \in \mathcal{M} : E(D(m, k^{-1}), k) = D(E(m, k), k^{-1})$. Where k^{-1} is the decryption key corresponding to k.

- The security of the system should depend only on the secrecy of the keys. Algorithms E and D can be public.

1.2.2 Symmetric or asymmetric cryptosystem

Although many cryptosystems exist, they are divided in two main classes: symmetric and asymmetric cryptosystems.

Definition 1.2.1 (symmetric cryptosystem)
A symmetric (or secret key) cryptosystem is one where the encryption key k can be computed from the decryption key k^{-1} and vice versa.

Since, in secret cryptosystem, k can be computed from k^{-1} and vice versa, we often suppose that $k = k^{-1}$ and thus use the same key to both encrypt and decrypt. Examples of symmetric cryptosystem are: DES, the RCx family, LUCIFER, BLOWFISH, SAFER, ... Detailed description of these cryptosystems together with other examples can be found in [60, 76].

In a symmetric cryptosystem, when *Alice* wants to send a message m to *Bob*, they first need to share a secret key k. Then *Alice* sends $E(m, k)$ to *Bob* who can retrieve the original message by applying the decryption operator $D(E(m, k), k)$. Note that here *Bob* is assured that $D(E(m, k), k)$ and thus m indeed comes from *Alice* since only she has access to the key k. In order to exchange a secret message, they first need to exchange a secret key. The advantage here is that the key can be exchanged (in a meeting) before the need to send a secret message arises. The problem with this symmetric system is the more often a key is used, the less secure it will become. In [21], they argue that a secret key should be used just once. Thus arises the symmetric key exchange problem.

Definition 1.2.2 (Symmetric key exchange problem)
Whenever two persons want to exchange a secret message m, they must have beforehand exchanged a key that will be used specifically to encrypt m.

If the number of secret messages to send is considerable (which is the case with most secure communication such as E-commerce, military communication and so on), the number of keys shared will be considerable. Another situation where symmetric key exchange raises a problem is when there is a network of n persons where every person may need to communicate secretly with any other. In this case, every n person needs to share a key with all the other $n - 1$ persons.

To counter this key exchange problem, *Diffie* and *Hellman* proposed, in [24], what would become asymmetric cryptosystem.

Definition 1.2.3 (asymmetric cryptosystem)
An asymmetric (or public key) cryptosystem is one where the encryption key k is completely distinct from the decryption key k^{-1}. By completely distinct, we mean that they are not computable one from another.

Generally speaking, a public key cryptosystem works this way:

- *Bob* first creates a pair of keys k_B and k_B^{-1}, his public and private key respectively.
- *Bob* makes available his public key k_B to anyone from whom he expects to receive a secret message (the public key does not have to be secret at all but the private key must be known only by its owner).
- When *Alice* wants to send a secret message m to *Bob*, she encrypts her message with *Bob*'s public key forming $c = E(m, k_B)$ and sends him the resulting cipher text c.
- When *Bob* receives the cipher text c, he decrypts it with his private key using $m = D(c, k_B^{-1})$ thus recovering the original clear text m.

Note that when *Bob* receives the message c, he cannot be sure if the message really comes from *Alice*; since anyone has access to his public key k, anyone could have encrypted m with k. To address this little problem, one can do the following steps to assure the receiver of a message that it indeed comes from the claimed sender. This process is referred to as digital signature.

Definition 1.2.4 (Digital Signature)
The digital signature is used to achieve a similar goal as handwritten signature; to convince anyone about the origin of a message.

- *Alice* first creates a pair of keys k_A and k_A^{-1}, her public and private key respectively.
- *Alice* makes available her public key k_A to anyone whom she expects to send a signed message to (the public key does not have to be secret at all but the private key must be known only by its owner).
- When *Alice* wants to send a signed message m to *Bob*, she encrypts her message with her own private key forming $c = E(m, k_A^{-1})$ and sends him the resulting cipher text c.

- When *Bob* receives the cipher text c, he decrypts it using *Alice*'s public key forming $m = D(c, k_A)$ thus recovering the original clear text m.

So *Bob* will be convinced that *Alice* sent the message since only her has access to k_a^{-1}. But in this case, the message m is not secret since anyone can use k_A to decrypt c and recover m. To send a message that is both secret and signed, *Alice* can use the model of Figure 1.1.

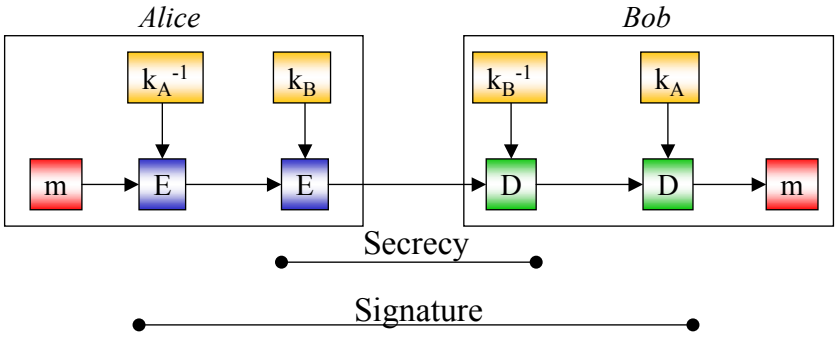

Figure 1.1: Secret and signed message

Among the public key cryptosystems, there is the famous RSA, Diffie-Hellman, MCEliece, the Knapsack problem, Rabin, ElGamal, ... Details about these and some others can be found in [60, 72, 76]. Information about a public key cryptosystem developed by *Tao Renji* and based on finite automata can be found in [70, 71]. Other information on asymmetric cryptosystem is presented in [63, 75].

One should remark here that with a public key cryptosystem, the two problems referred to as the symmetric key exchange problem of Definition 1.2.2 are no more. There is no need to keep the exchanged key secret as the encryption keys are now public and in a network of n persons, each needs to generate a single pair of keys. Unfortunately, public key cryptosystem also suffers from two problems.

Definition 1.2.5 (Small space of clear text message problem)
If the size of the clear text message is too small, an intruder can perform an exhaustive search of the clear text space to learn the value of an intended secret message.

Definition 1.2.6 (Asymmetric key exchange problem)
While retrieving Bob's public key, Alice is vulnerable to a man-in-the-middle attack. When Alice receives a public key k, which she believes to be Bob's, what if k is in fact Eve's public key?

Example 1.2.1 (Small space of clear text attack)
Suppose that the secret message m *Alice* needs to send to *Bob* is a day of the week. Assuming *Eve* knows that *Alice* will send a day of the week but doesn't know which one. Let *Bob*'s public key be k_B and the encryption function E. *Alice* sends a cipher text c that is the result of $E(m, k_B)$. *Eve* can learn the value of m in the following way. *Eve* intercepts c, she then computes $c_i = E(m_i, k_B)$ (she can do this since k_B and E are of public knowledge) where i range over 1 to 7 and $m_1 =$ Sunday, $m_2 =$ Monday, ... *Eve* simply needs to find the j such that $c_j = c$ to conclude that $m_j = m$ is the intended secret message sent from *Alice* to *Bob*.

Another major drawback of public key cryptosystems is that they are based on mathematical problems that are believed to be intractable but not yet proved so. For example, RSA is based on the belief that it is quite difficult to factorize a large number into its prime factors. In this sense, it is possible (although improbable) that someone knows a way to break RSA and exploits this knowledge to his own advantage. In fact, one of the breakthroughs in technology that could bring down RSA is the quantum computation. An algorithm that could factorize a number, provided that it is run on a quantum computer, has already been developed. This algorithm is known as *Shor's quantum algorithm for factorization*. Details of this algorithm fall way beyond the scope of this thesis but it is nevertheless a very interesting topic and can be found in [11]. In [8], we can see some progress on quantum factorization dating from 2001.

1.2.3 Future way in cryptography

While the current generation of cryptosystems is based on difficult mathematical problems, the next one could be based on physics properties. While quantum computation could bring a real threat to the security of today's public key cryptosystems, it could also provide some new, very secure, cryptosystems. In 1984, a new research field started: quantum cryptography. *Brassard & Bennet* developed one of the first quantum cryptosystem BB84. Since quantum cryptography is not mentioned in the rest of this thesis, we discuss it no further, but simply state that information about this young field of research are in [7, 8, 11].

1.2.4 Perfect cryptography assumption

In the rest of this thesis, we are interested in the analysis of cryptographic protocols. To simplify our task, we will make the assumption of the existence of a perfect cryptosystem; that is a cryptosystem that cannot be broken. Surprisingly, when dealing with perfect encryption, flaws still arise among protocols (we talk about some of these flaws in Section 1.5). Thus, even under the perfect cryptography assumption, there is still a lot to do before claiming that we are secure from any attack. The perfect encryption assumption, a part of the *Dolev-Yao* model in [25], can be formalized as follows:

Definition 1.2.7 (Perfect cryptography assumption)
In a perfect cryptosystem, only the operations that are intended by the cryptosystem are allowed to be done, i.e. to decrypt a message that is encrypted under k, one absolutely needs to know the key k^{-1}. A perfect cryptosystem satisfies the following requirements:

- *Knowing m and $E(m,k)$, it should be impossible to compute k^{-1}.*
- *Knowing $E(m,k)$, it should be impossible to compute m without knowing k^{-1}.*
- *Knowing m and $E(m,k)$, it should be impossible to compute the key k.*
- *Without knowing k, it should be impossible to compute message $E(m,k)$ for a given m.*

1.3 Security properties

When we need secure communications, secure can mean a lot of different things. To be more precise, when talking about the security of a communication system, we talk about the security properties it should achieve. Of the possible security properties, the following first four are the most common, while the last four are useful in electronic commerce. This is not intended to be an exhaustive list of security properties but simply to give some insight about what we mean by secure. Some of these security property definitions are from [3].

- **Secrecy:** This property, also known as confidentiality, allows someone to send a private message to a specific person while being sure that nobody else can get the information. For a protocol to preserve the secrecy of one of its component means that it does not leak any sensitive information about this component during its execution.

- **Authentication:** With the authentication property, the receiver of a message can establish the exact identity of the sender. In other words, each time *Bob* accepts a message as being from *Alice*, *Alice* must have sent exactly the same message a while ago.

- **Integrity:** This property ensures that a message has not been altered since its creation by the sender. This is a very important security property since there is a huge difference between a message telling your bank to transfer 1 000$ from your bank account to another and a message telling your bank to transfer 100 000$ from your bank account to another.

- **Non-repudiation:** Non-repudiation is defined as the impossibility for a person involved in a communication to deny his participation in this communication. It should be possible for *Alice* to prove to anyone else that *Bob* has participated in a communication with her, if it is really the case.

- **Chaotic:** A protocol is said to be chaotic for a given key k if someone is able to both encrypt any message m with the key k and decrypt any cipher text c that was encrypted under k without ever knowing the value of k. That is, an intruder uses the protocol as a "black-box" to break a perfect cryptosystem. This property has just been recently addressed. In [59], *Mejri* shows that some well known protocols are chaotic and explains how the chaotic property should change our view of some basic security properties.

- **Fairness:** In a fair protocol, participants require protection from each other, rather than from an external intruder. For example, in electronic contract signing, we must be able to prevent a participant to halt the process when the other has completed his part.

- **Availability:** This one states that a service should always be available when a participant wants to use that service. For example, in electronic auction, it should be possible for everyone to bid until the time is up; otherwise, those who cannot bid are disadvantaged.

- **Anonymity:** This provides the ability to make a transaction that cannot be tracked back to its source. We say that a system is anonymous over some set of events E if it has the following property. When an event from E occurs, it should not be possible for an observer to identify who caused this event. In electronic commerce, for instance, it should not be allowed that an observer, a publicity company say, be able to deduce who purchases what.

- **Money atomicity:** When a money transfer is required, it should be either entirely done or not at all. In other words, virtual money creation and virtual money destruction should not be allowed.

- **Good atomicity:** On an electronic transaction, we want to be sure that the consumer will receive the goods if and only if the merchant is paid.
- **Certified delivery:** This property, allows both the client and merchant to prove exactly what they received from the transaction (money or goods). If there is something wrong, those evidences can be shown to a judge to resolve the problem.

Although the cryptosystems discussed in Section 1.2 can address directly some security properties, we need stronger constructions to achieve others or multiple security goals at a time. These constructions will be cryptographic protocols.

1.4 Cryptographic protocols

Here we describe the basic notations of cryptographic protocols and we give some examples of such constructions. From now on, we will use $\{m\}_k$ instead of $E(m, k)$.

Definition 1.4.1 (Communication protocol)
A communication protocol is a set of rules that precisely say how the different principals of a distributed system must communicate.

We use communication protocols everyday for sending e-mail, sending ordinary mail, talking on the phone,... etc.

Example 1.4.1 (Telephone communication protocol)
One of the best examples of communication protocols is clearly the telephone protocol. When you want to call someone, you first need to get his (her) phone number. Then you compose the number on your phone and wait for the other person to answer. When you're done talking, you simply hang up the phone.

Example 1.4.2 (Availability testing communication protocol)
Suppose *Alice* wants to chat with *Bob*. First, *Alice* needs to know if *Bob* is online (available to chat). If he is, they can begin chatting. So *Alice* needs a communication protocol to test whether *Bob* is online or not. The communication protocol of Table 1.1 tries to achieve this goal. If *Alice* receives the answer from *Bob*, then she knows he's online, unless the message does not come from *Bob*. To rule this possibility out, we ask *Bob* to sing Message2 before sending it back to *Alice*. This gives us the protocol of Table 1.2. We'll see later that this last one does not work either.

$\mathcal{K}_A = \mathcal{A}gt$	
$\mathcal{K}_B = \mathcal{A}gt$	
Message1	$A \rightarrow B$: Are you online?
Message2	$B \rightarrow A$: Yes

Table 1.1: Availability testing communication protocol 1

$\mathcal{K}_A = \mathcal{A}gt \cup \{K_b\}$	
$\mathcal{K}_B = \mathcal{A}gt \cup \{K_b^{-1}\}$	
Message1	$A \rightarrow B$: Are you online?
Message2	$B \rightarrow A$: $\{\text{Yes}\}_{K_b^{-1}}$

Table 1.2: Availability testing communication protocol 2

A cryptographic protocol is a communication protocol, with the difference that it uses cryptography to achieve one or more security goals.

Definition 1.4.2 (Cryptographic protocol)
A cryptoprotocol is a precisely defined sequence of communication and computation steps. A communication step transfers messages form the sender to the receiver, while a computation step (encryption, decryption, verification, ...) update a principal's internal state.

In the rest of this thesis, we use protocol or cryptoprotocol to denote cryptographic protocol.

1.4.1 Notations

In this thesis, we adopt the standard notation for describing cryptographic protocols. A message is composed of one or more primitive words. A message m encrypted with key K is written $\{m\}_K$ and forms a word by itself. Concatenated messages are separated by dots. Words have the following naming conventions: Encryption keys, nonces and timestamps are respectively written K, N and T. Principals are written A, B, S and I, where A and B stand for principals who wish to communicate, S for a trusted server and I for a potential intruder. We use $I(A)$ to mean that the intruder is impersonating the principal A. Agt denotes the set of all agents that may use the system. $\mathcal{K}_i$ stands

for the initial knowledge of principal i. Subscripts will be used to denote an association to a principal; for example N_a is a nonce that belongs to A and K_{bs} is a shared key between B and S. K_a stands for A's public key while K_a^{-1} denotes A's private key. A message m signed with A's private key would look like $\{m\}_{K_a^{-1}}$.

Through the following protocol, we want to explain some basic concepts.

$\mathcal{K}_A = \mathcal{A}gt \cup \{N_a.K_b\}$	
$\mathcal{K}_B = \mathcal{A}gt \cup \{K_b^{-1}\}$	
Message1	$A \rightarrow B$: Are you online?.N_a
Message2	$B \rightarrow A$: Yes.$\{N_a\}_{K_b^{-1}}$

Table 1.3: Availability testing communication protocol 3

- **Initial Knowledge:** To describe cryptographic protocols, it is important to give every principal some initial knowledge, that is something they know before they even start a run of the protocol. In some cases, we suppose A and B share a secret key; this can be stated by putting K_{ab} in the initial knowledge of A and B. For example, in the protocol above, A's initial knowledge is $\mathcal{A}gt \cup \{N_a, K_b\}$. It means that A knows the identifiers of every principal in the system, the fresh nonce N_a (that he generated himself) and finally B's public key. The concept of initial knowledge allows simplifying cryptoprotocol in the sense that we do not require an explicit key exchange to occur for A to get B's public key. Note that, from now on, we omit the explicit enumeration of principals' initial knowledge as often as possible (when it is clear from the context) to reduce the size of protocols.

- **Roles and principals:** Suppose *Alice* wants to check if *Bob* is available. *Alice* and *Bob* are principals, real entities executing the protocol. In order to test *Bob*'s availability, *Alice* will have to play the role A and *Bob* the role B. Roles are protocol abstractions that denote the way principals should act and how they perceive the different messages. If later *Bob* wants to check *Alice*'s availability, he will then play the role A and *Alice* will play role B. If a protocol allows principals to change their role from one execution to another, this is a multi-role protocol. However, if we can attribute role A to *Alice* and role B to *Bob*, we are in a single-role protocol. In this case *Alice* can check the availability of *Bob* while *Bob* cannot check *Alice*'s. From now on, we assume that the protocols we are dealing with are all multi-role protocols.

- **Protocol run:** A run of a protocol is an instantiation of the protocol specification where we specify which principals will run the protocol, what role each principal

will play and the concrete value of the messages. We often use sessions to indicate a run of a protocol.

- **Nonces and timestamps:** Nonces are randomly generated numbers. They are used to warrant the freshness of messages. In other words, they prevent someone from attacking the protocol by using old messages. In the protocol of Table 1.3, the nonce N_a is used by A as a challenge. If B is available, he can answer the challenge to prove his availability. In the protocol of Table 1.2, which does the same job, an intruder I might be able to convince A of B's availability while B is not. Suppose A tests whether B is available or not and B is. B will answer with the message $\{\text{Yes}\}_{K_b^{-1}}$, according to the protocol. If I records this message, he can fool A whenever A asks for B's availability in the future. This flaw exists because there is no way for A to tell whether Message2 is fresh or not; it is not enough to be sure that the message originates from B, it must also be fresh. Protocol of Table 1.4 gives an example of using timestamp for freshness.

Message1	$A \rightarrow B$: Are you online?
Message2	$B \rightarrow A : \{\text{Yes}.T_b\}_{K_b^{-1}}$

Table 1.4: Availability testing communication protocol 4

In this last protocol, principal B will include the value of his local clock in the message. Now A can check whether the received message is fresh by checking the time of B's clock with his own (allowing a little amount of time for network delay). The main problem with timestamp is the need for A and B to synchronize their clocks.

1.4.2 Role abstraction

Since role abstraction and role generalization will be important in this thesis, we feel the need to develop more on this than the mere definitions we just gave. A role is an abstraction of a protocol in which we are only interested in a given principal's point of view. In other words, roles are meant to describe how a principal perceives and acts in the protocol. It is clear that two different principals do not perceive the protocol in the same manner. There are as much roles in a protocol as there are principals involved. For example, let's take the *Woo & Lam* protocol of Table 1.5. In this protocol, we have three roles: A, B and S. Clearly, a principal playing the role of A does not perceive the protocol in the same way as a principal playing the role of B because A will participate in only the first three steps while B will participate in all five steps. The three roles of

this protocol, namely $\mathcal{R}(A)$, $\mathcal{R}(B)$ and $\mathcal{R}(S)$, are pictured in Table 1.6. To extract the role of a principal p in a given protocol, we proceed in four steps:

- We take out the steps where p doesn't participate and keep only those where p is involved.
- For each step, we replace the principal p' with whom p communicates by an intruder playing his role ($I(p')$). This is to emphasis the fact that the intruder sees every exchanged message, see Section 1.4.4 for the intruder model.
- For each step, we replace the step number x by $\alpha.x$ to denote the step x of any given session α (for the sake of clarity, we use α for the role of A, β for the role of B and δ for S's role).
- For every message, we replace each fresh message by the same message bound to the session α by using a superscript notation. For example, the nonce N_a would become N_a^α.

1	$A \rightarrow B : A$
2	$B \rightarrow A : N_b$
3	$A \rightarrow B : \{N_b\}_{K_{as}}$
4	$B \rightarrow S : \{A.\{N_b\}_{K_{as}}\}_{K_{bs}}$
5	$S \rightarrow B : \{N_b\}_{K_{bs}}$

Table 1.5: *Woo & Lam* protocol

		$\alpha.1$ $A \rightarrow I(B) : A$
		$\alpha.2$ $I(B) \rightarrow A : N_b^\alpha$
		$\alpha.3$ $A \rightarrow I(B) : \{N_b^\alpha\}_{k_{as}}$
1 $A \rightarrow B : A$		$\beta.1$ $I(A) \rightarrow B : A$
2 $B \rightarrow A : N_b$		$\beta.2$ $B \rightarrow I(A) : N_b^\beta$
3 $A \rightarrow B : \{N_b\}_{k_{as}}$	$\Longrightarrow$	$\beta.3$ $I(A) \rightarrow B : \{N_b^\beta\}_{k_{as}}$
4 $B \rightarrow S : \{A.\{N_b\}_{k_{as}}\}_{k_{bs}}$		$\beta.4$ $B \rightarrow I(S) : \{A.\{N_b^\beta\}_{k_{as}}\}_{k_{bs}}$
5 $S \rightarrow B : \{N_b\}_{K_{bs}}$		$\beta.5$ $I(S) \rightarrow B : \{N_b^\beta\}_{k_{bs}}$
		$\delta.4$ $I(B) \rightarrow S : \{A.\{N_b^\delta\}_{k_{as}}\}_{k_{bs}}$
		$\delta.5$ $S \rightarrow I(B) : \{N_b^\delta\}_{k_{bs}}$

Table 1.6: Extracting roles

1.4.3 Role generalization

A generalized role is a generalization of a single role. For any given role, there is only one generalized role. The goal of generalized roles is to explicitly show what verifications a principal can do when he receives a message and how he really acts in the steps of his role. If the principal cannot do any verification on a received message, then we replace this message by a variable to mean that regardless of what is received, he will accept it as a good value. From the three roles given above, we extract the three generalized roles, namely $\mathcal{GR}(A)$, $\mathcal{GR}(B)$ and $\mathcal{GR}(S)$, which can be found in Table 1.7. The procedure to produce the generalized role of a principal p is the following step.

- For every message $m_1, m_2, ..., m_n$, we regard each m_i and if p already knows the value of m_i, we leave it. Otherwise, if m_i is of the form $\{m_{i1}, m_{i2}, ..., m_{ip}\}_K$ and p knows the key K, we start the process back with every m_{ij}, if not, we replace m_i by a variable.

$\alpha.1$	$A \rightarrow I(B) : A$		$\alpha.1$ $A \rightarrow I(B) : A$
$\alpha.2$	$I(B) \rightarrow A : N_b^\alpha$	$\Longrightarrow$	$\alpha.2$ $I(B) \rightarrow A : X$
$\alpha.3$	$A \rightarrow I(B) : \{N_b^\alpha\}_{k_{as}}$		$\alpha.3$ $A \rightarrow I(B) : \{X\}_{k_{as}}$
$\beta.1$	$I(A) \rightarrow B : A$		$\beta.1$ $I(A) \rightarrow B : A$
$\beta.2$	$B \rightarrow I(A) : N_b^\beta$		$\beta.2$ $B \rightarrow I(A) : N_b^\beta$
$\beta.3$	$I(A) \rightarrow B : \{N_b^\beta\}_{k_{as}}$	$\Longrightarrow$	$\beta.3$ $I(A) \rightarrow B : Y$
$\beta.4$	$B \rightarrow I(S) : \{A.\{N_b^\beta\}_{k_{as}}\}_{k_{bs}}$		$\beta.4$ $B \rightarrow I(S) : \{A.Y\}_{k_{bs}}$
$\beta.5$	$I(S) \rightarrow B : \{N_b^\beta\}_{k_{bs}}$		$\beta.5$ $I(S) \rightarrow B : \{N_b^\beta\}_{k_{bs}}$
$\delta.4$	$I(B) \rightarrow S : \{A.\{N_b^\delta\}_{k_{as}}\}_{k_{bs}}$	$\Longrightarrow$	$\delta.4$ $I(B) \rightarrow S : \{A.\{Z\}_{k_{as}}\}_{k_{bs}}$
$\delta.5$	$S \rightarrow I(B) : \{N_b^\delta\}_{k_{bs}}$		$\delta.5$ $S \rightarrow I(B) : \{Z\}_{k_{bs}}$

Table 1.7: Generalizing roles

The process of role generalization will make the following modifications on the roles:

- In the role of A: step one remains unchanged; in steps two and three, we replaced the message N_b^α by the variable X since the nonce received is a fresh value sent by B and thus A has no knowledge over this value and he cannot perform any kind of verification.

- In the role of B: steps 1,2 and 5 are unchanged; in both steps 3 and 4, we had to replace the message $\{N_b^\beta\}_{k_{as}}$ by the variable Y to explicitly show that principal B cannot check the value of this message since it is encrypted under a key unknown to B, namely k_{as}.
- In the role of S: we replace message N_b^δ in both steps for the same reason as the modification in the role of A.

In the generalized role of S, we can really see the importance of the generalized role concept. In the protocol, S is used to decrypt the message $\{N_b\}_{K_{as}}$ to allow B to check if A has been able to answer the challenge. But in reality, S has no way to be sure that the message he will send to B will be $\{N_b\}_{K_{bs}}$. In fact, if B sends to S any message of the form $\{A.\{Z\}_{K_{as}}\}_{K_{bs}}$ at step four (where Z stands for any message), the server will answer with $\{Z\}_{K_{bs}}$ regardless of the message Z. So the server should be seen as a third party able to decrypt a message encrypted with the secret key shared between any principal and the server provided that it is supplied with the principal identifier (the server needs to know which key to use to decrypt the message) and that you are a legitimate user of the system (you need a key shared with the server).

1.4.4 Intruder abilities

It is important, when talking about protocol security, to define the ability of a possible intruder against which we need to keep information secure. We could need to have a protocol that is secure over a passive intruder, that is one who only spies on the network and, with the learned messages, tries to compute the secret he wants to know. However, such a model is not realistic for a hostile environment as the Internet. In fact, over the Internet, an intruder is not limited to spy the messages sent, but he may also create some messages. In the worst case, we could think of an intruder being in total control of the network; as represented by the *Dolev-Yao* model (see [25]) in which the intruder can do the following:

- The intruder intercepts every message circulating on the network.
- He can send messages whenever he wants to.
- When he receives a message of the form $m_1.m_2$, he can decompose the message to get m_1 and m_2.
- When he receives a message of the form $\{m\}_k$ and he knows the key k, than the intruder can decrypt the message and get m.

- Knowing two messages X and Y, the intruder is able to concatenate them to form $X.Y$ and to encrypt them to form $\{X\}_Y$.

1.4.5 Valid traces

From a protocol, we can form infinitely many traces. A trace is a sequence of communication step where every exchanged message passes through the intruder. Generally, a trace is formed by using an interleaving of many sessions. For example, if we have a protocol with four steps M_1, M_2, M_3 and M_4, a trace could be $M_1, M_2, M_3, M_1, M_2, M_1, M_2$. Since many traces are not realizable by a given protocol, we would like to reduce to valid traces only.

Definition 1.4.3 (Valid trace)
A trace is valid if it is well-formed and well defined.

Definition 1.4.4 (Well-formed trace)
For a trace to be well-formed, we first need that the honest principals taking part in that trace act exactly according to the protocol. Moreover, we need the trace to be separable in session, where every session is formed by the prefix of one and only one generalized role. This is of course up to a substitution of the variables and principal names for this generalized role.

Definition 1.4.5 (Well defined trace)
A trace is well defined if every message sent by the intruder is derivable from his current knowledge. The intruder's current knowledge is formed by his initial knowledge plus the knowledge he can derive, by applying encryption, decryption, concatenation and message decomposition, on the messages he has received before the current step.

1.1	$A \rightarrow I(B) : A$
1.2	$I(B) \rightarrow A : toto$
1.3	$A \rightarrow I(B) : \{tata\}_{K_{as}}$

Table 1.8: Not well-formed trace 1

Example 1.4.3 (Not well-formed traces)
Here we present two examples of traces that are not well-formed and thus not valid over the *Woo & Lam* protocol of Table 1.5. In Table 1.8, we have a trace composed of only one session that is a prefix of $\mathcal{RG}(A)$. The problem is that no substitution unifying the trace and $\mathcal{RG}(A)$ can be found since X would have to be substituted for two values: *toto* and *tata*. For the trace in Table 1.9, we have two sessions: one with

session identifier 1 and one with session identifier 2. The session 1 is a prefix of $\mathcal{RG}(A)$ with a proper substitution $\sigma = \{\alpha \leftarrow 1, X \leftarrow toto\}$. However, session 2 is not a prefix of any generalized role and thus this trace is neither well-formed nor valid.

1.1	$A \rightarrow I(B) : A$
1.2	$I(B) \rightarrow A : toto$
1.3	$A \rightarrow I(B) : \{toto\}_{K_{as}}$
2.2	$B \rightarrow I(A) : tata$
2.3	$I(A) \rightarrow B : anything$

Table 1.9: Not well-formed trace 2

Example 1.4.4 (Not well defined trace)
Taking the trace of Table 1.10 as an example, we can explain why it is not well defined. The messages sent by the intruder are A and $\{toto\}_{K_{bs}}$. The intruder knows A since it is the identifier of a principal. But for the message $\{toto\}_{K_{bs}}$, the intruder knows only *toto* and does not know the key needed to encrypt the message so he cannot figure out this message. Thus this trace is neither well defined nor valid.

1.1	$I(A) \rightarrow B : A$
1.2	$B \rightarrow I(A) : toto$
1.3	$I(A) \rightarrow B : \{toto\}_{K_{bs}}$

Table 1.10: Not well defined trace

Example 1.4.5 (Valid trace)
In the trace of Table 1.11, we have 5 sessions. Session 1 is a prefix of $\mathcal{RG}(B)$ with the substitution $\sigma_1 = \{Y \leftarrow anything\}$. Sessions 2 and 3 are prefix of $\mathcal{RG}(A)$ with substitutions $\sigma_2 = \{A \leftarrow C, B \leftarrow D, X \leftarrow N_b\}$ and $\sigma_3 = \{A \leftarrow C, B \leftarrow E, X \leftarrow C, \{N_b\}_{K_{cs}}\}$ respectively. We can also find a substitution for sessions 4 and 5 which are prefix of $\mathcal{RG}(B)$ and $\mathcal{RG}(S)$ respectively. With the previous discussion, we conclude that this trace is well-formed. To show it is well defined, let's take out every message sent by the intruder:

- At step 1.1, the intruder sends the message A, which is ok.
- At step 1.3, he sends *anything*, which is clearly ok.
- At step 2.2, he sends the message N_b that he received in step 1.2.
- At step 3.2, he concatenates the messages received at steps 3.1 and 2.3.

- The message of step 4.1 is common knowledge.
- At step 4.3, this is simply a replay of message 3.3.
- Message 5.4 is a replay of message 4.5.
- The last message at step 1.5 is received by the intruder at step 5.5.

From there we conclude the trace is well defined and since it is also well-formed, it is valid.

1.1	$I(A) \rightarrow B : A$
1.2	$B \rightarrow I(A) : N_b$
1.3	$I(A) \rightarrow B : anything$
1.4	$B \rightarrow I(S) : \{A.anything\}_{K_{bs}}$
2.1	$C \rightarrow I(D) : C$
2.2	$I(D) \rightarrow C : N_b$
2.3	$C \rightarrow I(D) : \{N_b\}_{K_{cs}}$
3.1	$C \rightarrow I(E) : C$
3.2	$I(E) \rightarrow C : C.\{N_b\}_{K_{cs}}$
3.3	$C \rightarrow I(E) : \{C.\{N_b\}_{K_{cs}}\}_{K_{cs}}$
4.1	$I(C) \rightarrow B) : C$
4.2	$B \rightarrow I(C) : N'_b$
4.3	$I(C) \rightarrow B : \{C.\{N_b\}_{K_{cs}}\}_{K_{cs}}$
4.4	$B \rightarrow S : \{C.\{C.\{N_b\}_{K_{cs}}\}_{K_{cs}}\}_{K_{bs}}$
4.5	$S \rightarrow I(B) : \{C.\{N_b\}_{K_{cs}}\}_{K_{bs}}$
5.4	$I(B) \rightarrow S : \{C.\{N_b\}_{K_{cs}}\}_{K_{bs}}$
5.5	$S \rightarrow I(B) : \{N_b\}_{K_bs}$
1.5	$I(S) \rightarrow B : \{N_b\}_{K_bs}$

Table 1.11: Valid trace

1.4.6 Protocol classification

There are many ways of classifying protocols. One can choose to classify protocols regarding their security goals (confidentiality, authentication, E-commerce, ...). Another might wish to separate protocols that use different cryptosystems (public key vs secret key). We can also differentiate protocols by the help they require from a third party (trusted server, judge, ...). we will develop no further on the subject but refer the interested reader to [52, 57, 58] for some classifications.

1.5 Flaws in protocols

Regarding the importance of cryptographic protocols (bank transactions, military secret communications, ...), it is primordial that those protocols be correct. We should not be content with a protocol that works in most of the cases but fails in few others. We need flawless protocols. Basically, a flaw is a property of a protocol that contradicts the security goals of the protocol. In [12], *Landweher*, *Bull*, *McDermott* & *Choi* defined a flaw as follows.

Definition 1.5.1 (Security flaw)
A security flaw is a part of a program that can cause the system to violate its security requirements. Finding security flaws, then, demands some knowledge of the system security requirements. These requirements vary according to the system and the application.

Definition 1.5.2 (Flawed protocol)
A protocol P is flawed with respect to a given security property when there is a valid trace over the steps of P such that, when executing this trace, the security property is violated.

Definition 1.5.3 (Correct protocol)
A protocol P is correct with respect to a given security property if no valid trace over the steps of P violate the security property.

To have a good understanding of the difficulties in constructing flawless cryptographic protocols or proving their correctness, it is convenient to start with a classification of the different flaws that may arise in a protocol. The classification proposed is based on the work of *Carlsen* [14] and *Clark* & *Jacob* [16].

To prove the existence of a flaw in a protocol, we give an attack on this protocol that violates the security property. Our way to describe an attack is to give a procedure (a valid trace as defined in Section 1.4.5) to exploit the flaw in the protocol.

1.5.1 Freshness flaws

A freshness flaw occurs when an intruder can attack the protocol by reusing messages that he intercepted in a previous run of the protocol. The protocol of Table 1.3 contains such a flaw, as we argued earlier to support the need of nonces and timestamps. For another example of freshness flaw, let's build an attack on the following protocol:

1	$A \rightarrow S : A.B.N_a$
2	$S \rightarrow A : \{N_a.B.K_{ab}.\{K_{ab}.A\}_{K_{bs}}\}_{K_{as}}$
3	$A \rightarrow B : \{K_{ab}.A\}_{K_{bs}}$
4	$B \rightarrow A : \{N_b\}_{K_{ab}}$
5	$A \rightarrow B : \{N_b - 1\}_{K_{ab}}$

Table 1.12: *Needham-Schroeder* symmetric key distribution protocol

This protocol aims to distribute a symmetric key between principals A and B. After such a distribution, it is important, among other things, that B believes that he shares the key with A. In the attack of Table 1.13, this flaw was found by *Denning* & *Sacco* in 1981, an intruder may be able to convince B that the key is shared with A while it is shared with I. Suppose the intruder intercepted $Message3$ of a previous run of the protocol, so I intercepted $\{K'_{ab}, A\}_{K_{bs}}$, and that he has been able to figure out the shared key K'_{ab}. Although K'_{ab} is not used anymore between A and B, for it is too old, the intruder can use its knowledge of this key and attack the protocol given on Table 1.12. This attack is possible since there is no way for B to tell if either the message received at step 3 is fresh or not.

3	$I(A) \rightarrow B : \{K'_{ab}.A\}_{K_{bs}}$
4	$B \rightarrow I(A) : \{N_b\}_{K'_{ab}}$
5	$I(A) \rightarrow B : \{N_b - 1\}_{K'_{ab}}$

Table 1.13: *Needham-Schroeder* symmetric key distribution protocol attack

1.5.2 Oracle flaws

A cryptoprotocol has an oracle flaw if it allows an intruder to learn a secret message or to foretell its content. The example we provide is based on the following simple protocol.

1	$A \rightarrow B : \{N_a\}_{K_{ab}}$
2	$B \rightarrow A : \{N_a + 1\}_{K_{ab}}$

Table 1.14: Availability testing communication protocol 5

As the protocols presented previously in tables 1.1 to 1.4, it aims to test if the

principal playing the role B is available. In the attack presented in Table 1.15, an intruder will be able to convince principal A that B is available while it is not necessarily the case. This attack works as follows (we suppose B is not available). First A sends a challenge for B. The intruder will use the same challenge, in a new run of the protocol, to learn the correct answer to this challenge. To do this, $I(B)$ asks for A's availability by providing exactly the same challenge. Now A is able to answer it, since he knows the required key, and thus provides $I(B)$ with the answer to his own challenge. From there I can convince A that B is available even if it is not the case.

1.1	$A \rightarrow I(B) : \{N_a\}_{K_{ab}}$
2.1	$I(B) \rightarrow A : \{N_a\}_{K_{ab}}$
2.2	$A \rightarrow I(B) : \{N_a + 1\}_{K_{ab}}$
1.2	$I(B) \rightarrow A : \{N_a + 1\}_{K_{ab}}$

Table 1.15: Oracle flaw

1.5.3 Type flaws

As messages are represented by binary string, it is important to know the length of each submessage to retrieve them from a concatenated message.

Example 1.5.1 (Retrieving parts of a concatenated message)
Suppose the message received by a principal at a given step of a protocol run is supposed to be $N_a.K_{ab}.m$. Let's take "01101010010110100101" as the received binary string. To know the value of N_a, K_{ab} and m, the principal needs to know how to separate the binary string.

A type flaw occurs when an intruder is able to switch the type of submessages and fools the concerned principal. The protocol of Table 1.16 provides a good example of such a flaw. The intruder lets A and B execute the first three steps of the protocol and then sends in step four the message intercepted in step 2 ($\{N_a + 1.N_b\}_{K_{ab}}$). With this, A will believe that $N_a + 1$ is the key shared between him and B. This is possible only if A is not able to distinguish between the type of message $N_a + 1$ and the type of the intended message K'_{ab}.

1	$A \rightarrow B : A.N_a$
2	$B \rightarrow A : \{N_a + 1, N_b\}_{K_{ab}}$
3	$A \rightarrow B : \{N_b + 1\}_{K_{ab}}$
4	$B \rightarrow A : \{K'_{ab}.N'_b\}_{K_{ab}}$

Table 1.16: Key distribution protocol with a type flaw

Heather, Lowe & Schneider propose in [35] a technique to prevent the occurrence of type flaw in cryptoprotocols.

1.5.4 Binding flaws

When using a public key cryptosystem, it is very important for principal A to be convinced that the public key he has received is indeed B's public key. A binding flaw occurs if an intruder can convince A that B's public key is K_I, which is in fact the public key of the intruder. With such a flaw, I will be able to read every message sent from A to B. The protocol of Table 1.17 presents such a flaw. We discussed briefly this kind of flaw in Section 1.2.2 see Definition 1.2.6.

1	$A \rightarrow S : A.B.N_a$
2	$S \rightarrow A : S.\{S.A.N_a.K_b\}_{K_s^{-1}}$

Table 1.17: Public key distribution protocol with a binding flaw

In this protocol, principal A wishes to know the public key of another principal (the one playing B's role). In step 1, A sends a request to the server saying: *I'm A, I want to have B's public key and here's a nonce to ensure the freshness of the key.* In step 2, the server replies by saying *Here's the public key you asked for with your nonce for freshness, everything is signed with my private key.* The problem with this protocol is that the server does not explicitly say whose public key he is sending. The attack can be performed as in Table 1.18.

1.1	$A \rightarrow I(S) : A.B.N_a$
2.1	$I(A) \rightarrow S : A.I.N_a$
2.2	$S \rightarrow I(A) : S.\{S.A.N_a.K_I\}_{K_s^{-1}}$
1.2	$I(S) \rightarrow A : S.\{S.A.N_a.K_I\}_{K_s^{-1}}$

Table 1.18: Binding flaw attack

1.5.5 Repudiation flaws

There is a repudiation flaw in a cryptoprotocol when one participant can deny his participation in a run of the protocol. A good example of this kind of flaw occurs in the coin tossing protocol proposed by *Toussaint*. A description of this protocol is given is the following table.

1	$B \rightarrow A : Head$ or $Tail$
2	$A \rightarrow B : \{N_a.Head\}_{N_a}.\{N_a.Tail\}_{N_a}$ or $\{N_a.Tail\}_{N_a}.\{N_a.Head\}_{N_a}$
3	$B \rightarrow A : \{N_a.Head\}_{N_a}$ or $\{N_a.Tail\}_{N_a}$
4	$A \rightarrow B : N_a$

Table 1.19: *Toussaint* coin tossing protocol

This protocol works as follows. In the first step, B guesses head or tail and sends his guess to A. At step two, A creates two messages and sends them to B, the order of the two messages must be chosen randomly: the first message is head concatenated with N_a and then, encrypted with N_a; the other message is the same except that tail replaces of head. . At step 3, B will try to guess which of the two messages contains his choice of step 1 and will send his guess to A. A can check if B's guess corresponds to his initial choice and he will send to B the key N_a used for encryption so B can verify the outcome. B wins if he guessed the same in message 1 and 3, otherwise, A wins. In [83], *Toussaint* showed that A and B win with the same probability (1/2). Unfortunately, if between steps 3 and 4 A sees that B guessed correctly, he can abort the protocol, denying his participation. Thus, either A wins or A aborts the protocol.

1.5.6 Implementation flaws

When searching for flaws in a protocol, we often suppose that the underlying cryptosystem is perfect. However, it could be the case that the interaction of the messages in the protocol together with the cryptosystem is enough to create a flaw. The three pass protocol shown in Table 1.20 can create such a flaw if the cryptosystem used is the XOR function.

1	$A \rightarrow B : \{m\}_{K_a}$
2	$B \rightarrow A : \{\{m\}_{K_a}\}_{K_b}$
3	$A \rightarrow B : \{m\}_{K_b}$

Table 1.20: Three passes protocol

Simply by using the XOR function properties, see Definition 1.5.4 below, a passive intruder can derive the secret message m simply by collecting the 3 messages sent in the protocol and XORing them together. Here is how to do this:

$$(\{m\}_{K_a} \oplus \{\{m\}_{K_a}\}_{K_b}) \oplus \{m\}_{K_b} \tag{1.1}$$

By the definition of using XOR as the encryption function, equation 1.1 becomes:

$$((m \oplus K_a) \oplus ((m \oplus K_a) \oplus K_b)) \oplus (m \oplus K_b) \tag{1.2}$$

This gives us, by the associativity of XOR:

$$(((m \oplus K_a) \oplus (m \oplus K_a)) \oplus K_b) \oplus (m \oplus K_b) \tag{1.3}$$

Which can be simplified, by the XOR self inverse property, to give:

$$(0 \oplus K_b) \oplus (m \oplus K_b) \tag{1.4}$$

Since 0 is a neutral element of the XOR operation we get:

$$K_b \oplus (m \oplus K_b) \tag{1.5}$$

Applying commutativity and associativity once more gives us:

$$(K_b \oplus K_b) \oplus m \tag{1.6}$$

Which is the same, using self inverse property, as:

$$0 \oplus m \tag{1.7}$$

That finally gives us:

$$m \tag{1.8}$$

Definition 1.5.4 (XOR properties)
Here are some useful properties of the XOR operation:

- *Identity:* $0 \oplus X = X$
- *Self inverse:* $X \oplus X = 0$
- *Commutativity:* $X \oplus Y = Y \oplus X$
- *Associativity:* $(X \oplus Y) \oplus Z = X \oplus (Y \oplus Z)$

1.5.7 Others

Other kinds of flaws may be found in cryptographic protocol. For example, we may have a key guessing flaw if the key space is small enough to allow exhaustive search. We may also found an elementary flaw in a protocol, that is an obvious flaw that does not require much effort from the intruder. An example of such a flaw can be found in the key distribution protocol of Table 1.21. In this protocol, the exchanged secret key is not encrypted but simply signed and thus anyone reading the message of step 1 can recover this key (since the key K_a is public knowledge).

1	$A \rightarrow B : \{N_a.K_{ab}\}_{K_a^{-1}}$
2	$B \rightarrow A : \{N_a\}_{K_{ab}}$

Table 1.21: Elementary flaw

1.6 Conclusion

This chapter was dedicated to cryptography and cryptographic protocols. We presented a formalism to describe protocols, formalism that will be used throughout this thesis, and some basic concepts such as roles and generalized roles that are important for some other chapters. Finally, we presented some typical flaws that may arise in a protocol. From this last part, it seems essential to develop formal tools to either prove the security of protocols or to develop new secure protocols.

Chapter 2

Protocol Analysis

Abstract

This chapter provides some examples of different techniques based on formal methods for the analysis of cryptographic protocols. We classify these methods in four groups depending on the goals they aim to achieve. Finally, we try to discuss the advantages and the limitations of each group and each method.

2.1 Introduction

At the end of Chapter 1, we saw that even if a protocol is designed carefully, it may still contain flaws. After the discovery, in 1995, of a flaw in the *Needham-Schroeder* protocol, which had been used for 17 years, the community realized the importance of having automatic formal methods for verifying cryptographic protocols; see [85]. It is now well known that the creation of a new protocol is highly error prone and the task of analyzing a protocol is a hard one. This chapter presents methods used for the analysis of cryptoprotocols; it is structured in the following way: in Section 2.2, we talk about some techniques used to find flaws in a protocol; Section 2.3 regroups techniques used to prove the correctness of a protocol and Section 2.4 talks about methods to decide if either a protocol is correct or not and finally in Section 2.5 we talk about some other methods, that are either historically important or close to the approach we will take in the next chapters, without classifying them. We do not intend to give an exhaustive classification of formal methods, but merely to give some examples on how formal methods may be used in analyzing protocols. For more details about the formal

methods used to analyze security protocols, see [55, 56].

2.2 Refuting correctness

In this section we present a method that tries to refute the correctness of a given protocol, that is to show the protocol is flawed. As shown in Figure 2.1, if the attempt to refute correctness is successful, then an attack against the protocol is provided and then the protocol is classified as flawed; otherwise, if the attempt fails (no attack is found) we cannot conclude that the protocol is flawed (since no flaw has been found) nor that the protocol is correct (since a flaw may still be present although it has not been found). The major drawback of this approach is that if it fails to find a flaw, no conclusion can be drawn. The main advantage here is that when a flaw is found, an attack scenario is provided so that we can convince anyone else that the protocol is flawed by showing the attack or we can analyze this attack to get some knowledge about why a flaw occurs in this protocol.

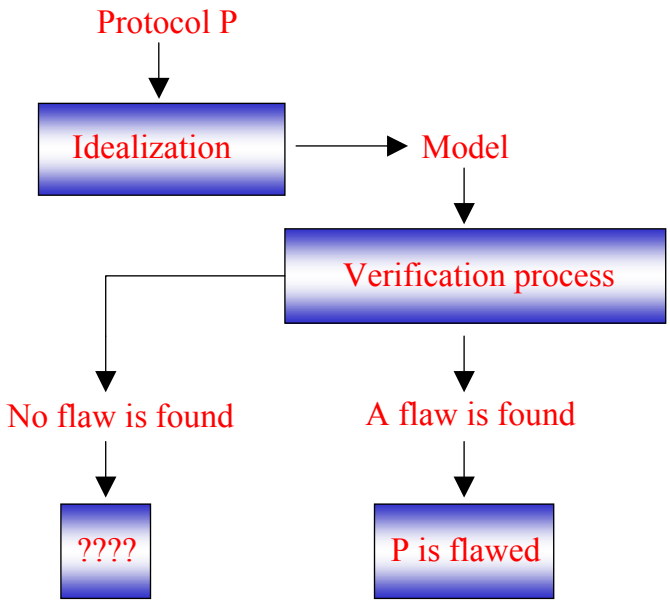

Figure 2.1: Refuting correctness

2.2.1 DYMNA

DYMNA has been developed at Université Laval by *Debabi, Mejri* & al. [18, 19, 20] and implemented by *Massicotte* [52]. This method uses the concepts of role and generalized role (see Section 1.4.2 and 1.4.3) to extract, from the protocol, the abilities of a potential intruder. Once these abilities are extracted and combined with the usual abilities of the intruder, we can try to see if an intruder is able to break the security property of the protocol. The general process of DYMNA is pictured in Figure 2.2. We explain the full process of DYMNA verification within the example of the *Woo* & *Lam* protocol of Table 1.5 presented earlier. This protocol was designed to allow one-way authentication, that is A wants to prove his identity to B.

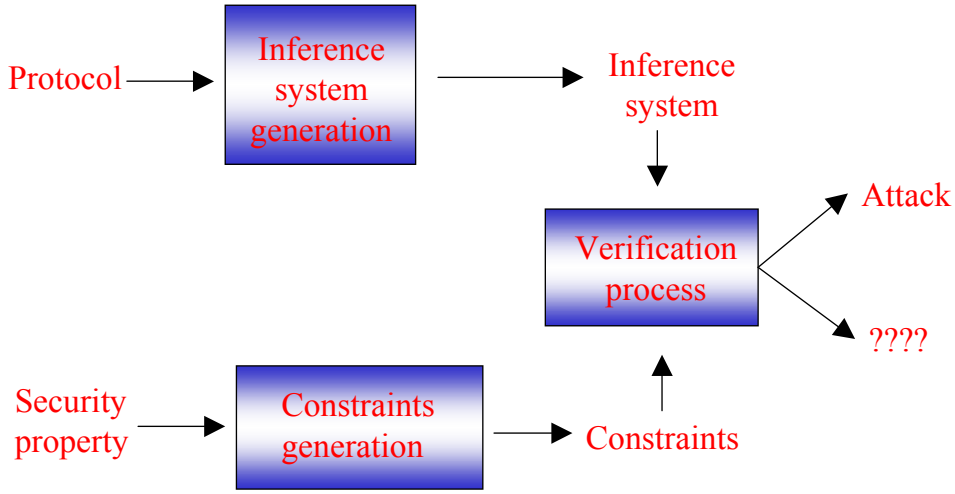

Figure 2.2: DYMNA general process

Inference system generation:

Starting from the initial protocol, we first extract the roles of the protocol as we did in Section 1.4.2, see Table 1.6. The role of A includes the steps marked with identifier α, the role of B those marked with identifier β and δ is used to indicate the steps of S's role.

For every role, we will use the generalization process to create the generalized roles of the protocol as we did in Section 1.4.3, see Table 1.7. Recall that the process of role generalization aims to exhibit the true behavior of honest principals and thus the abilities of the intruder to manipulate the protocol.

From every generalized role, we extract a set of inference rules that models the ability given to the intruder by a specific generalized role. A rule $R = \frac{m_1 \quad m_2}{m}$ must be interpreted in this way: if the intruder knows messages m_1 and m_2, then he can learn the message m by using the protocol. From the generalized role of an agent P, we extract a rule for every step s in which P is the sender, each rule is constructed like this: for every step s' preceding s such that P is the receiver in s', we add the message of step s' to the list of premises of the rule; the message of step s is the conclusion of the rule. In Table 2.1 we see that A's role produce two inference rules. The first one states that by doing nothing, the intruder can get the identity of any agent, while the second one illustrates that if the intruder can provide a message X, then the protocol allows him to get $\{X\}_{k_{as}}$. We also see that the role of B is modeled through two rules. The first one means that providing an agent identifier gives to the intruder a fresh nonce generated by the agent B. The second says that the intruder can get $\{A.X\}_{k_{bs}}$ if he can provide both message A and message X. Finally, we get the rule extracted from the server's generalized role, which says that the intruder can get $\{Z\}_{k_{bs}}$ by providing $\{\{A.Z\}_{k_{as}}\}_{k_{bs}}$. In Table 2.1, together with every rule $R = \frac{m_1 \quad m_2}{m}$ is associated a proof, this proof is the trace proving how the intruder can learn message m by using the protocol with the messages m_1 and m_2. Those traces will be used to build an attack if a flaw is discovered in the protocol.

Generalized role	**Rule**	**Proof**
$\mathcal{GR}(A)$	$R_1 = \frac{}{A}$	$\alpha.1 \quad A \longrightarrow I(B) : A$
$\mathcal{GR}(A)$	$R_2 = \frac{X}{\{X\}_{k_{as}}}$	$\alpha.1 \quad A \longrightarrow I(B) : A$ $\alpha.2 \quad I(B) \longrightarrow A : X$ $\alpha.3 \quad A \longrightarrow I(B) : \{X\}_{k_{as}}$
$\mathcal{GR}(B)$	$R_3 = \frac{A}{N_b^\beta}$	$\beta.1 \quad I(A) \longrightarrow B : A$ $\beta.2 \quad B \longrightarrow I(A) : N_b^\beta$
$\mathcal{GR}(B)$	$R_4 = \frac{A \quad Y}{\{A.Y\}_{k_{bs}}}$	$\beta.1 \quad I(A) \longrightarrow B : A$ $\beta.2 \quad B \longrightarrow I(A) : N_b^\beta$ $\beta.3 \quad I(A) \longrightarrow B : Y$ $\beta.4 \quad B \longrightarrow I(S) : \{A.Y\}_{k_{bs}}$
$\mathcal{GR}(S)$	$R_5 = \frac{\{A.\{Z\}_{k_{as}}\}_{k_{bs}}}{\{Z\}_{k_{bs}}}$	$\delta.4 \quad I(B) \longrightarrow S : \{A.\{Z\}_{k_{as}}\}_{k_{bs}}$ $\delta.5 \quad S \longrightarrow I(B) : \{Z\}_{k_{bs}}$

Table 2.1: Extracting rules from generalized roles

To the set of rules extracted from the protocol, we add the rules representing the usual intruder abilities such as encryption, decryption, ... These rules are presented in Table 2.2, they are: R_c used to concatenate messages, R_{d_1} and R_{d_2} both modeling

deconcatenation, R_e for encryption, R_d for decryption and $R_\mathcal{K}$ to exploit the initial knowledge of the intruder. These rules have no trace associated with them since they don't result from an utilization of the protocol but simply of an internal computation by the intruder. Taking the rules contained in Table 2.1 together with those of Table 2.2 we get an inference system modeling exactly what the intruder can do in the protocol. So the intruder can learn a message m using the protocol if and only if m is a theorem in the corresponding inference system S when the intruder knowledge is $\mathcal{K}$ (noted $\mathcal{K} \vdash_S m$).

$R_c = \frac{X \quad Y}{X.Y}$	$R_{d_1} = \frac{X.Y}{X}$	$R_{d_2} = \frac{X.Y}{Y}$
$R_e = \frac{X \quad Y}{\{X\}_Y}$	$R_d = \frac{\{X\}_Y \quad Y}{X}$	$R_\mathcal{K} = \frac{m \in \mathcal{K}}{m}$

Table 2.2: Extracting rules from the intruder abilities

Constraints generation:

From a given security property, we want to generate a set of constraints in such a way that the protocol is flawed if every constraint is provable in the underlying inference system. In our example, we want to check the one-way authentication over B for the *Woo & Lam* protocol. We take each step of $\mathcal{RG}(B)$ one by one. If in step s B receives a message, then the intruder must be able to provide it, if in s B sends a message, then we add this message to the knowledge of the intruder. Every step is detailed in Table 2.3. In step $\beta.1$, B must receive the message A, so the intruder must be able to provide this message. In step $\beta.2$, B sends the message N_b^β so we increase the knowledge of the intruder with this message. We can now take the set of constraints to be $\{\mathcal{K}_I \vdash A, \mathcal{K}^1 \vdash Y, \mathcal{K}^2 \vdash \{N_b^\beta\}_{k_{bs}}\}$.

$$
\begin{array}{lll}
\beta.1 \quad I(A) \to B : A & \Longrightarrow & \mathcal{K}_I \vdash A \\
\beta.2 \quad B \to I(A) : N_b^\beta & \Longrightarrow & \mathcal{K}^1 = \mathcal{K}_I \cup \{N_b^\beta\} \\
\beta.3 \quad I(A) \to B : Y & \Longrightarrow & \mathcal{K}^1 \vdash Y \\
\beta.4 \quad B \to I(S) : \{A.Y\}_{k_{bs}} & \Longrightarrow & \mathcal{K}^2 = \mathcal{K}^1 \cup \{\{A.Y\}_{k_{bs}}\} \\
\beta.5 \quad I(S) \to B : \{N_b^\beta\}_{k_{bs}} & \Longrightarrow & \mathcal{K}^2 \vdash \{N_b^\beta\}_{k_{bs}}
\end{array}
$$

Table 2.3: Extracting constraints from $\mathcal{GR}(B)$

Verification process:

What remains to be done is to check if every constraint is a theorem in the inference system generated from the protocol. The main problem of DYMNA is that given a message m and an inference system S, it is in general undecidable whether or not m is a theorem in S. The search of a proof for m may not terminate. We will address the problem of termination of inference systems in Chapter 3. Thus if we find a proof for m, then we are sure the intruder can learn m from the protocol. Otherwise, we cannot conclude. For the particular example we are working on, the three constraints $\mathcal{K}_I \vdash A$, $\mathcal{K}^1 \vdash Y$ and $\mathcal{K}^2 \vdash \{N_b^\beta\}_{k_{bs}}$ are provable as follows:

- $\mathcal{K}_I \vdash A$: it follows from the fact that $A \in \mathcal{K}_I$;
- $\mathcal{K}^1 \vdash Y$: it is a simple consequence of $\mathcal{K}^1 \neq \emptyset$;
- $\mathcal{K}^2 \vdash \{N_b^\beta\}_{k_{bs}}$: since $N_b^\beta \in \mathcal{K}^2$, we can use R_2 with $\{X \leftarrow N_b^\beta, a \leftarrow b\}$ to get $\{N_b^\beta\}_{k_{bs}}$. The proof tree of this last constraint is shown in equation 2.1.

$$\sigma(R_2)\frac{R_{\mathcal{K}^2}\frac{N_b^\beta \in \mathcal{K}^2}{N_b^\beta}}{\{N_b^\beta\}} \tag{2.1}$$

Since all constraints are provable, we conclude that the protocol is flawed and we can mount an attack as shown in Table 2.4. We can easily check that the trace forming the attack is a valid trace as defined in Section 1.4.5.

$$\begin{array}{ll}
1.1 & I(A) \rightarrow B : A \\
1.2 & B \rightarrow I(A) : N_b^1 \\
1.3 & I(A) \rightarrow B : anything \\
1.4 & B \rightarrow I(S) : \{A.anything\}_{k_{bs}} \\
\quad 2.1 & B \rightarrow I(C) : B \\
\quad 2.2 & I(C) \rightarrow B : N_b^1 \\
\quad 2.3 & B \rightarrow I(C) : \{N_b^1\}_{k_{bs}} \\
1.5 & I(S) \rightarrow B : \{N_b^1\}_{k_{bs}}
\end{array}$$

Table 2.4: Attack on the *Woo & Lam* protocol

2.3 Proving correctness

In this section, we present some methods designed to prove that a given protocol is correct with respect to a specified security property. If the use of one of these methods results in a success (a proof of the correctness of the protocol has been built), the protocol can be marked as correct and no more analysis is required on this protocol. On the other hand, if the method fails to prove the protocol correct, we cannot conclude that it is flawed nor that it is correct. The general process of such methods is pictured in Figure 2.3. The main advantage of these approaches is that when we are successful in proving the correctness of a protocol, this protocol can thereafter be used with no problem. The major disadvantage is that a failure does not provide any information about the correctness of the protocol.

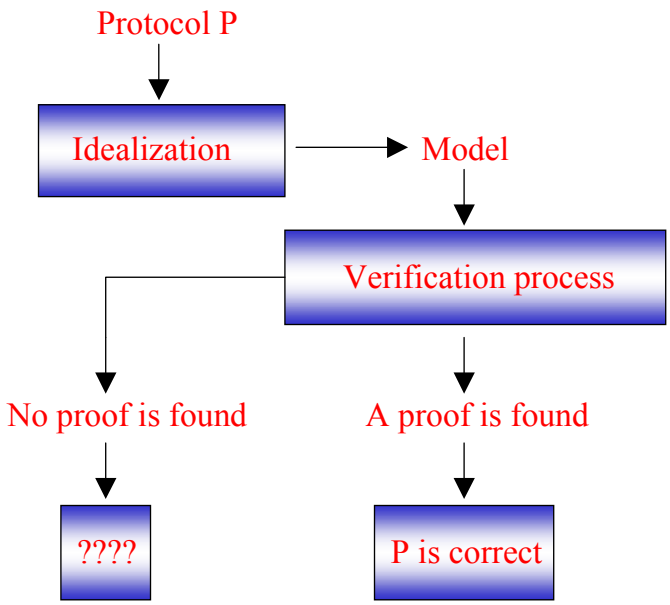

Figure 2.3: Proving correctness

2.3.1 Sufficient conditions for correctness

In [38, 39], *Houmani* proposes a set of sufficient conditions to assure the correctness of a protocol with respect to the secrecy property. Given a protocol, if it respects every condition then it is correct otherwise no conclusion may be drawn since these conditions are not necessary. Informally, the three conditions are:

- Every sensitive message exchanged during an execution of the protocol must be properly encrypted.
- An honest agent is not allowed to send an unknown message. However, an unknown message may be used as a key to encrypt another message.
- If the message $\{m\}_k$ is exchanged during the protocol, then k cannot be part of the message m.

The less restrictive are the conditions, the better is this kind of approach. Unfortunately, the conditions given by Houmani seem rather restrictive in the sense that very few existing protocol satisfy them (see [39] for a protocol satisfying these conditions). Another disadvantage of this approach is the incompatibility with other security properties. The conditions were designed to assure secrecy and cannot be used for other security properties. The main advantage of this result is its usefulness in protocol construction. To design a secure protocol (w.r.t. secrecy), one has only to build a protocol that respects the conditions. Very few approaches are useful in understanding and helping the, error prone, process of protocol construction. Furthermore, new insights are given about how to avoid forcing an honest agent to send an unknown message (remember that in the role generalization process, the more unknown messages are sent by an agent, the more powerful the intruder will be). Another advantage would be in the easiness of checking whether or not a given protocol satisfies the conditions.

2.3.2 Gavin Lowe approach

The approach introduced by *Lowe* in [48] uses model checking on a "small model" of the protocol to be analyzed. The "small model" has the property of being decidable for the secrecy property, that is, it is possible to decide whether or not a "small system" fulfills secrecy using model checking tools. Unfortunately, not every protocol can be transformed into a "small model". Moreover, the "small model" is not equivalent to the original protocol in the sense that if an attack occurs in the "small model", it is not necessarily present in the original protocol. The former problem is addressed by imposing restrictions on the protocol we analyze while the latter is said to occur on only very few occasions (but it may appear anyway thus disallowing the use of this technique as a decision procedure). The "small system" of a protocol is designed in such a way that if it is secure, with respect to secrecy, so is the original protocol. The conditions used to assure that a protocol is transformable in a "small model" do not seem to be very restrictive, however some modifications over most existing protocols

are required for them to meet those conditions. More details about the approach can be found in [49].

2.3.3 Stand space model

In [31, 33], *Thayer Fábrega, Herzog & Guttman* propose a graph-structured model for protocols called strand space. A strand corresponds to a sequence of events (an execution of the protocol or a sequence of action by an intruder). A strand space is a collection of strands equipped with a graph structure generated by the causal interaction between different strands. From the *Needham-Schroeder-Lowe* protocol (NSL for short) presented in Table 2.5, we illustrate how the strand space model can be used to prove the correctness of this protocol. NSL was proposed by *Lowe* to fix a flaw discovered in the original *Needham-Schroeder* protocol. NSL differs from the original version only in step 2 (B's name was not present in the original version). The simplified version of the protocol we use here presupposes that the principals have already exchanged their public key.

$$\begin{array}{ll} 1 & A \rightarrow B : \{N_a.A\}_{k_B} \\ 2 & B \rightarrow A : \{N_a.N_b.B\}_{k_A} \\ 3 & A \rightarrow B : \{N_b\}_{k_B} \end{array}$$

Table 2.5: NSL protocol

From the protocol, we first generate the strands. In this particular case, we have the initiator strand (corresponding to the agent A), the responder strand (agent B) and of course the intruder strands. A strand has the form $< \pm t_1, \pm t_2, ..., \pm t_n >$ where t_i are terms (messages) and the sign of a message stands for its direction: $-a$ means that the message a is received while $+b$ means that the message b is sent. The initiator strand from NSL is: $< +\{N_a.A\}_{k_B}, -\{N_a.N_b.B\}_{k_A}, +\{N_b\}_{k_b} >$ while the responder's one looks like: $< -\{N_a.A\}_{k_B}, +\{N_a.N_b.B\}_{k_A}, -\{N_b\}_{k_b} >$. The intruder's strands (modeling his abilities) are independent from the protocol and look like:

- M: $< +t >$ where $t \in T$
- F: $< -g >$
- T: $< -g, +g, +g >$

- C: $< -g, -h, +g.h >$
- S: $< -g.h, +g, +h >$
- K: $< +k >$ where $k \in K_p$
- E: $< -k, -h, +\{h\}_k >$
- D: $< -k^{-1}, -\{h\}_k, +h >$

Here T is a set of text messages that can be generated during the protocol and K_p is the set of keys initially known to the intruder.

A bundle is a directed acyclic graph formed by composing together some strands. A node in the graph is a message found in one of the strand and arrows are of two kinds: $n_1 \rightarrow n_2$ means that message n_1 is sent in a given strand s_1 and it is received in another strand s_2 at node n_2 so $n_1 = +t$ and $n_2 = -t$ thus creating a causal link between the two strands; $n_1 \Rightarrow n_2$ means that n_1 is followed immediately by n_2 on the same strand s. We use $n_1 \preceq_C n_2$ to denote the fact that there is a sequence of zero or more edges (of either kind) from n_1 to n_2 in a given bundle C. The relation $\preceq_C$ expresses causal precedence of messages in the bundle C, that is $n_1 \preceq_C n_2$ means n_1 must occur before n_2. As an example, we give in Table 2.6 the bundle representing the intended execution of the NSL protocol. Intuitively, a bundle may be seen as a valid trace as defined in Section 1.4.5. More formally, a bundle C is a set of edges such that N_C is the set of nodes incident with C and:

- C is finite;
- C is acyclic;
- if $n_1 \in N_C$ and n_1 is a negative node, then there is a unique n_2 such that $n_2 \rightarrow n_1 \in C$;
- if $n_1 \in N_C$ and $n_2 \Rightarrow n_1$ in the strand space, then $n_2 \Rightarrow n_1 \in C$.

$$
\begin{array}{ccc}
A & & B \\
+\{N_a.A\}_{k_B} & \longrightarrow & -\{N_a.A\}_{k_B} \\
\Downarrow & & \Downarrow \\
-\{N_a.N_b.B\}_{k_A} & \longleftarrow & +\{N_a.N_b.B\}_{k_A} \\
\Downarrow & & \Downarrow \\
+\{N_b\}_{k_B} & \longrightarrow & -\{N_b\}_{k_B}
\end{array}
$$

Table 2.6: NSL bundle

The proofs in the strand space model are inductive-like proofs that use the existence of a minimal member in every non-empty set. First, let's define a subterm relation: $t_1 \sqsubset t_2$ means that t_1 is a subterm of t_2. $\sqsubset$ is defined as:

- $a \sqsubset t$ for $t \in T$ iff $a = t$;
- $a \sqsubset k$ for a key k iff $a = k$;
- $a \sqsubset \{g\}_k$ iff $a \sqsubset g$ or $a = \{g\}_k$;
- $a \sqsubset g.h$ iff $a \sqsubset g$, $a \sqsubset h$ or $a = gh$.

Next we define some technical words. A term t *occurs* at node n iff $t \sqsubset n$; t *originates* at node n iff n is a positive node, t occurs at n and whenever n' precedes n on the same strand, $t \not\sqsubset n'$; and t is *uniquely* originating iff t originates at a unique node n. If a term t originates uniquely in a strand space, then it can be used as a fresh value like a nonce or a session key.

The inductive like proof of the strand space model are based on the following lemmas:

- Let C be a bundle, then every non-empty subset of the nodes in C has a $\preceq_C$-minimal members.
- Let C be a bundle and suppose S is a set of nodes such that $|m| = |m'|$[1] implies $m \in S$ iff $m' \in S$ for all nodes m, m'. If n is a $\preceq_C$-minimal member of S, then n is a positive node.
- Let C be a bundle, t a term and $n \in C$ a $\preceq_C$-minimal members of $\{m \in C | t \sqsubset m\}$. t originates at node n.

These technical lemmas, see [31] for their proofs, serve as basic tools to get knowledge about minimal element of a set. We will see how to use them in the process of proving a protocol correct. As an example, we show that the NSL protocol of Table 2.5 is correct with respect to the secrecy of the responder nonce, that is N_b is confidential. First we need the following assumptions:

- Σ is a NSL strand space and C is a bundle containing the responder's strand s on which lies the nonce N_b;

[1] Let m be a node, $|m|$ stands for the message used at node m regardless of its direction (send or receive).

- $N_a \neq N_b$ and N_b is uniquely originating in Σ;
- $k_a^{-1}, k_b^{-1} \notin K_p$, otherwise the protocol is trivially flawed.

The correctness proof will be stated as follows: for all nodes $n \in C$ such that $N_b \sqsubset n$, we have either $\{N_a.N_b.B\}_{k_A} \sqsubset n$ or $\{N_b\}_{k_B} \sqsubset n$. If this holds, clearly the intruder can never learn the value of N_b. But if it doesn't hold, it is not trivial to conclude that the protocol is flawed. The proposition stated above is equivalent to the emptiness of the following set:

$$S \quad = \quad \{n \in C | N_b \sqsubset n \text{ and } \{N_a.N_b.B\}_{k_A} \not\sqsubset n \text{ and } \{N_b\}_{k_B} \not\sqsubset n\}$$

The idea of the proof goes like this (see [31] for the full proof): suppose S is non-empty, then it has at least one $\preceq_C$-minimal member, say m, and m is positive (this is a straightforward consequence of the lemmas presented earlier). We first need to show that m cannot lie on a regular strand (that is a strand of a legitimate agent, initiator or responder in our case) and then that it cannot lie on an intruder strand. Thus, if such a node cannot lie on a strand of the protocol, it cannot exist in the current strand space, which is a contradiction with non-emptiness of S.

The main drawback of this method is that if we cannot prove the statement for the correctness of the protocol, then we cannot conclude that the protocol is flawed. Even if we disprove the statement, it is not necessarily the case that a flaw can be found in the protocol. Moreover, the generation of the statement to be proved is not automatic and thus is error prone (in cases more complex than our example, it may not be trivial whether or not proving the statement implies proving the correctness of the protocol). However, this approach has many advantages:

- proofs seem easy and intuitive enough to be made by humans;
- this technique gives some insight about the assumptions required for a protocol to be correct;
- some general theorems may be proved on the bound of the intruder abilities (see [33]);
- although not presented by *Thayer Fábrega* & al. some automatization of this model is possible as proposed by *Song* in a tool called ATHENA, based on the strand space model (see [81]);

- the authors claim that it is interesting to have a specific statement to prove depending on the protocol and the security property considered rather than have a general statement for all protocols.

For more relevant information about the strand space model, see [30, 32].

2.4 Deciding correctness

Few approaches have been developed with the objective to decide about the correctness of protocols. Since the problem of telling whether a given protocol respects a given security property is undecidable (see [15, 27, 29]), one cannot devise a general procedure to decide the correctness of any protocol. Instead, one can hope to restrict the set of protocols to a subset for which it may be possible to decide about their correctness, see Figure 2.4. Given a set of conditions, if a given protocol respects those conditions, then we are able to decide about its correctness. To be interesting, such an approach must be the least restrictive as possible, that is we must be able to decide the correctness for a wide interesting range of protocols. We don't give any example of such an approach here, but we reserve the entire Chapter 3 for a technique developed by *Mejri* [57] that is close to achieve this. We chose to spend an entire chapter on this particular method since the main part of this thesis is based on it: addressing its fundamental problem. Note that the result we present in Chapter 5 falls under this category.

2.5 Others

Here we mention some other methods without trying to classify them and we present in more details one of the first techniques in protocol analysis, the BAN logic. *Lowe* was one of the first to use process algebra (CSP [37]) to specify and analyze cryptoprotocols, many others followed including *Abadi & Gordon* with the spi-calculus [1] (an extension of the pi-calculus [61]) and *Milner* with CCS [2]. *Meadows* used the NRL protocol analyzer, see [54], to find flaws in some protocols. This methods aims to do an exhaustive search of the state space that a protocol can be in and check if there exists a state where the property is violated. The inductive approach of *Paulson* (see [66, 67]) is one of the most famous using automatic theorem provers to generate a proof of correctness. Although this approach has provided very good results, it suffers from two problems: first the protocol idealization is not yet an automatic step (transforming the original

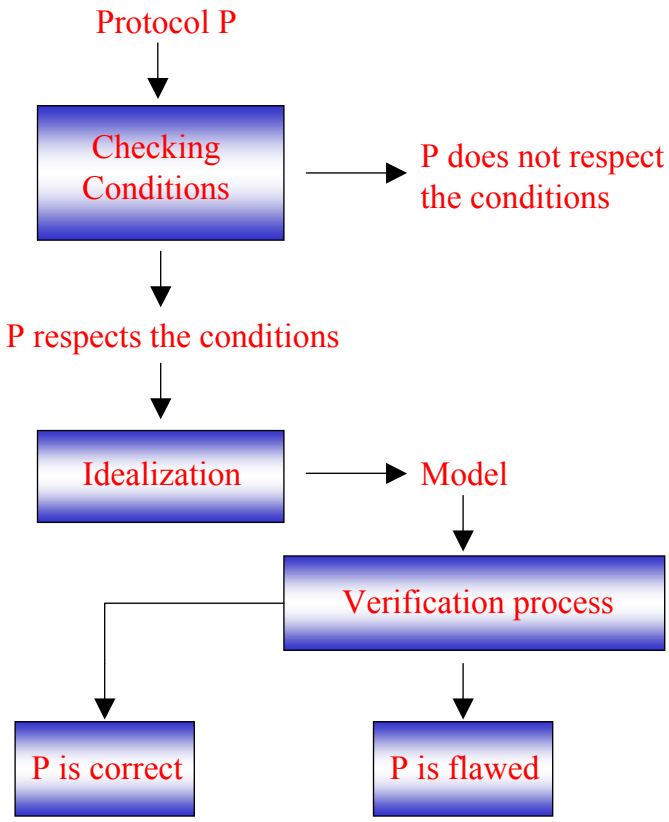

Figure 2.4: Deciding correctness

protocol into the formal model), it requires specialized human intervention and thus is error prone; second, it surely inherits the general non-termination problem of the automatic theorem provers. Some formal methods address the generation of protocols rather than the analysis. These methods (some of them can be found in [68, 73, 74, 82]) aim to generate a protocol that respects a given set of security properties. Most of the works done in protocol analysis assumes that protocols are independent, that is they do not interact. But in real applications, there is often many protocols in a shared environment (they share keys, a trusted server, ...) and is it not possible anymore to analyze the protocols independently. A new field of research is dedicated to the composition of protocols. It aims to study the interaction between protocols that are composed (they may run in parallel, one after the other or interlaced). Some of the methods addressing composition can be found in [36, 41, 53].

2.5.1 BAN logic

One of the first formal methods used in the analysis of cryptographic protocol is the BAN logic (from *Burrows, Abadi & Needham*). BAN models the beliefs of the participants in a protocol and the evolution of these beliefs after messages are exchanged. It is mostly dedicated to the verification of the authentication property. The syntax of this logic goes as follows:

- $P \models X$: principal P believes that X is true.
- $P \triangleleft X$: P sees X (someone sent the message X to P).
- $P \vdash X$: P said X (P sent the message X to someone).
- $P \mid\Rightarrow X$: P rules over X (when a server is required to generate keys, we say he rules over the keys).
- $\#X$: X is a fresh message.
- $P \leftrightarrow^k Q$: the key k is shared between P and Q.
- $\mapsto^k P$: P has a public key.
- $P \leftharpoondown^X Q$: P and Q share the secret X.
- $\{X\}_k$: message X is encrypted under the key k.
- $\langle X \rangle_Y$: message X combined with message Y (it is assumed that Y is a secret message and it proves the identity of the sender of message $\langle X \rangle_Y$).

Together with the syntactic constructions comes a set of rules to manipulate beliefs:

R_1: $\frac{P \models Q \leftrightarrow^k P \quad P \triangleleft \{X\}_k}{P \models Q \vdash X}$ R_2: $\frac{P \models \mapsto^k Q \quad P \triangleleft \{X\}_{k-1}}{P \models Q \vdash X}$ R_3: $\frac{P \models P \rightleftharpoons^Y Q \quad P \triangleleft \{X\}_Y}{P \models Q \vdash X}$

R_4: $\frac{P \models \#X \quad P \models Q \vdash X}{P \models Q \models X}$ R_5: $\frac{P \models Q \mid\Rightarrow X \quad P \models Q \models X}{P \models X}$ R_6: $\frac{P \models Q \vdash (X,Y)}{P \models Q \vdash X}$

R_7: $\frac{P \models X \quad P \models Y}{P \models (X,Y)}$ R_8: $\frac{P \models (X,Y)}{P \models X}$ R_9: $\frac{P \models Q \models (X,Y)}{P \models Q \models X}$

R_{10}: $\frac{P \triangleleft (X,Y)}{P \triangleleft X}$ R_{11}: $\frac{P \triangleleft \langle X \rangle_Y}{P \triangleleft X}$ R_{12}: $\frac{P \models Q \leftrightarrow^k P \quad P \triangleleft \{X\}_k}{P \triangleleft X}$

R_{13}: $\frac{P \models \mapsto^k P \quad P \triangleleft \{X\}_k}{P \triangleleft X}$ R_{14}: $\frac{P \models \mapsto^k Q \quad P \triangleleft \{X\}_{k-1}}{P \triangleleft X}$ R_{15}: $\frac{P \models \#X}{P \models \#(X,Y)}$

R_{16}: $\frac{P \models R \leftrightarrow^k R'}{P \models R' \leftrightarrow^k R}$ R_{17}: $\frac{P \models Q \models R \leftrightarrow^k R'}{P \models Q \models R' \leftrightarrow^k R}$ R_{18}: $\frac{P \models R \rightleftharpoons^X R'}{P \models R' \rightleftharpoons^X R}$

R_{19}: $\frac{P \models Q \models R \leftrightarrow^k R'}{P \models Q \models R' \leftrightarrow^k R}$

Table 2.7: BAN rules

Given the *Needham-Schroeder* protocol of Table 1.12, we check if it achieves the following goals which represent the bidirectional authentication:

- G_1: $A \models A \leftrightarrow^{k_{ab}} B$
- G_2: $A \models B \models A \leftrightarrow^{k_{ab}} B$
- G_3: $B \models A \leftrightarrow^{k_{ab}} B$
- G_4: $B \models A \models A \leftrightarrow^{k_{ab}} B$

The first thing to do is to transform the protocol into its idealized form (a set of BAN's formulas). The idealization process works in this way, for every step of the protocol:

- non-crypted messages are removed.
- every part of an encrypted message that are not relevant for the evolution of knowledge are also removed.

- the intended meaning of this step is formalized with the syntax of BAN logic.

Finally, we enumerate a set of initials hypothesis that are written in the BAN syntax. The idealization cannot be done in a step-by-step way since a global knowledge of the entire protocol is required to discover the intended meaning of messages. The idealization process is not automatic and is ambiguous (as stated in [47]), in particular with the generation of the hypothesis, and thus it is very error prone. For example, the idealized version of the *Needham-Schroeder* protocol of Table 1.12 is presented in Table 2.8 together with some initial hypothesis (H_1 to H_{11}).

1	$A \rightarrow S :$
2	$S \rightarrow A : \{N_a, A \leftrightarrow^{k_{ab}} B, \#(A \leftrightarrow^{k_{ab}} B)\{A \leftrightarrow^{k_{ab}} B\}_{k_{bs}}\}_{k_{as}}$
3	$A \rightarrow B : \{A \leftrightarrow^{k_{ab}} B\}_{k_{bs}}$
4	$B \rightarrow A : \{N_b, A \leftrightarrow^{k_{ab}} B\}_{K_{ab}}$
5	$A \rightarrow B : \{N_b, A \leftrightarrow^{k_{ab}} B\}_{K_{ab}}$

H_1: $A \models S \mid\Rightarrow A \leftrightarrow^{k_{ab}} B$	H_2: $A \models \#N_a$	H_3: $A \models A \leftrightarrow^{k_{as}} S$
H_4: $A \models S \mid\Rightarrow \#(A \leftrightarrow^{k_{ab}} B)$	H_5: $B \models \#N_b$	H_6: $B \models B \leftrightarrow^{k_{bs}} S$
H_7: $B \models S \mid\Rightarrow A \leftrightarrow^{k_{ab}} B$	H_8: $S \models A \leftrightarrow^{k_{as}} S$	H_9: $B \models B \leftrightarrow^{k_{bs}} S$
H_{10}: $S \models A \leftrightarrow^{k_{ab}} B$	H_{11}: $S \models \#(A \leftrightarrow^{k_{ab}} B)$	

Table 2.8: *Needham-Schroeder* protocol idealized

The idealization is far from being straightforward. We give here the main ideas of idealizing the *Needham-Schroeder* protocol, see [13] for a more detailed explanation. At step 2, since A knows that N_a is a fresh message, then by using rule R_{15}, he can deduce that the key k_{ab} is also fresh. The goal of steps 4 and 5, is to convince B that A has the shared key k_{ab}, this is why it appears in the idealization. Most of the hypotheses concern the fact that principals share some keys and that they use nonces in the protocol. H_1, H_4 and H_7 represent the fact that S is trusted to produce good keys. From the idealization of our protocol, we can start to show the four goals required to hold for the authentication to be respected.

G_1: $A \models A \leftrightarrow^{k_{ab}} B$:

Starting from message of step 2, using H_3 and R_1, we can deduce:

$$A \models S \vdash (N_a, A \leftrightarrow^{k_{ab}} B, \#(A \leftrightarrow^{k_{ab}} B), \{A \leftrightarrow^{k_{ab}} B\}_{k_{bs}}) \tag{2.2}$$

Applying H_2 and R_{15} to equation 2.2 gives us:

$$A \models \#(N_a, A \leftrightarrow^{k_{ab}} B, \#(A \leftrightarrow^{k_{ab}} B), \{A \leftrightarrow^{k_{ab}} B\}_{k_{bs}}) \tag{2.3}$$

Using 2.2 together with 2.3 through R_4 we get:

$$A \models S \models (N_a, A \leftrightarrow^{k_{ab}} B, \#(A \leftrightarrow^{k_{ab}} B), \{A \leftrightarrow^{k_{ab}} B\}_{k_{bs}}) \tag{2.4}$$

Applying R_9 to 2.4 we deduce:

$$A \models S \models A \leftrightarrow^{k_{ab}} B \tag{2.5}$$
$$A \models S \models \#(A \leftrightarrow^{k_{ab}} B) \tag{2.6}$$

Finally from H_1, 2.5 and R_5 it follows that:

$$A \models A \leftrightarrow^{k_{ab}} B \tag{2.7}$$

□

G_2: $A \models B \models A \leftrightarrow^{k_{ab}} B$:

Starting from 2.6, using H_4 and R_1, we can deduce:

$$A \models \#(A \leftrightarrow^{k_{ab}} B) \tag{2.8}$$

Applying R_{15} to 2.8 we get:

$$A \models \#(N_b, A \leftrightarrow^{k_{ab}} B) \tag{2.9}$$

From message 4 and H_3, R_1 takes us to:

$$A \models B \vdash (N_b, A \leftrightarrow^{k_{ab}} B) \tag{2.10}$$

Using both 2.9 and 2.10 in rule R_4 brings us to:

$$A \models B \models (N_b, A \leftrightarrow^{k_{ab}} B) \tag{2.11}$$

Finally, from 2.11, using R_9 we establish that:

$$A \models B \models A \leftrightarrow^{k_{ab}} B \tag{2.12}$$

□

In trying to prove G_3 and G_4, one will fail since, as explained in [13], the protocol is correct only under the "strange" assumption that B believes the key k_{ab} to be fresh ($B \models \#(A \leftrightarrow^{k_{ab}} B)$). This assumption is important since the protocol does not provide B with a means to assure the freshness of the key. Whether or not we can add this assumption to our set of initial hypothesis without changing the meaning of the protocol is a non-trivial question. Although the process of proving a protocol correct is a tedious, error prone and ambiguous one, the BAN logic has been widely used to compare new formal methods to existing ones, to study the intended behavior of some protocols. Many other logics were designed around the BAN ideas, trying to improve this technique. Although it is not very well suited for automatic protocol analysis, BAN has been, and still is, very useful for understanding protocols. In addition to the authentication property, BAN is used to learn:

- What a protocol can accomplish?
- Which hypotheses are required for a protocol?
- Which actions in the protocol are superfluous?
- Is there any encrypted message that could be sent non-encrypted?

2.6 Conclusion

In this chapter, we saw that many formal methods have been developed for the analysis of cryptographic protocols. Each method brought some insight about protocol analysis or protocol design. Where it was once extremely relevant to produce and attack when the protocol was flawed, it is now important to assure the correctness of a protocol. One does not need an attack against a protocol when it is flawed, but instead wants to know if a protocol is secure or not. This is why the methods addressing the decision problem regarding the correctness of protocols are becoming increasingly interesting.

Chapter 3

On the Termination of Inference Proof Systems

Abstract

In the previous chapter, while studying how to analyze cryptoprotocols using the DYMNA approach, we stumbled upon the problem of decidability of an inference system. This problem prevents DYMNA from being a decision procedure for the correctness of protocols. This chapter addresses precisely this decidability problem.

3.1 Introduction

This chapter studies the decidability problem of inference systems such as done by *Mejri* in [57]. *Mejri* is not the only one to address the decidability (also called termination) problem of inference systems in the context of cryptographic protocols; *Blanchet* [9, 10] and *Kindreed & Wing* [42, 43] propose similar techniques. However, since our work is directly based on the results from *Mejri*, the emphasis will be placed on his approach.

This chapter is divided in six main sections: Section 3.2 introduces inference systems; Section 3.3 provides a way allowing us to be sure that the theory of an inference system is decidable; Section 3.4 relates how *Mejri* proposes to resolve non-termination of inference systems; Section 3.5 describes an algorithmic version of the transformation provided by *Mejri*; Section 3.6 gives an example of applying the transformation algorithm of an

inference system; finally, Section 3.7 gives an overview of the methods used by *Blanchet* and *Kindreed & Wing*.

3.2 Inference systems

An inference system is a set of rules where each has a set of premises $p_1, p_2, \ldots, p_n$ and a conclusion c. We use $\mathcal{P}(R)$ and $\mathcal{C}(R)$ to denote respectively the set of premises and the conclusion of R. Sometimes, we use $\mathcal{P}_i(R)$ for the i^{th} premise of R. Note that rules are not allowed to have infinitely many premises. When the rule has no premise ($n = 0$), we call it an axiom and denote it by $R' = \frac{\square}{c'}$. It is well known that an inference system with no axiom has an empty theory. The theory of an inference system S, noted $TH(S)$, is the set of all formulae provable in S. We use IS to denote the set of inference systems. To test if a formula belongs to a given inference system's theory, one traditionally uses either backward chaining or forward chaining techniques on the rules of the inference system.

In backward chaining, we start from a formula f and apply a rule R such that f and $\mathcal{C}(R)$ are unifiable[1] and have σ as their most general unifier. Now the proof of f can be transformed in the separate proofs of $\{p\sigma | p \in \mathcal{P}(R)\}$. If we are able to prove all these sub-formulae, then we can conclude that f is indeed provable. Of course, more then one rule may be applied to prove a given formula and every rule must be taken into account if we want to decide whether or not f is provable. Given an inference system S, it is not necessarily the case that backward chaining will terminate while trying to prove a term. To illustrate this, suppose we take the following set of rules S to be our inference system where x_i are variables, a is a constant and f a unary function.

$$S = \{R_1 = \frac{f(f(x_1))}{f(x_1)}, R_2 = \frac{x_2}{f(f(f(x_2)))}, R_3 = \frac{\square}{a}\} \tag{3.1}$$

It is obvious that the term $f(b)$ does not belong to the theory of S. However, using backward chaining leads to explore infinitely many new terms and thus does not terminate.

While proving f in the inference system S using forward chaining, we start from the axioms and generate the whole theory of S in a step by step manner. After each step, we check if f belongs to the partial theory of S (the theory built so far). A step consist of taking every rule $R \in S$, every n-tuple t of elements in the partial theory of S (where

[1] Two terms m and m' are unifiable if there exists a substitution σ such that $m\sigma = m'\sigma$. More about unification can be found in Section 4.3

n is the number of premises for the rule R) and adding $\sigma(\mathcal{C}(R))$ to the partial theory of S (if such a $\sigma(\mathcal{C}(R))$ exists). σ is taken to be mgu[2]$(t_1, \mathcal{P}_1(R)) \circ mgu(t_2, \mathcal{P}_2(R)) \circ \ldots \circ mgu(t_n, \mathcal{P}_n(R))$. As backward chaining, forward chaining does not always terminates.

Since backward chaining acts in a goal oriented manner, it is often used instead of forward chaining. From now on, we are interested only in backward chaining (or a slight modification of backward chaining).

3.3 Termination of inference systems

Definition 3.3.1 (Backward terminating inference system)
Consider t as being any well-formed term. An inference system S is backward terminating when it is not possible to loop indefinitely while using backward chaining to prove test the membership of t in $Th(S)$.

Since we use only backward chaining, terminating (resp. non-terminating) will be used as a synonym for backward terminating (resp. backward non-terminating).

Definition 3.3.2 (Well-founded ordering)
A partially strict-ordered set[3] *$(S, \prec)$ is said to be well-founded if there are no infinite descending sequences $\ldots \prec s_3 \prec s_2 \prec s_1$ of elements in S.*

Example 3.3.1 (Well-founded ordering)
The natural numbers $\mathbb{N}$ under there natural ordering $>$ are well-founded since no sequence of natural numbers can descend beyond 0. But $>$ is not a well-founded ordering over the entire set of $\mathbb{Z}$, since $0 > -1 > -2 > \ldots$ is an infinite descending sequence.

Definition 3.3.3 (Termination ordering)
A well-founded ordering $\prec$ is a termination ordering if it respects the following two conditions (see [22, 23] for more information of termination orderings):

- *Monotone:* $(t_1 \prec t_2) \Rightarrow (t\sigma_1 \prec t\sigma_2)$ *where* $\sigma_1 = \{X \leftarrow t_1\}$, $\sigma_2 = \{X \leftarrow t_2\}$
- *Preserved by substitution:* $(t_1 \prec t_2) \Rightarrow t_1\sigma \prec t_2\sigma$

[2] $mgu(t_1, t_2)$ is the most general unifier σ of t_2 and t_2. That is, for any unifier θ of t_1 and t_2, there is a substitution θ' such that $\sigma\theta' = \theta$

[3] A set S is partially strict-ordered if there is a transitive, irreflexive and antisymmetric binary relation $\prec$ defined on the elements of S

We can easily show that an inference system S is terminating if there is a terminating ordering $\prec$ such that for every rule $R \in S$ and for every $p \in \mathcal{P}(R)$, $p \prec \mathcal{C}(R)$ holds.

3.4 Handling termination

The idea proposed by *Mejri* to handle the termination problem of inference system is to transform a non-terminating inference system S into an "equivalent" (see Definition 3.4.1 $S \equiv S'$) terminating one S'. To achieve this, the rules of S are partitioned into two sets: those that cause a problem from the point of view of termination (non-oriented rules or non-terminating rules) and oriented rules (or terminating rules). Then, new rules are generated, using a composition operator (see Definition 3.4.2) and added to the inference system in the proper set. Finally, redundant rules are eliminated (see Definition 3.4.4). The process is repeated until a fixed point is reached; the current set of oriented rules is then equivalent to the original inference system and does not cause any termination problems. In the following, this approach is detailed.

Definition 3.4.1 (Comparison of inference systems)
Let S_1 and S_2 be two inference systems:

- $S_1 \leq S_2$ *iff* $\forall$ $Th \in TH(S_1)$, *it is the case that* $Th \in TH(S_2)$.
- $S_1 \geq S_2$ *iff* $S_2 \leq S_1$.
- $S_1 \equiv S_2$ *iff* $S_1 \leq S_2$ *and* $S_1 \geq S_2$.

Definition 3.4.2 (Rule composition)
Let R be an inference rule with n premises. Let $(R_1, R_2, \ldots, R_n)$ be a n-tuple of inference rules. We define the composition of R with $(R_1, R_2, \ldots, R_n)$, denoted by $R \Uparrow (R_1, R_2, \ldots, R_n)$, to be the following inference rule:

$$\frac{\sigma(\mathcal{P}(R_1)) \quad \sigma(\mathcal{P}(R_2)) \quad \ldots \quad \sigma(\mathcal{P}(R_n))}{\sigma(\mathcal{C}(R))}$$

where $\sigma = cmgu$[4]$((\mathcal{P}_1(R), \mathcal{C}(R_1)), (\mathcal{P}_2(R), \mathcal{C}(R_2)), \ldots, (\mathcal{P}_n(R), \mathcal{C}(R_n)))$. The composition is defined only if such a σ exists.

The definition of rule composition above is extended to compose two sets of inference rules in the following way.

[4] *cmgu* should be seen a the most general unifier of many pair of terms. See [57] for details about cmgu

Definition 3.4.3 (Set of rules composition)
Let $S_1 = \{R_1^1, R_2^1, \ldots, R_{n_1}^1\}$ and $S_2 = \{R_1^2, R_2^2, \ldots, R_{n_2}^2\}$ be two sets of inference rules. We define the composition of each rule in S_2 with rules in S_1, written $S_1 \looparrowright S_2$, by:

$$S_1 \looparrowright S_2 = S_1 \cup S_2 \cup \Big(\bigcup_{R_i^2 \in S_2} \big(\bigcup_{t \in T(S_1, |\mathcal{P}(R_i^2)|)} \{R_i^2 \Uparrow t\} \big) \Big)$$

where $T(S, n)$ is the set of all n-tuple built over the element of S and $\{R_i^2 \Uparrow t\} = \emptyset$ whenever $R_i^2 \Uparrow t$ is not defined.

Definition 3.4.4 (Simplified inference system)
Let S be an inference system, we define the simplified inference system associated with S, written $S_{\Downarrow}$ (that is the inference system S with the redundant rules removed), as follows: let $R \in S$, then $R \in S_{\Downarrow}$ if the following two conditions hold simultaneously:

- *No trivial rules: $\forall p \in \mathcal{P}(R), p \neq \mathcal{C}(R)$*
- *No instantiated rules: there is no rule $R' \in S, (R' \neq R)$ with a substitution σ such that: $\sigma(\mathcal{P}(R')) \subseteq \mathcal{P}(R)$ and $\sigma(\mathcal{C}(R')) = \mathcal{C}(R)$.*

Definition 3.4.5 (Series of inference systems)
Let S be an inference system. We partition S in $S_{\prec}$, the set of terminating rules, and $S_{\succcurlyeq}$ the remaining rules (the non-terminating ones). We associate to S a series of inference system originating from S, designed by $(S^n)_{n \geq 0}$, as follows:

- $S^0 = S$
- $S^n = (S_{\prec}^{n-1} \looparrowright S_{\succcurlyeq}^{n-1})_{\Downarrow} \;\; n \geq 1$

Definition 3.4.6 (Series convergence)
We say the series $(S^n)_{n \geq 0}$ converges when there is a k such that $S^k = S^{k+1}$. We call S^k a fixed point of the series.

The next two results about the composition procedure, provided by *Mejri*, are very important. Lemma 3.4.1 states that the theory of an inference system is invariant under the composition of its rules. Theorem 3.4.1 states that if no new rule is generated by composing a non-oriented rule with oriented rules, then the current set of oriented rules is equivalent to the whole inference system.

Lemma 3.4.1
Let S_1 and S_2 be two sets of inference rules. We have that $S_1 \looparrowright S_2 \equiv S_1 \cup S_2$.

The idea behind this lemma (see [57] for full proof) is as follows. Let $S = S_1 \cup S_2$ and $S' = S_1 \looparrowright S_2$. $S \leq S'$: since $S \subseteq S'$. $S' \leq S$: suppose instead that $Th \in TH(S')$ but $Th \notin TH(S)$, it must be the case that the proof of Th in S' uses at least a rule $R \notin S$. But R is the result of combination of rule in S, say $R = R' \Uparrow (R_1, R_2, \ldots, R_n)$, thus the same proof holds in S' replacing R by the sequence composing R' with $(R_1, R_2, \ldots, R_n)$. Once all rules such as R are replaced by the composition of other rules in S, we have a proof that $Th \in TH(S')$ using only rules of S. We conclude that $Th \in TH(S)$ also holds. Having $S \leq S'$ and $S' \leq S$, it follows that $S_1 \equiv S_2$.

Theorem 3.4.1
Let $S = S_1 \cup S_2$ be an inference system such that no rule in S_2 is an axiom. If $(S_1 \looparrowright S_2)_{\Downarrow} = (S_1 \cup S_2)_{\Downarrow}$ then $S \equiv S_1$.

The idea behind the theorem is the following: suppose a fixed point S^k is reached in the series, then the combination of any rule R in $S^k_{\succcurlyeq}$ with rules in $S^k_{\prec}$ gives a rule that is redundant in S^k. Since all the combinations of R with other rules are made redundant, R is useless; its appearance in a proof can always be replaced. So we can safely remove R from $S^k_{\succcurlyeq}$ without changing the underlying theory. Since every rule of $S^k_{\succcurlyeq}$ can be removed in such a way, clearly $S^k \equiv S^k_{\prec}$.

From these results, we express two corollaries that are related to the series of inference systems of Definition 3.4.5. First, we can see that all inference systems of a series $(S^n)_{n\geq 0}$ are equivalent and in particular they are equivalent to their generator[5] S (see Corollary 3.4.1). Second, if S^k is a fixed point of the series, then $S \equiv S^k_{\prec}$ (see Corollary 3.4.2).

Corollary 3.4.1
Let S be an inference system and $(S^n)_{n\geq 0}$ its associated series. It follows directly from Lemma 3.4.1 and from the Definition 3.4.5 that $S = S^0 \equiv S^1 \equiv S^2 \equiv \ldots \equiv S^n \equiv \ldots$.

Corollary 3.4.2
Let S be an inference system and $(S^n)_{n\geq 0}$ its associated series. Suppose S^k is a fixed point, then it follows from Theorem 3.4.1 that $S^k \equiv S^k_{\prec}$ and from Corollary 3.4.1 that $S \equiv S^k_{\prec}$ and thus we have an inference system $S^k_{\prec}$ equivalent to S such that $S^k_{\prec}$ has no termination problem.

If the series associated to S converges, we can then compute a terminating inference system S' that is equivalent to S. S' can be used to decide if $t \in TH(S')$ or not and we know, from the equivalence of S and S', that $t \in TH(S') \Leftrightarrow t \in TH(S)$. Unfortunately, the series does not necessarily converge. As a result, we cannot directly

[5]The generator of a series $(S^n)_{n\geq 0}$ is S^0, the inference system to which the series is associated

use the transformation explained above (and pictured in algorithm of Table 3.1) to solve the non-termination problem underlying the inference system generated by the DYMNA approach (see Section 2.2.1). Note that an inference system S extracted from a protocol, as it is done by DYMNA, is always non-terminating due to the intruder's rules of message deconcatenation and decryption. The convergence (or non-convergence) of the series associated with S will be discussed in Chapter 4. *Mejri* provides a way to assure that the series associated to an inference system will be convergent; it is stated in Lemma 3.4.2.

Lemma 3.4.2 (Proving the convergence of a series)
Let S be an inference system and $(S^n)_{n\geq 0}$ the series associated with S. The series $(S^n)_{n\geq 0}$ converges iff there is $t_0 \in \mathcal{T}$ and a well-founded ordering relation $\preccurlyeq$ such that:

$$\forall n \geq 0,\ R \in S^n \Rightarrow \forall t \in \mathcal{T}(R) \text{ we have } t \preccurlyeq t_0$$

where $\mathcal{T}$ is the set of all terms that are syntactically well-formed (from the point of view of the algebra concerned) and $\mathcal{T}(R)$ the set of all terms forming the rule R.

Proof:
See the proof 5.6.2 in [57] p. 76.

□

Before we continue, it is important to understand that given an inference system S, it is not necessarily the case that the transformation process will converge when applied to S. It could be the case that two rules R_1, R_2 of S can be composed to form a new rule $R_3 = R_1 \uparrow R_2$ such that R_1 can be composed with R_3 to form R_4, and so on. Example 3.4.1 shows an inference system on which such a problem arise. This is a simple case of non-convergence, but it may be the case that a much more complicated pattern leads to non-convergence.

Example 3.4.1 (A case of non-convergence)
Let S be an inference system containing two rules R_1 and R_2 defined as:

$$R_1 = \frac{\square}{\{X_1.m.m'.m_1.Y_1\}}$$
$$R_2 = \frac{\{X_2.m_2.m.Y_2\}}{\{X_2.m.m_2.Y_2\}}$$

where we use X_i, Y_i to denote variables while m, m', m_i denote constants. $\{m\}$ models the encryption while $m.m'$ models messages concatenation (we suppose here that concatenation is associative). With $\sigma_3 = \{X_1 \rightarrow X_2.m_2, Y_2 \rightarrow m'.m_1.Y_1\}$, we obtain, from

$R_2 \Uparrow R_1$ after renaming the variables, the following rule:

$$R_3 \quad = \quad \frac{\square}{\{X_3.m.m_2.m'.m_1.Y_3\}}$$

Now, with $\sigma_4 = \{X_3 \rightarrow X_2.m_2, Y_2 \rightarrow m_2.m'.m_1.Y_3\}$, we obtain, from $R_2 \Uparrow R_3$ after renaming the variables, the following rule:

$$R_4 \quad = \quad \frac{\square}{\{X_4.m.m_2.m_2.m'.m_1.Y_4\}}$$

finally, with $\sigma_i = \{X_{i-1} \rightarrow X_2.m_2, Y_2 \rightarrow (m_2)^{i-3}.m'.m_1.Y_{i-1}\}$, we obtain, from $R_2 \Uparrow R_{i-1}$ after renaming the variables, the following rule:

$$R_i \quad = \quad \frac{\square}{\{X_i.m.(m_2)^{i-2}.m'.m_1.Y_i\}}$$

To prove that the transformation process does not converge for a given inference system S one need only to show that the algorithm would generate infinitely many new rules. However, to prove that the transformation converges for S is much harder. We will prove it, in the next chapter, by showing that the set of rules that can be generated from S using the rule composition process is finite.

3.5 Algorithmic transformation

We give here an algorithmic version of the transformation schema developed by *Mejri* as explained in Section 3.4.

The main function *TransformInferenceSystem*(S) depicted in Table 3.1 computes the series associated with the inference system S as in Definition 3.4.5. Once a fixed point is reached, if it is the case that the series converges, the algorithm returns an inference system equivalent to S but for which none of the rule causes a termination problem. Given a set of rules S, there is possibly many different subsets S' of S such that no rule in S' causes a termination problem. This is the reason why we do not provide here an algorithm for computing the subset of terminating rule from a set of rules. However, note that the axioms must all be considered as terminating rules. The other auxiliary functions are presented in Table 3.2 and explained below.

TransformInferenceSystem(S):
  $S = \mathit{EliminateRedundantRules}(S)$
  $S' = S$
  DO
    $S = S'$
    $S_{\prec} = \mathit{GetTerminatingRules}(S)$
    $S_{\succcurlyeq} = S \backslash S_{\prec}$
    $S' = \mathit{ComposeSetOfRules}(S_{\prec}, S_{\succcurlyeq})$
    $S' = \mathit{EliminateRedundantRules}(S')$
  WHILE ($S \neq S'$)
  RETURN $\mathit{GetTerminatingRules}(S)$

Table 3.1: The transformation algorithm: Part I

The function $\mathit{EliminateRedundantRules}(S)$ will remove the useless rules from S, thus computing $S_{\Downarrow}$ as introduced in Definition 3.4.4. It uses the predicate $\mathit{IsNotSelfRedundant}(R)$ that returns true iff R is not trivial (the conclusion of R is not in its premises) and the test $\mathit{IsMadeRedundantBy}(R, R')$ that returns true whenever R is a particular instance of R' (that is R' is more general than R).

The function $\mathit{ComposeSetOfRules}(S, S')$ computes the composition operator $S \looparrowright S'$ of Definition 3.4.3. It uses the function $\mathit{GetAllTuples}(S, n)$ which returns a set containing all n-tuples formed over the elements of S. Note that a tuple may contain multiple occurrences of the same element, hence there is exactly m^n n-tuples return by $\mathit{GetAllTuples}(S, n)$ where $m = |S|$. The last consideration about the tuple generation function is that all rules in all tuples must be freshly renamed to avoid variable clashes. That is, the rules appearing in a tuple are not the exact rule of the inference system, but instances of these rules where variables are renamed. Moreover, two elements of a same, or a different, tuple must share no variable. Although we are not always explicit in treating this last consideration, since it is of little theoretical importance, one must remains careful not to end up with linking variables where it should not be the case.

Finally, the function $\mathit{ComposeRule}(R, t)$ is the realization of the $R \Uparrow t$ operator (see Definition 3.4.2) where t in a n-tuple of rules and $n = |\mathcal{P}(R)|$. It uses the test $\mathit{Unifiable}(m, m')$ to check whether or not there is a substitution σ such that $m\sigma = m'\sigma$. If such a substitution exists, it uses $\mathit{FindMgu}(m, m')$ to find the most general

EliminateRedundantRules(S):
  $S' = \emptyset$
  FOR ALL $R \in S$
    IF(*IsSelfRedundant*(R))
      isRedundant = TRUE
    ELSE
      isRedundant = FALSE
      FOR ALL $R' \in S$ such that $R \neq R'$
        IF(*IsMadeRedundantBy*(R, R'))
          isRedundant = TRUE
    IF(isRedundant == FALSE)
      $S' = S' \cup \{R\}$
  RETURN S'

ComposeSetOfRules(S, S'):
  $S'' = S \cup S'$
  FOR ALL $R \in S'$
    $n = |\mathcal{P}(R)|$
    $T =$ *GetAllTuples*(S, n)
    FOR ALL $t \in T$
      $R' =$ *ComposeRule*(R, t)
      IF ($R' \neq$ NULL)
        $S'' = S'' \cup \{R'\}$
  RETURN S''

ComposeRule(R, t):
  $n = |\mathcal{P}(R)|$
  FOR $i = 1$ to n
    IF (*Unifiable*($\mathcal{P}_i(R)$, $\mathcal{C}(t_i)$))
      $\sigma_i =$ *FindMgu*($\mathcal{P}_i(R)$, $\mathcal{C}(t_i)$)
    ELSE
      RETURN NULL
  IF (*AreComposable*(σ_1, σ_2, ..., σ_n))
    $\sigma = \sigma_1 \circ \sigma_2 \circ \ldots \circ \sigma_n$
    $R' = \frac{\sigma(\mathcal{P}(t_1)) \quad \sigma(\mathcal{P}(t_2)) \quad \ldots \quad \sigma(\mathcal{P}(t_n))}{\sigma(\mathcal{C}(R))}$
    RETURN R'
  RETURN NULL

Table 3.2: The transformation algorithm: Part II

substitution[6] unifying m and m'. The utility function *AreComposable*(σ_1, σ_2) tests if σ_1 and σ_2 can be composed or if they are incompatible. For example, let $\sigma_1 = \{X \leftarrow f(b)\}$ and $\sigma_2 = \{X \leftarrow a\}$. Then σ_1 and σ_2 are incompatible (so they cannot be composed) because they disagree on the value that should be given to X and their disagreement cannot be overcome. Suppose instead that σ_2 would have been $\{X \leftarrow f(Y)\}$ and σ_1 remains unchanged, then here again σ_1 and σ_2 disagree on the value for X, but their disagreement can be overcome since $X\sigma_1$ and $X\sigma_2$ are unifiable by $\sigma' = \{Y \leftarrow b\}$. We provide in Section 4.3 a precise meaning for the composition[7] of substitutions ($\sigma = \sigma_1 \circ \sigma_2 \circ \ldots \circ \sigma_n$) as used in the function *ComposeRule*(R, t).

3.5.1 Optimization

We discuss some possible optimizations on the algorithms presented above. The first one should result in a good speed improvement while the second would probably prove itself useless because of the difficulty to implement it.

The first optimization takes place in the innermost loop of the function *ComposeSetOfRules*(S, S'), that is "FOR ALL $t \in T$". We often obtain the following situation that should be avoided for time consideration: we test for the composition of rule R with tuple t and obtain that they cannot be composed since the premise $\mathcal{P}_i(R)$ is not unifiable with the term t_i (the element at position i in the tuple t), after that, it is irrelevant to test if the rule R can be composed with a tuple t' where $t'_i = t_i$. Since there is m^{n-1} tuple t' (where m is the cardinality of S) having term t_i at position i ($t'_i = t_i$), this would yield in a good speed improvement. For the sake of algorithmic simplicity, we do not consider the optimized version.

The second optimization is found in the main function: *TransformInferenceSystem*(S) at the line: "$S' =$ *ComposeSetOfRules*($S_\prec$, $S_\succcurlyeq$)". Suppose $R_1 \in S_\prec$ and $R_2 \in S_\succcurlyeq$ at the first passage in the loop and that $R_3 = R_2 \Uparrow R_1$ exists. Now if R_1 and R_2 are still there at the second passage in the loop, the third passage, ..., it is useless to check for the same composition $R_2 \Uparrow R_1$ since it has already been computed. One may attempt to compose only new rules of $S_\succcurlyeq$ with any rule of $S_\prec$ and old rules of $S_\succcurlyeq$ with new rules in $S_\prec$, but it is much more complicated (new rules refer to the rules that were not present at the previous passage in the loop). A rule $R \in S_\succcurlyeq$ containing n premises may not be composable with a n-tuple of old $S_\prec$ rules, but it may be composable with a n_1-tuple of old $S_\prec$ rules and a n_2-tuple of new $S_\prec$ rules. This yields too

[6] Note that here we are working in $\emptyset$-unification, thus the *mgu*, if it exists, is unique (See Section 4.3).

[7] We chose to use composition of substitution here to replace the *cmgu* of *Mejri*.

many cases to allow a good algorithmic implementation. However, when applying the algorithm by hand, it may decrease the amount of work dramatically. In the example given in the next section, we apply the algorithm without this optimization. But in the rest of this thesis, when applying the algorithm, we shall consider this optimization to avoid useless computation.

As proposed in [10], when a rule is composed of multiple equal premises, we can keep only one of these equal premises and remove the others. We insist on the exactly equal since in the case that one premise has a link with the conclusion while the other has not, we cannot always safely remove one of them. This simplification will be useful for the proof of Lemma 4.4.1.

3.6 Example of inference system transformation

In this section, we show how to apply the transformation algorithm over the inference system S of equation 3.1. We will use the following ordering on terms: $t_1 \prec t_2$ iff t_1 contains strictly less function symbols than t_2.

$S^0_{\prec}$	$S^0_{\succcurlyeq}$
$R_2 = \frac{x_2}{f(f(f(x_2)))}$ $R_3 = \frac{\square}{a}$	$R_1 = \frac{f(f(x_1))}{f(x_1)}$

Table 3.3: $S^0_{\prec}$ and $S^0_{\succcurlyeq}$

First, we separate $S = S^0$ into $S^0_{\prec}$ and $S^0_{\succcurlyeq}$ as shown in Table 3.3. Next we compute $S' = S^0_{\prec} \looparrowright S^0_{\succcurlyeq} = S^0_{\prec} \cup S^0_{\succcurlyeq} \cup \{R_4 = R_1 \Uparrow R_2 = \frac{x_2}{f(f(x_2))}\}$. Then we eliminate the redundant rules of S' (computing $S'_{\Downarrow}$). Since no rule in S' is redundant, we rename the variables of the newly generated rule R_4 and $S'_{\Downarrow}$ becomes S^1.

$S^1_{\prec}$	$S^1_{\succcurlyeq}$
$R_2 = \frac{x_2}{f(f(f(x_2)))}$ $R_3 = \frac{\square}{a}$ $R_4 = R_1 \Uparrow R_2 = \frac{x_3}{f(f(x_3))}$	$R_1 = \frac{f(f(x_1))}{f(x_1)}$

Table 3.4: $S^1_{\prec}$ and $S^1_{\succcurlyeq}$

We start back, separating S^1 into $S^1_{\prec}$ and $S^1_{\succcurlyeq}$ as shown in Table 3.4. Next we compute

$S' = S^1_{\prec} \looparrowright S^1_{\succcurlyeq} = S^1_{\prec} \cup S^1_{\succcurlyeq} \cup \{R_5 = R_1 \Uparrow R_2 = \frac{x_2}{f(f(x_2))}\} \cup \{R_6 = R_1 \Uparrow R_4 = \frac{x_3}{f(x_3)}\}$. Then we eliminate the redundant rules of S' (computing $S'_{\Downarrow}$). The rule R_5 is made redundant by the rule R_4 (they are exactly the same rule modulo variable renaming). We rename the variables of the newly added rule R_6 and set S^2 to be $S'_{\Downarrow}$.

$S^2_{\prec}$	$S^2_{\succcurlyeq}$
$R_2 = \frac{x_2}{f(f(f(x_2)))}$ $R_3 = \frac{\square}{a}$ $R_4 = R_1 \Uparrow R_2 = \frac{x_3}{f(f(x_3))}$ $R_6 = R_1 \Uparrow R_4 = \frac{x_4}{f(x_4)}$	$R_1 = \frac{f(f(x_1))}{f(x_1)}$

Table 3.5: $S^2_{\prec}$ and $S^2_{\succcurlyeq}$

We start again, separating S^2 into $S^2_{\prec}$ and $S^2_{\succcurlyeq}$ as shown in Table 3.5. Next we compute $S' = S^2_{\prec} \looparrowright S^2_{\succcurlyeq} = S^2_{\prec} \cup S^2_{\succcurlyeq} \cup \{R_7 = R_1 \Uparrow R_2 = \frac{x_2}{f(f(x_2))}\} \cup \{R_8 = R_1 \Uparrow R_4 = \frac{x_3}{f(x_3)}\} \cup \{R_9 = R_1 \Uparrow R_6 = \frac{f(x_1)}{f(x_1)}\}$. Then we eliminate the redundant rules of S' (computing $S'_{\Downarrow}$). The rule R_7 is made redundant by R_4, the rule R_8 is made redundant by R_6 while the rule R_9 is self-redundant (since its premise is equal to its conclusion). We take S^3 to be $S'_{\Downarrow}$.

$S^3_{\prec}$	$S^3_{\succcurlyeq}$
$R_2 = \frac{x_2}{f(f(f(x_2)))}$ $R_3 = \frac{\square}{a}$ $R_4 = R_1 \Uparrow R_2 = \frac{x_3}{f(f(x_3))}$ $R_6 = R_1 \Uparrow R_4 = \frac{x_4}{f(x_4)}$	$R_1 = \frac{f(f(x_1))}{f(x_1)}$

Table 3.6: $S^3_{\prec}$ and $S^3_{\succcurlyeq}$

Since $S^2 = S^3$ (see tables 3.5 and 3.6), we have reached a fixed point in the series. We can now take $S^3_{\prec}$ to be an inference system equivalent to S such that all rules are oriented with respect to the ordering $\prec$ defined above. Note that the ordering $\prec$ is not a terminating ordering as defined in 3.3.3, so we cannot conclude that, in general, the resulting inference system will be a terminating inference system. However, it is easy to see that in this particular case, $S^3_{\prec}$ is terminating.

3.7 Other Methods

We give here a very quick overview of two other methods developed to solve the termination problem of inference systems.

3.7.1 *Blanchet*

In [9], *Blanchet* proposes a technique very similar to the one used by *Mejri* to solve the termination problem of inference systems; again in the context of cryptographic protocols. The main difference between the two approaches is the extraction of an inference system from a protocol. Where *Blanchet* introduces an approximation in the model, *Mejri* is a lot more precise. In fact, *Blanchet* proposes a model in which a step of the protocol can be completed several times as long as the previous steps have been completed at least once between the same principals. This approximation introduces the possibility to find false attacks against a protocol. As a matter of fact, if a flaw is found within the framework of *Blanchet*, it is not necessarily the case that the protocol under consideration is flawed (the trace representing the flaw can be a non-valid trace in the sense of Definition 1.4.3).

Another difference between the two approaches is in the way rules are selected for composition. *Mejri* simply partitions the entire set of rules in terminating and non-terminating ones, always composing a non-terminating rule with terminating ones. On the other hand, *Blanchet* uses an heuristic to select rules to be composed with the objective to reduce, as much as possible, the risk of infinite looping during composition.

Blanchet also proposes to simplify some rules in the following way: if a rule R contains the premise $p = X$ such that X does not appear in the conclusion, the premise p can safely be removed from R. The idea is that the intruder can clearly provide a message to replace X, and since X has no influence on the rule R, the message provided is unimportant. This idea of *Blanchet* has been important for us in at least two points:

- by removing such a premise, we were able to simplify some proofs (see the proof of Lemma 5.3.2 for example);
- it helped understand the existential quantification of free variables (this point is discussed at the end of Section 4.5.1)

In [10], *Blanchet* & *Podelski* propose a class of protocols, namely tagged proto-

cols, on which the transformation algorithm of *Blanchet* terminates. A protocol is tagged when all different messages are tagged (with a constant message) before being encrypted. This tagging prevents the intruder to use a message from a step α in a different step β. It is shown, in [35], that tagged protocols are not subject to type flaws. Thus tagging is now seen as a good practice in protocol construction. Unfortunately, among the already existing protocols, not all are tagged (some may be implicitly tagged as mentioned in [10]). *Blanchet* & *Podelski* also propose a transformation from a protocol to a tagged version of this protocol. The tagged version of the protocol can then be analyzed using the method proposed by *Blanchet*. Unfortunately, since the tagged version of a protocol is possibly more secure then its original version, the tagged protocol can be declared secure while the initial version is flawed. Moreover, since the verification can introduce a false attack (as discussed above), starting from a non-tagged protocol and analyzing its tagged version does not, in general, provide very good information about its security.

3.7.2 *Kindreed & Wing*

In [45], see also [42, 43, 44], *Kindreed & Wing* propose an algorithm to generate a finite and decidable representation of the whole theory of an inference system. They do not, like *Mejri* and *Blanchet*, propose an idealization process turning a cryptographic protocol into an inference system. Instead, they concentrate on using existing logics to model protocols and they provide a mean to decide these logics (represented as inference systems).

Rules of the inference systems are separated into S-rules and G-rules. A S-rule R is such that at least one primary[8] premise of R is not smaller then the conclusion (S-rules could be seen as our set $S_{\succcurlyeq}$). A G-rule R is such that each premise of R is smaller than the conclusion (G-rules could be seen as our set $S_{\prec}$).

The theory is built using a S-rule in a forward chaining way (proving primary premises with axioms) and then using G-rules in a backward chaining way to prove the non-primary premises of the S-rule. The theory is completely generated when no new formula can be generated.

[8] A premise is primary if it cannot be unified with any G-rule conclusion.

3.8 Conclusion

In this chapter we presented a transformation process, from a non-terminating inference system into a terminating one, due to *Mejri* in [57]. This transformation was developed to address the problem of non-termination of the inference system extracted from a protocol in an attempt deciding to decide the correctness of the protocol, with respect to some restricted security properties. However, the transformation process will not always terminate, thus we introduced a new termination problem by solving another one. In the next chapters, we address this new termination problem and show that for a wide class or cryptographic protocols, the transformation process always terminates.

Chapter 4

On the Convergence of the Transformation Algorithm

Abstract

In the previous chapter, we explained how an inference system can be transformed into another equivalent, but terminating one. We mentioned that this transformation process also suffers from the termination problem. We address here the termination problem of the transformation process.

4.1 Introduction

In this chapter, we are interested in the termination of the transformation algorithm of *Mejri* (see [57]); the one presented in Chapter 3. To avoid confusion, we adopt the following conventions:

- We will use the expression *termination* in the context of an inference system. An inference system is terminating if there is a proof-search procedure for the theory of this inference system.
- By *transformation algorithm* (or *transformation process*), we mean the algorithm of Section 3.5 transforming a non-terminating inference system into an equivalent (possibly) terminating inference system. When using the transformation process, we say *initial inference system* for the inference system on which the transforma-

tion process is first applied and *resulting inference system* for the inference system returned by the transformation algorithm.

- We will talk about *convergence* (resp. *non-convergence*) to denote the termination (resp. non-termination) of the transformation algorithm. This is inspired by the equivalence between the termination of the transformation process and the convergence of the associated series of inference systems.

The present chapter contains the main result of this thesis: we define a class of inference systems, namely structured inference systems, and a partitioning of the rules such that the transformation algorithm is shown to converge. We then prove that the resulting inference system is terminating and we present a decision procedure for its theory. To do this, we first need to define a message algebra, that is the message on which we build the inference systems; this will be done in Section 4.2. Next, we will discuss, in Section 4.3, the unification of terms and the composition of substitutions. These points have not been formally addressed in the previous chapter, so we give a more formal approach here. Section 4.4 contains the convergence proof of the transformation process when applied to a structured inference system. Finally, since it is not trivial that the resulting inference systems are terminating, we propose in Section 4.5 a decision procedure for resulting inference systems together with a termination proof for this procedure.

4.2 Message algebra

Before working with inference systems, we need to know which terms can be used to build them. To define the terms that are valid (also called well-formed terms), we propose an algebra, namely the simple-linear-sorted algebra. We need the following definitions to introduce our simple-linear-sorted algebra (or simple-linear for short).

Definition 4.2.1 (Signature)
An order-sorted signature is a pair $((S, \leq), \Sigma)$ *where* $(S, \leq)$ *is a partially ordered set of sorts (the types' names) and* Σ *is a set of function symbols. Each function symbol is associated with an arity i.e. the number of arguments required by this function. For example, function* $f : s_1 \times \ldots \times s_n \rightarrow s_{n+1}$*, where the* s_i *range over the domain* S*, has arity* n*;* $s_1 \times \ldots \times s_n$ *is called the domain of* f *while* s_{n+1} *is its codomain. Functions of arity 0 are called constants.*

The sorts (or types as they are called from now on) can be divided into two: atomic and non-atomic types. A type s is atomic if every term of type s has size 1[1], otherwise it is non-atomic. The size will be defined more formally below. We assume from now on that $(S, \leq)$ forms a finite forest, that is: for any $s, s_1, s_2 \in S$, $s_1 \geq s \leq s_2$ implies $s_1 \geq s_2$ or $s_1 \leq s_2$. The choice to use a forest structure for the sorts will become clear in the next section.

Definition 4.2.2 (Σ-algebra)
Let $((S, \leq), \Sigma)$ be an order-sorted signature. An order-sorted Σ-algebra is a pair (A, F) where A is an S-indexed family of sets, say $A = \cup_{s \in S} A_s$, and F is a Σ-indexed set of functions $\{f_A | f \in \Sigma\}$, such that for all sorts s and s', $s \leq s'$ implies $A_s \subseteq A_{s'}$.

Definition 4.2.3 (Free algebra)
Let $((S, \leq), \Sigma)$ be an order-sorted signature and X be an S-indexed set of variable identifiers i.e. $X = \cup_{s \in S} X_s$ such that $X \cap \Sigma = \emptyset$. The free Σ-algebra over X is the pair $(T_\Sigma(X), F_\Sigma)$ where:

- *The S-indexed family $T_\Sigma(X)$ is defined as the least set satisfying:*
  - *For all $x \in X_s$, $x \in (T_\Sigma(X))_s$;*
  - *For all $cte :\rightarrow s \in \Sigma$, $cte \in (T_\Sigma(X))_s$;*
  - *For all $f : s_1 \times \ldots \times s_n \rightarrow s \in \Sigma$ and for all $(t_1, \ldots, t_n) \in (T_\Sigma(X))_{s_1} \times \ldots \times (T_\Sigma(X))_{s_n}$, $f(t_1, \ldots, t_n) \in (T_\Sigma(X))_s$.*
- *The Σ-indexed set of functions F_Σ is made of functions $f_{T_\Sigma(X)} : T_\Sigma(X))_{s_1} \times \ldots \times (T_\Sigma(X))_{s_n}$ where $f(t_1, \ldots t_n) \in (T_\Sigma(X))_s$. It takes the tuple $(t_1, \ldots, t_n)$ to the term $f(t_1, \ldots, t_n)$.*

To clarify the concept of types and order between types, we give an example of free algebra. This free algebra is the one used to describe messages in cryptographic protocols.

Example 4.2.1 (Cryptoprotocols' free algebra)
We consider the types *Principals*, *Keys*, *Nonces*, *Int* and *Msg* partially ordered by: type *Msg* is bigger then every other type. Only the types *Principals* and *Int* are atomic, the others are non-atomic. The functions are:

- $E : Msg \times Keys \rightarrow Msg$;

[1] We could probably define atomic type to be bounded by a constant, instead of bounded by 1. This would require a modification on our proofs.

- $C : Msg \times Msg \rightarrow Msg$.
- $K : Principals \times Principals \rightarrow Keys$.
- $N : Principals \times Int \rightarrow Nonces$.

Some free terms of the mentioned algebra would be:

- $t_1 : C(p_1, E(X_1, K(p_1, p_2))$
- $t_2 : C(E(X_2, K(p_2, p_3)), E(X_2, K(p_3, P_4)))$
- $t_3 : C(p_5, E(E(C(X_3, p_5), K(p_5, p_6)), K(p_6, p_7)))$
- $t_4 : C(E(X_4, K(p_8, p_9)), E(X'_4, K(p_8, p_9)))$

where we use p_i and X_i for variables of type *Principals* and *MSG* respectively.

Variables will play a very important role in this work. Among other things, we will distinguish between atomic variables and non-atomic variables. A variable $x \in X_s$ is atomic if s is an atomic type, otherwise it is non-atomic. We now give functions to compute the set of atomic variables and the set of non-atomic variables from a given term.

Definition 4.2.4 ($AVar(\_)$)
Given a term t, we use the function $AVar(t)$ to get the set of atomic variables in t. $AVar(t)$ is defined to be:

- $\emptyset$ *in the case* $t = cte :\rightarrow s$;
- $\{t\}$ *in the case* $t \in X_s$ *such that* s *is an atomic type;*
- $AVar(t_1) \cup \ldots \cup AVar(t_n)$ *in the case* $t = f(t_1, \ldots, t_n)$.

Definition 4.2.5 ($NAVar(\_)$)
Given a term t, we use the function $NAVar(t)$ to get the set of non-atomic variables in t. $NAVar(t)$ is defined to be:

- $\emptyset$ *in the case* $t = cte :\rightarrow s$;
- $\{t\}$ *in the case* $t \in X_s$ *such that* s *is a non-atomic type;*

- $NAVar(t_1) \cup \ldots \cup NAVar(t_n)$ *in the case* $t = f(t_1, \ldots, t_n)$.

From the two definitions above, we can easily compute the set of variables in a term t by $AVar(t) \cup NAVar(t)$.

Since we distinguish terms containing non-atomic variables from the other terms, we propose the following definitions. Intuitively, a term m is invariant if its size is preserved by substitution, that is if m contains only atomic variables.

Definition 4.2.6 (Variant and invariant term)
A term m is variant if $NAVar(m) \neq \emptyset$. Otherwise, it is invariant.

We are now ready to define the simple-linear algebra. A term t is simple-linear if it contains at most one non-atomic variable, say X, and X occurs at most once in t.

Definition 4.2.7 (Simple-linear algebra)
Let $((S, \leq), \Sigma)$ be an order-sorted signature and X be an S-indexed set of variable identifiers i.e. $X = \cup_{s \in S} X_s$ such that $X \cap \Sigma = \emptyset$. The simple-linear Σ-algebra over X is the pair $(T_\Sigma(X), F_\Sigma)$ where:

- *The S-indexed family $T_\Sigma(X)$ is defined as the least set satisfying:*
  - *For all $x \in X_s$, $x \in (T_\Sigma(X))_s$;*
  - *For all $cte :\rightarrow s \in \Sigma$, $cte \in (T_\Sigma(X))_s$;*
  - *For all $f : s_1 \times \ldots \times s_n \rightarrow s \in \Sigma$ and for all $(t_1, \ldots, t_n) \in (T_\Sigma(X))_{s_1} \times \ldots \times (T_\Sigma(X))_{s_n}$, if:*
    - *$NAVar(t_i) \cap NAVar(t_j) = \emptyset$ for every $i \neq j$ and*
    - *$|NAVar(f(t_1, \ldots, t_n))| \leq 1$, then $f(t_1, \ldots, t_n) \in (T_\Sigma(X))_s$.*
- *The Σ-indexed set of function F_Σ is made of functions $f_{T_\Sigma(X)} : T_\Sigma(X))_{s_1} \times \ldots \times (T_\Sigma(X))_{s_n}$ where $f(t_1, \ldots t_n) \in (T_\Sigma(X))_s$ if $NAVar(t_i) \cap NAVar(t_j) = \emptyset$ for every $i \neq j$ and $|NAVar(f(t_1, \ldots, t_n))| \leq 1$. It takes the tuple $(t_1, \ldots, t_n)$ to the term $f(t_1, \ldots, t_n)$.*

We choose to use the simple-linear algebra for the following reasons. The main objective was to restrict the algebra in such a way that would allow us to control the size growth during unification (this follows from the intuition behind Lemma 3.4.2). The restriction to simple terms allows a simple proof to bound the size of the term t resulting from unifying t_1 with t_2 (as used in the proofs of propositions 4.3.5 and 4.3.6).

Linearity over non-atomic variables is very useful to prove that unification behave nicely (see for example the proof of Proposition 4.3.3). Note that we are not sure yet if these conditions are necessary to assure the convergence of our transformation process. We could probably relax these conditions, but this may require a great deal of effort to adapt the proofs.

The simple-linear algebra defined above will be the one considered for the terms used to build rules in our inference systems. Note that the restrictions imposed by the simple-linear algebra are only on non-atomic variables. We will say a simple-linear term to denote a term built in the simple-linear algebra and we use $\mathcal{T}$ to denote the set of simple-linear terms. From the free terms presented in Example 4.2.1, only terms t_1 and t_3 are also part of the simple-linear algebra. t_2 is not a simple-linear term since the non-atomic variable X_2 appears twice in the term, while t_4 is not simple-linear because it is composed of two non-atomic variables.

From now on, we shall consider the set of function symbols to be finite. We now define some functions to allow the comparison of different terms.

Definition 4.2.8 (Size function)
Let t be a simple-linear term. The size of t is the number of atomic messages (variables and constants) used to build it. We use $Size(t)$ (or $|t|$ for short) to denote the size of t. $Size(t)$ is:

- *1 if $t \in X$;*
- *1 if $t :\rightarrow s \in \Sigma$;*
- *$Size(t_1) + \ldots + Size(t_n)$ if $t = f(t_1, \ldots, t_n)$.*

Definition 4.2.9 (Subterm)
Let t be a simple-linear term. We say t' is a subterm of t, noted $t' \sqsubseteq t$ if:

- *$t = t'$ or*
- *$t = f(t_1, \ldots, t_n)$ and $\exists 1 \leq i \leq n$ such that $t' \sqsubseteq t_i$.*

When t is a simple-linear term, we say t' is a strict subterm of t, noted $t' \sqsubset t$, if $t' \sqsubseteq t$ and $t' \neq t$.

Definition 4.2.10 (Term equality)
We say terms t_1 and t_2 are equal, denoted $t_1 = t_2$, if they are syntactically equal. We use $t_1 =_R t_2$ to say t_1 equals t_2 modulo renaming, that is $t_1\sigma = t_2$ where σ is a renaming of variables.

To define the rules of our inference systems, we will need the notion of **term structure**. A term structure is a high level representation of a class of terms such that variables and constants are replaced by their corresponding type identifiers. Thus a term structure is common to a class of term, but a given term has only one associated term structure.

Example 4.2.2
Given the following term structure $f(\tau_1, h(\tau_2, \tau_3))$ where τ_i are types, the following messages are derived from the above term structure:

- $t = f(a, h(b, c))$
- $t' = f(x_1, h(b, c))$
- $t'' = f(x_1, h(x_2, x_3))$

where a, b, c are constants of type τ_1, τ_2, τ_3 respectively, and x_1, x_2, x_3 are variables of type τ_1, τ_2, τ_3 respectively.

We now define the set of size bounded term structure. The size of a term structure is the number of type identifier (not necessarily distinct) used to build it. The term structure of Example 4.2.2 has size 3. Note that linearity and simplicity does not apply to term structures.

Definition 4.2.11 ($\mathcal{TS}^{(v,t)}$)
Let s be a term structure, $s \in \mathcal{TS}^{(v,t)}$ if:

- $|s| \leq v$ *and s contains at least one non-atomic type, or*
- $|s| \leq t$ *and s contains only atomic types.*

Lemma 4.2.1
$\mathcal{TS}^{(v,t)}$ *is finite.*

Proof:
It follows directly from the fact that there is a finite number of function symbols and a finite number of type.

□

Any term is derived from a term structure. A term t is said to derive from term structure s if t can be obtained from s by replacing every type identifier of s either by a variable or a constant (the use of complex term such as $F(x_1, x_2)$ is not allowed in the derivation process) of the corresponding type. We now define the set of size-bounded simple-linear terms.

Definition 4.2.12 ($\mathcal{T}^{(v,t)}$)
$t \in \mathcal{T}^{(v,t)}$ if t is simple-linear and if there exists a term structure $s \in \mathcal{TS}^{(v,t)}$.

It is clear that $\mathcal{T}^{(v,t)}$ has infinitely many elements since we are not limited to a finite number of variables. However, it is the case that there is a finite number of size bounded simple-linear term different modulo renaming (this is a consequence of the finiteness of $\mathcal{TS}^{(v,t)}$ together with the fact that there is a finite number of constants).

4.3 Unification

In this section we will prove some general properties of simple-linear terms unification under the empty equational theory[2]. We also provide a way to compute the most general unifier of two terms and, more importantly, a way to compose substitutions. For a general treatment or an introduction to unification, the reader is referred to [6, 77, 80].

It is well-known (see [51, 78]) that $\emptyset$-unification is decidable and unitary for an untyped algebra. It is possible to know whether two terms t and t' are unifiable ($t\sigma = t'\sigma$ for some substitution σ). In the case two terms are unifiable, it is possible to compute (using the algorithm of [51]) the most general unifier of t and t' ($mgu(t, t')$) and this *mgu* is always unique. However, the unitary property does not necessarily hold for a typed algebra. Example 4.3.1 provide a good proof that $\emptyset$-unification is not necessarily unitary when considering a typed algebra.

Example 4.3.1
Suppose we have the following types τ_1, τ_2, τ_3 and τ_4 partially ordered by $\tau_1 < \tau_3, \tau_1 < \tau_4, \tau_2 < \tau_3$ and $\tau_2 < \tau_4$. The terms $t_1 = X : \tau_3$ and $t_2 = Y : \tau_4$ are $\emptyset$-unifiable and they possess the two, incomparable, *mgu*:

- $\sigma_1 = \{X \leftarrow \alpha : \tau_1, Y \leftarrow \alpha : \tau_1\}$

[2]The functions are neither associative nor commutative, ...

- $\sigma_2 = \{X \leftarrow \beta : \tau_2, Y \leftarrow \beta : \tau_2\}$

However, we know from [84] that having a typed algebra with sorts forming a forest structure is a sufficient condition to get an unitary $\emptyset$-unification. That is why we made the assumption that our sorts form a forest structure. The less restrictive condition of having a finite set of types should pose no problem (since it is known to be finitary, see again [84]) but would require a more complex treatment.

Using the result of [84] when sorts form a forest, we can adapt the unification algorithm of [51] to compute the *mgu* of two simple-linear terms. Computing the *mgu* of a pair of terms $S = \{(t, t')\}$ is done by repeatedly (and non-deterministically) performing any of the following transformation. Select a pair of terms $(t_1 : \tau_1, t_2 : \tau_2)$ from S, if:

- (A) both t_1 and t_2 are variables and $t_1 = t_2$, then remove (t_1, t_2) from S.
- (B) both t_1 and t_2 are variables and $\tau_1 < \tau_2$, then remove (t_1, t_2) from S and add (t_2, t_1) to S.
- (C) both t_1 and t_2 are variables and $\tau_1 \bowtie$ [3] τ_2, then stop with failure.
- (D) t_2 is a variable and t_1 is not; if $\tau_1 \leq \tau_2$ then remove (t_1, t_2) and insert the pair (t_2, t_1) in S, otherwise stop with failure.
- (E) t_1 and t_2 are not variables. If the two root function symbol are different, then stop with failure, otherwise remove (t_1, t_2) and add the pairs $(t_{11}, t_{21}), \ldots, (t_{1n}, t_{2n})$ to S, where $t_1 = f(t_{11}, \ldots, t_{1n})$ and $t_2 = f(t_{21}, \ldots, t_{2n})$.
- (F) t_1 is a variable which occurs somewhere else in S and $t_2 \neq t_1$. If $t_1 \sqsubset t_2$ or $\tau_1 < \tau_2$, then stop with failure, otherwise replace every occurrence of t_1 by t_2 in S, except the one in the pair (t_1, t_2). Note that (t_1, t_2) remains in S.

When no transformation applies, stop with S containing the *mgu* of t and t'. We first check if for every pair $(t_1 : \tau, t_2 : \tau')$ of S we have $\tau' \leq \tau$, if it is not the case, the initial terms were not unifiable. If all types checks are ok, we form the *mgu* σ from S by taking every pair (t_1, t_2) of S and adding the affectation $t_1 \leftarrow t_2$ to σ.

When we talk about the unification of two terms t_1 and t_2, we assume $Var(t_1) \cap Var(t_2) = \emptyset$ (if it is not the case, we can rename the variables shared between the

[3] $\tau_1 \bowtie \tau_2$ means τ_1 and τ_2 are incomparable

two terms before the unification). We are now ready to give some properties respected in the unification of two simple-linear terms. These properties will be very helpful in proving the convergence of the transformation algorithm.

The first property states that substituting a variable in a simple-linear term by another simple-linear term yields a simple-linear term.

Proposition 4.3.1
Let $t \in \mathcal{T}$ and $\sigma = \{X \leftarrow t'\}$ be such that $t' \in \mathcal{T}$, then $t\sigma \in \mathcal{T}$.

Proof:
If $X \not\sqsubseteq t$, then $t\sigma = t$ and it is simple-linear. Otherwise, $NAVar(t) = \{X\}$, since t is simple and X occur exactly once in t, because t is linear. Since t' is simple and $NAVar(t) = \{X\}$, $NAVar(t\sigma) = NAVar(t')$, thus $t\sigma$ is simple. Since t and t' are linear, so will be $t\sigma$.

□

The next property says that the *mgu* of two simple-linear terms contains only simple-linear affectation. An affectation $X \leftarrow m$ is simple-linear if m is simple-linear.

Proposition 4.3.2
Let $t_1, t_2 \in \mathcal{T}$ and σ be their mgu. If $X \leftarrow m \in \sigma$, then $m \in \mathcal{T}$.

Proof:
We suppose that every pair in S is simple-linear[4] (as it is the case when the algorithm is first called), and we show the application of any transformation rule preserves this situation.

- If we apply rule A: this rule simply removes a pair from S.
- If we apply rule B: since the pair (t_1, t_2) is simple-linear, the pair (t_2, t_1) will also be.
- If we apply rule C: this will end up with a failure.
- If we apply rule D: this will either add the pair (t_2, t_1), which is simple-linear since (t_1, t_2) is or it will end up with a failure.

[4] A pair of terms is simple-linear if both terms of this pair are simple-linear.

- If we apply rule E: suppose $t_1 = f(t_{11}, \ldots, t_{1n})$ and $t_2 = f(t_{21}, \ldots, t_{2n})$. Since t_1 is simple-linear, every t_{1i} must also be simple-linear. In the same way, every t_{2i} is simple-linear. Thus every pair (t_{1i}, t_{2i}) is simple-linear.
- If we apply rule F: Let $t_1 = X$, $\sigma = \{X \leftarrow t_2\}$ and t be any term in S. Since t_2 and t are simple-linear, then $t\sigma$ will also be simple-linear(see Proposition 4.3.1).

□

We now give a bound on the size of the affectations in the *mgu* of two simple-linear terms. By the size of the affectation $X \leftarrow m$, we mean $|m|$.

Proposition 4.3.3
Let $t_1, t_2 \in \mathcal{T}$ be such that $X \in NAVar(t_1)$ and $X \notin NAVar(t_2)$ and let $\sigma = mgu(t_1, t_2)$. If $X \leftarrow m \in \sigma$, then $|m| \leq |t_2|$.

Proof:
Suppose we apply the unification algorithm with the initial set $S = \{(t_1, t_2)\}$ such that $X \in NAVar(t_1)$ and $X \notin NAVar(t_2)$. The algorithm will eventually end up with a pair (X, m') such that $m' \sqsubseteq t_2$. Clearly here, $|m'| \leq |t_2|$. It is clear that only the transformation rule F of the algorithm can increase the size of m'. If m' contains no non-atomic variables, then clearly $|m| = |m'| \leq t_2$. Now suppose $NAVar(m') = \{Y\}$. For the transformation rule to increase the size of m', another pair (t'_1, t'_2) must exist in S such that $t'_1 = Y$. Clearly, Y cannot come from t_1 since $NAVar(t_1) = \{X\}$. It is also clear that the single occurrence of Y in t_2 (remember that t_2 is linear) cannot be present in two pairs (it is already in m' of the pair (X, m')). Thus no transformation rule can increase the size of a term in a pair.

□

When unifying two terms t_1 and t_2 with σ, the result is $t = t_1\sigma = t_2\sigma$. The next three properties concern this t. We first prove that t is simple-linear, then we bound the size of t. These properties are the key of our convergence proof.

Proposition 4.3.4
Let $\sigma = mgu(t_1, t_2)$ where $t_1, t_2 \in \mathcal{T}$ and let $t = t_1\sigma = t_2\sigma$, then $t \in \mathcal{T}$.

Proof:
Since t_1 and t_2 are simple-linear terms, we know by Proposition 4.3.2 that every affec-

tation in σ has the form $X \leftarrow m$ where m is simple-linear. Proposition 4.3.1 implies that $t = t_1\sigma = t_2\sigma$ is simple-linear.

□

Proposition 4.3.5
Let $\sigma = mgu(t_1, t_2)$ where $t_1, t_2 \in \mathcal{T}$ and let $t = t_1\sigma = t_2\sigma$. If $NAVar(t) \neq \emptyset$ then $|t| \leq max(|t_1|, |t_2|)$.

Proof:
It must be the case that $NAVar(t_1) = \{X\}$ and $NAVar(t_2) = \{Y\}$ to have $NAVar(t) \neq \emptyset$. σ must contain either $X \leftarrow m$ or $Y \leftarrow m'$ but not both. Suppose σ contains $X \leftarrow m$. We know $|t| = |t_2\sigma| = |t_2|$ and since $t_1 \leq |t_1\sigma| = |t|$, we conclude that $|t_2| \leq max(|t_1|, |t_2|)$.

□

Proposition 4.3.6
Let $\sigma = mgu(t_1, t_2)$ where $t_1, t_2 \in \mathcal{T}$ and let $t = t_1\sigma = t_2\sigma$. Suppose $NAVar(t) = \emptyset$, then:

- *1: if $NAVar(t_2) = \emptyset$, we have $|t| = |t_2|$;*
- *2: if both $NAVar(t_1)$ and $NAVar(t_2)$ are non-empty, we have $|t| < |t_1| + |t_2|$.*

Proof:

- 1: Since $NAVar(t_2) = \emptyset$, we have $|t| = |t_2\sigma| = |t_2|$.
- 2: Suppose $NAVar(t_1) = \{X\}$ and $NAVar(t_2) = \{Y\}$. Since $NAVar(t) = \emptyset$, σ must contain both affectations $X \leftarrow m$ and $Y \leftarrow m'$. We know $|t| = |t_1\sigma| = |t_1| - 1 + |m| < |t_1| + |m| \leq |t_1| + |t_2|$ by Proposition 4.3.3 and by the form of σ.

□

In the process of rules composition, we compute independently the *mgu* of some pairs of terms and then compute the final *mgu* by composing the intermediate *mgu*s (see the algorithm of Table 3.2). We now explain how the composition of two substitutions can

be done and also some properties about the substitution resulting of this composition. We assume the unification is done over some *NA*-disjoint[5] pairs of simple-linear terms $(t_1, t'_1) \dots (t_n, t'_n)$.

Definition 4.3.1 (Substitution conflict)
Given two substitutions σ_1 and σ_2. There is a conflict between σ_1 and σ_2 when $X \leftarrow m_1 \in \sigma_1$ and $X \leftarrow m_2 \in \sigma_2$ where $m_1 \neq m_2$. The conflict can be resolved iff $m_1\sigma = m_2$ for some substitution σ.

It is clear that when a conflict arises for the variable X, then X is an atomic variable (since a non-atomic variable is present in at most one of the two terms). Thus solving the conflicts will not change the size of any message in the substitutions. We now give the conditions under which two substitutions are composable.

Definition 4.3.2 (Composability of substitutions)
Given two disjoint pairs of simple-linear terms (t_1, t'_1) and (t_2, t'_2) such that $\sigma_i = mgu(t_i, t'_i)$. The two substitutions σ_1, σ_2 are composable into $\sigma = \sigma_1 \circ \sigma_2$ iff every conflict between σ_1 and σ_2 can be resolved.

The next two properties about the composition of substitution are very important. We let $\sigma_i = mgu(t_i, t'_i)$ for some disjoint pairs of simple-linear terms and $\sigma = \sigma_1 \circ \dots \circ \sigma_n$. Then:

Proposition 4.3.7
If $X \leftarrow m \in \sigma$, then $m \in \mathcal{T}$.

Proof:
It is the application of Proposition 4.3.2 with the observation that the composition simply resolves conflict between atomic variables.

□

Proposition 4.3.8
$|t_1\sigma_1| = |t'_1\sigma_1| = |t_1\sigma| = |t'_1\sigma|$.

Proof:
It follows directly from the observation that the size of the affectations in a substitution remains unchanged through composition, since composition simply resolves conflicts between atomic variables.

[5] Two different terms have no common non-atomic variable

□

The restrictions imposed on terms to be unified do not always hold, but they do hold for the transformation process over inference systems (some of them will hold only for the transformation process over structured inference systems).

4.4 The transformation algorithm converges

In this section, we define a class of inference systems, namely structured inference systems (see Definition 4.4.4), for which the transformation process always converges when partitioning the rules in a particular manner (as explained in Definition 4.4.5). As a direct consequence, we will have a decision procedure for every protocol whose inference system is structured. This last point will be the subject of Chapter 5. To achieve this convergence result, we will first model the transformation process by a function Φ, taking an inference system and computing the next inference system of the associated series (see def 3.4.5). Once this function is defined and some properties are established, we prove that the recursive application of this function on any structured inference system reaches a fixpoint, thus proving that the series converges. Of course, we start by defining the class of structured inference systems.

4.4.1 Structured inference systems

We say that a rule is simple-linear if all the terms forming this rule (the premises and conclusion) are simple-linear. We talk about a simple-linear inference system to denote an inference system containing only simple-linear rules. Two rules R and R' are equals modulo renaming, denoted $R =_R R'$, if $\mathcal{C}(R) =_R \mathcal{C}(R')$ and for all $p \in \mathcal{P}(R)$, there exists a $p' \in \mathcal{P}(R')$ such that $p =_R p'$ and vice versa.

In Section 3.4, we defined (see Definition 3.4.4) the simplified inference system associated with S by $S_{\Downarrow}$. In this simplified inference system, the redundant rules were eliminated. We give here a new definition of simplified inference system (we will denote it by $S_{\downarrow}$ to avoid confusion). As a consequence of this new definition, the convergence proof will be much more elegant and simple.

Definition 4.4.1 (Simplified inference system)
Let S be an inference system, we define $S_{\downarrow}$ as follows. Let $R \in S$, then $R \in S_{\downarrow}$ if

- *A:* $\forall p \in \mathcal{P}(R), p \neq \mathcal{C}(R)$ *and*
- *B:* $\forall R \in S_{\downarrow}, R \neq_R R'$.

It is easy to see that this new definition of simplified inference system does not change the results stated in the previous chapter. As a matter of fact, the previous definition of simplified inference systems (see Definition 3.4.4) corresponds to an optimization in the transformation process when compared to the present definition. Here, we allow some useless work to be done. Thus, if the transformation process converges with the present simplification of inference system, it surely converges with a more optimized simplification. Note that from now on, when computing $(S_1 \looparrowright S_2)_{\downarrow}$, we are careful not to remove any rule of $S_1 \cup S_2$. As a result, we always have $(S_1 \cup S_2) \subseteq (S_1 \looparrowright S_2)_{\downarrow}$.

The following two definitions introduce two conditions over rules, conditions that will be used to define the class of structured inference systems. The condition "*self-contained*" states that if a non-atomic variable X is found in the conclusion of a rule, then X must also be in at least one premise of this rule. Condition "*premises-disjoint*" imposes the different premises of a rule to be pairwise disjoint over their set of non-atomic variables.

Definition 4.4.2 (Self-contained rule)
A rule R is self-contained iff $NAVar(\mathcal{C}(R)) \subseteq \bigcup_{p \in \mathcal{P}(R)} NAVar(p)$.

Definition 4.4.3 (Premises-disjoint rule)
A rule R is premises-disjoint iff $\forall i \neq j,\ NAVar(\mathcal{P}_i(R)) \cap NAVar(\mathcal{P}_j(R)) = \emptyset$

An inference system S is said to be self-contained (resp. premises-disjoint) iff every rule of S is self-contained (resp. premises-disjoint). We use SC^{IS} (resp. PD^{IS}) to denote the set of all self-contained (resp. premises-disjoint) inference system.

With the two conditions defined just above, and using the simple-linear algebra defined in the previous section, we can now introduce the class of structured inference systems. We use *SIS* to denote the set of structured inference systems.

Definition 4.4.4 (Structured inference system)
A simple-linear simplified inference system S is structured if $S \in SC^{IS} \cap PD^{IS}$.

Given a structured inference system S, we need to separate it into two sets of rules: one with terminating rules and the other with non-terminating rules. We will write $T(S)$ for the set containing all the terminating rules of S and $NT(S)$ for the set containing all the non-terminating rules of S. There is possibly many ways to partition S into $T(S)$

and $NT(S)$ depending of the algorithm used to search for a proof (backward-chaining, forward-chaining, ...). Note that if the transformation process converges with a given partition, it will not necessarily converge for another partition. Definition 4.4.5 states how this separation is done in our work and Section 4.5 exhibits a terminating algorithm to decide the theory of the resulting inference systems.

Definition 4.4.5 (Terminating rules)
Given $S \in SIS$, the set of terminating rules from S (noted $T(S)$) is such that $R \in T(S)$ iff:

- $NAVar(\mathcal{C}(R)) = \emptyset$ *or*
- $\forall i$ *such that* $NAVar(\mathcal{P}_i(R)) = NAVar(\mathcal{C}(R)) \neq \emptyset$ *we have that* $|\mathcal{P}_i(R)| < |\mathcal{C}(\mathcal{R})|$

We use $NT(S)$ to denote the set of non-terminating rules from S. From the above definition, $NT(S) = S - T(S)$.

Let $S \in SIS$, we define some size measures over S:

- V_S: The size of the biggest variant term in S.
- C_S: The size of the biggest invariant term in S.
- T_S: $max(C_S, 2 \times V_S)$.

The bound $2 \times V_S$ has been chosen to reflect the unification property of Proposition 4.3.6-2.

In Section 4.2, we defined the set of size-bounded term structure. We now introduce the set of rules constructed over term derived from size-bounded term structure.

Definition 4.4.6 ($\mathcal{R}^{(v,t)}$)
Given a premises-disjoint and self-contained rule R, $R \in \mathcal{R}^{(v,t)}$ if:

- $\mathcal{C}(R) \in \mathcal{T}^{(v,t)}$ *and*
- $\mathcal{P}(R) \subseteq \mathcal{T}^{(v,t)}$ *and*
- $\nexists R' \in \mathcal{R}^{(v,t)}$ *such that* $R' =_R R$ *and*
- $\exists \alpha$ *such that* $|\mathcal{P}(R)| \leq \alpha$.

Lemma 4.4.1
$\mathcal{R}^{(v,t)}$ *is finite.*

Proof:
Since every rule has a bounded number of premise, since there is a finite number of size-bounded simple-linear terms different modulo renaming, infinitely many rules would yield infinitely many rules equal modulo renaming, and thus they would be eliminated.

□

We need $\mathcal{R}^{(v,t)}$ to include all rules that may be considered during the transformation process. This is done in the next lemma.

Lemma 4.4.2
$\mathcal{R}^{(v,t)}$ *is complete.*

Proof:
The restrictions on simple-linear terms, self-contained rules, premise-disjoint rules and size-bounded term cause no problem, see lemmas 4.4.5, 4.4.6, 4.4.7 and 4.4.8 with 4.4.9 respectively. It is more difficult to see why allowing only rules with no more then α premise does not affect completeness. This last point is discussed, although not formally proved, in Appendix A.

□

It is not necessarily the case that $\mathcal{R}^{(v,t)}$ is sound, however, we need only completeness and finiteness.

We will often consider the set of size-bounded rules constructed from a given inference system. We call it the extension of an inference system.

Definition 4.4.7 ($S^{(v,t)}$)
Given $S \in SIS$. *The extension of* S *is* $S^{(v,t)} = \mathcal{R}^{(v,t)}$ *where* $v = V_S$ *and* $t = T_S$.

The next two lemmas establish the finiteness of $S^{(v,t)}$ and its power set. This is an important step toward the convergence.

Lemma 4.4.3
Let $S \in SIS$, *then* $S^{(v,t)}$ *is finite.*

Proof:
Since the set $\mathcal{R}^{(v,t)}$ is finite (see Lemma 4.4.1) and $S^{(v,t)} = \mathcal{R}^{(v,t)}$, we deduce that $S^{(v,t)}$ contains a finite number of rules.

□

Lemma 4.4.4
Let $S \in SIS$, then $\wp(S^{(v,t)})$, the power set of $S^{(v,t)}$, is finite.

Proof:
Since $S^{(v,t)}$ is finite, it follows directly that $\wp(S^{(v,t)})$ is also finite.

□

4.4.2 The transformation function

We now give a model representing the computation of the series (see Definition 3.4.5)) associated with an inference system. This representation is called *transformation function*. The transformation function takes an inference system and produces the next inference system of the series.

Definition 4.4.8 ($\Phi(\_)$)
$\Phi(\_) : IS \rightarrow IS$. $\Phi(S) = (NT(S) \nrightarrow T(S))_{\downarrow}$.

The transformation process applied to the inference system S is modeled by:

- $S^0 = S$.
- $S^1 = \Phi(S)$.
- $S^2 = \Phi(\Phi(S))$.
- ...

Thus, to prove the convergence of the transformation process, we show that the recursive application of the transformation function $\Phi(\_)$ on any structured inference system reaches a fixpoint. To do this, we first need to understand its behavior when applied to a structured inference system.

Through the following five lemmas, we show some properties of the function $\Phi(\_)$ when applied to a structured inference system. These properties are: given a structured inference system S, then

- $\Phi(S)$ is a simple-linear inference system;
- Every rule of $\Phi(S)$ is self-contained;
- Every rule of $\Phi(S)$ is premises-disjoint;
- The size of any invariant message in $\Phi(S)$ is not greater than V_S;
- The size of any variant message in $\Phi(S)$ is not greater than T_S;

To prove these lemmas, we assume S is a structured inference system and we pose $S' = \Phi(S)$. To show that S' respects a property already respected by S, it is sufficient to show that every new rule in S' respects the property. Thus we denote R' any rule added to S' by the function $\Phi(S)$. Since R' is added to S', $R' = R \Uparrow (R_1, R_2, \ldots, R_m)$ for some $R \in NT(S)$ and $R_1, R_2, \ldots, R_m \in T(S)$, where:

$$R = \frac{p_1 \quad \cdots \quad p_m}{c} \qquad R_i = \frac{p_{i1} \quad \cdots \quad p_{im_i}}{c_i}$$

$$R' = \frac{p_{11}\sigma \ldots p_{1m_1}\sigma \;\; p_{21}\sigma \ldots p_{2m_2}\sigma \ldots p_{m1}\sigma \ldots p_{mm_m}\sigma}{c\sigma}$$

σ is defined to be $\sigma = \sigma_1 \circ \ldots \circ \sigma_m$, where $\sigma_i = mgu(c_i, p_i)$.

Lemma 4.4.5
If $S \in SIS$, then $\Phi(S)$ will be a simple-linear inference systems.

Proof:
Since R and the R_i are all simple-linear rules, the terms c, c_i, p_i and p_{ij} for $1 \leq i \leq m$ and $1 \leq j \leq m_i$ are all simple-linear. By the definition of σ and property 4.3.7, we know that every affectation $X \leftarrow m$ in σ is such that m is simple-linear. By Proposition 4.3.1, $c\sigma$ and $p_{ij}\sigma$ are all simple-linear.

□

Lemma 4.4.6
If $S \in SIS$, then $\Phi(S) \in SC^{IS}$.

Proof:
If $NAVar(c\sigma) = \emptyset$, R' is vacuously self-contained. We suppose $NAVar(c\sigma) = \{X\}$, which implies $NAVar(c) \neq \emptyset$.

- In the case $NAVar(c) = \{X\}$, we know $NAVar(p_i) = \{X\}$ for some i because $R \in S$ and S is a structured inference system. To have $NAVar(c\sigma_i) = \{X\}$, we must have $NAVar(c_i\sigma_i) = NAVar(p_i\sigma_i) = \{X\}$; this can hold only if $NAVar(c_i) = \{Y\}$ and σ_i contains $Y \leftarrow m$ where $X \in NAVar(m)$ ($\sigma_i = mgu(p_i, c_i)$). Since $NAVar(c_i) = \{Y\}$, $\exists k$ such that $NAVar(p_{ik}) = \{Y\}$ (because $R_i \in S$ and S is a structured inference system). $NAVar(p_{ik}\sigma_i) = \{X\}$ by the restriction imposed on σ_i above. Since σ_i is the only one to manipulate X and Y and since the composition with the other *mgu* does not change the non-atomic variables, we conclude that $NAVar(p_{ik}\sigma) = \{X\}$.
- Otherwise, $NAVar(c) = \{Y\}$ and then $NAVar(p_i) = \{Y\}$ for some i. The same reasoning applies here with the sole difference that it is required to have $NAVar(c_i) = \{X\}$.

□

Lemma 4.4.7
If $S \in SIS$, then $\Phi(S) \in PD^{IS}$.

Proof:
Let i and j range over $[1 \dots m]$, k ranges over $[1 \dots m_i]$ and l over $[1 \dots m_j]$ with the restriction that either $i \neq j$ or $k \neq l$. We know that $Var(p_{ik}) \cap Var(p_{jl}) = \emptyset$ because:

- if $i = j$, p_{ik} and p_{jl} are two premises of the rule R_i and R_i is premises-disjoint because $R_i \in S$ and S is a structured inference system;
- if $i \neq j$, p_{ik} and p_{jl} belong to two rules R_i and R_j respectively and $Var(R_i) \cap Var(R_j) = \emptyset$ (by renaming if R_i and R_j are two instances of the same rule in the inference system).

We suppose $Var(p_{ik}) = X$ and $Var(p_{jl}) = Y$, otherwise the condition trivially holds.

- In the case $i = j$ (then $l \neq k$):
  - If $Var(c_i) = X$, then $Var(\sigma(p_{il})) = Y$ while $Var(\sigma(p_{ik}))$ is either X, $\emptyset$ or $Var(p_i)$ none of which can be Y.

  - If $Var(c_i) = Y$, then $Var(\sigma(p_{ik})) = X$ while $Var(\sigma(p_{il}))$ is either Y, $\emptyset$ or $Var(p_i)$ none of which can be X.
  - If $Var(c_i) \neq Y$ and $Var(c_i) \neq X$, then $Var(\sigma(p_{il})) = Y$ while $Var(\sigma(p_{ik})) = X$.

- In the case $i \neq j$:
  - If $Var(c_i) \neq X$ and $Var(c_j) \neq Y$, then $Var(\sigma(p_{jl})) = Y$ while $Var(\sigma(p_{ik})) = X$.
  - If $Var(c_i) = X$ and $Var(c_j) = Y$, then $Var(\sigma(p_{ik}))$ is either X, $\emptyset$ or $Var(p_i)$ while $Var(\sigma(p_{jl}))$ is either Y, $\emptyset$ or $Var(p_j)$. Since $Var(p_i) \cap Var(p_j) = \emptyset$ (R is premises-disjoint because it is part of a structured inference system), $Y \notin Var(p_i)$ (because $Y \in Var(R_j)$ implies $Y \notin Var(R_i)$) and $X \notin Var(p_j)$, it holds that $Var(p_{ik}) \cap Var(p_{jl}) = \emptyset$.
  - If $Var(c_i) = X$ and $Var(c_j) \neq Y$, then $Var(p_{jl}) = Y$ while $Var(p_{ik})$ is either X, $\emptyset$ or $Var(p_i)$, none of which can be Y.
  - If $Var(c_i) \neq X$ and $Var(c_j) = Y$, then $Var(p_{ik}) = X$ while $Var(p_{jl})$ is either Y, $\emptyset$ or $Var(p_j)$, none of which can be X.

□

Lemma 4.4.8
If $S \in SIS$, then $V_{\Phi(S)} \leq V_S$.

Proof:
Suppose $NAVar(m\sigma) \neq \emptyset$, it implies $NAVar(m) = \{X\}$. If $X \notin Dom(\sigma)$, then clearly $|m\sigma| = |m| \leq V_S$.

- If $m = c$: Let p_i be the premise of R such that $NAVar(p_i) = NAVar(c) = \{X\}$. Since $R \in NT(S)$, $|c| \leq |p_i|$ and $|c\sigma_i| \leq |p_i\sigma_i|$. By Proposition 4.3.5, $|p_i\sigma_i| \leq max(|p_i|, |c_i|)$, which is clearly $\leq V_S$. Note that property 4.3.8 assures us to have $|p_i\sigma_i| = |p_i\sigma|$.

- If $m = p_{ab}$: $NAVar(c_a) = \{X\}$, otherwise, $X \notin Dom(\sigma)$. Since $R_a \in T(S)$, $|p_{ab}| < |c_a|$ and $|p_{ab}\sigma_a| < |c_a\sigma_a|$. By Proposition 4.3.5, $|c_a\sigma_a| \leq max(|c_a|, |p_a|)$, which is clearly $\leq V_S$. Note that property 4.3.8 assures us to have $|c_a\sigma_a| = |c_a\sigma|$.

□

Lemma 4.4.9
If $S \in SIS$, then $C_{\Phi(S)} \leq T_S$.

Proof:
In the case $NAVar(m) = \emptyset$, then clearly $|m\sigma| = |m| \leq T_S$. Thus, we suppose $NAVar(m) = \{X\}$ and $NAVar(m\sigma) = \emptyset$.

- If $m = c$: let p_i be the premise of R such that $NAVar(p_i) = \{X\}$. Since $R \in NT(S)$, $|c| \leq |p_i|$ and $|c\sigma_i| \leq |p_i\sigma_i|$. By Proposition 4.3.6, we have:
  - if $NAVar(c_i) \neq \emptyset$, then $|p_i\sigma_i| < |p_i| + |c_i| \leq 2 \times V_S \leq T_S$.
  - else $|p_i\sigma_i| = |c_i\sigma_i| = |c_i| \leq T_S$.

  Note that property 4.3.8 assures us that $|p_i\sigma_i| = |p_i\sigma|$ and in the same way, we know $|c\sigma_i| = |c\sigma|$.
- If $m = p_{ab}$: then $NAVar(c_a) = \{X\}$, otherwise $X \notin Dom(\sigma)$ and $NAVar(p_{ab}\sigma) \neq \emptyset$. Since $R_a \in T(S)$, $|p_{ab}| < |c_a|$ and $|p_{ab}\sigma_a| < |c_a\sigma_a|$. By Proposition 4.3.6, we have:
  - if $NAVar(p_a) \neq \emptyset$, then $|c_a\sigma_a| < |c_a| + |p_a| \leq 2 \times V_S \leq T_S$.
  - else $|c_a\sigma_a| = |p_a\sigma_a| = |p_a| \leq T_S$.

  Note that property 4.3.8 assures us that $|c_a\sigma_a| = |c_a\sigma|$ and in the same way, we know $|p_{ab}\sigma_i| = |p_{ab}\sigma|$.

□

We are now ready to give more general properties for $\Phi(\_)$ when applied to structured inference systems.

The next theorem shows that the application of the transformation function to a structured inference system will yield a structured inference system.

Theorem 4.4.1 (*SIS* is closed under $\Phi(\_)$)
If $S \in SIS$, then $\Phi(S) \in SIS$.

Proof:
It follows directly from lemmas 4.4.5 to 4.4.7 and from the fact that $\Phi(S)$ is clearly simplified (see the definition of Φ).

□

The next theorem shows that the set $\wp(S^{(v,t)})$ is closed under the transformation process (remember that in the definition of $S^{(v,t)}$, it is required that S be a structured inference system). This result will be very important to define a new, restricted, transformation function.

Theorem 4.4.2 ($\wp(S^{(v,t)})$ is closed under $\Phi(\_)$)
Let $S \in SIS$, then $S \subseteq S^{(v,t)}$ and $\Phi(S) \subseteq S^{(v,t)}$.

Proof:
Suppose S is a structured inference system and let $S' = \Phi(S)$. By the definition of $S^{(v,t)}$, it is clear that $S \subseteq S^{(v,t)}$. By Theorem 4.4.1, S' is a structured inference system. From Lemma 4.4.8, we know that $V_{S'} \leq V_S$ and from Lemma 4.4.9 that $C_{S'} \leq T_S$. We can conclude that $S' \subseteq S^{(v,t)}$.

□

The transformation function of Definition 4.4.8 represents the general pattern of composition. We are interested here in showing that the transformation process converges on every structured inference system. That is why we define a restricted version of our transformation function and call it the restricted transformation function. It is restricted on its domain and codomain.

Definition 4.4.9 ($\Phi_{S^{(v,t)}}(\_)$)
Let S be a structured inference system, $S^{(v,t)}$ be its extended inference system and $S' \in \wp(S^{(v,t)})$. $\Phi_{S^{(v,t)}}(\_) : \wp(S^{(v,t)}) \rightarrow \wp(S^{(v,t)})$. $\Phi_{S^{(v,t)}}(S') = \Phi(S')$.

It is important to prove that the codomain of $\Phi_{S^{(v,t)}}(\_)$ is really $\wp(S^{(v,t)})$. This is done in Proposition 4.4.1.

Proposition 4.4.1
Let $S' \in \wp(S^{(v,t)})$, then $\Phi_{S^{(v,t)}}(S') \in \wp(S^{(v,t)})$.

Proof:
Let $S' \in \wp(S^{(v,t)})$. By Theorem 4.4.1, $\Phi(S')$ is also a structured inference system and by Theorem 4.4.2, $\Phi(S') \in \wp(S^{(v,t)})$.

□

The transformation process is modeled by the recursive application of the transformation function on a given inference system. We now define formally what is the recursive application of the restricted transformation function on a given structured inference system. The generalization to the transformation function of Definition 4.4.8 and to all inference systems follows easily.

Definition 4.4.10 (Recursive application of $\Phi_{S^{(v,t)}}(\_)$)
Given an inference system S, the recursive application of the restricted transformation function on S is modeled by the series $\Phi_{S^{(v,t)}}(S)$, $\Phi_{S^{(v,t)}}(\Phi_{S^{(v,t)}}(S)), \ldots \Phi^n_{S^{(v,t)}}(S), \ldots$ The recursivity stops as soon as we reach a point i such that $\Phi^i_{S^{(v,t)}}(S) = \Phi^{i+1}_{S^{(v,t)}}(S)$. We say that i is a fixpoint for $\Phi_{S^{(v,t)}}(\_)$.

We now have a more specific model for the transformation process applied on structured inference systems. The convergence proof will be reduced to prove that the recursive application of $\Phi_{S^{(v,t)}}(\_)$ on S reaches a fixpoint for any structured inference system S. From now on, we say that $\Phi_{S^{(v,t)}}(\_)$ has a fixpoint to mean that the recursive application of $\Phi_{S^{(v,t)}}(\_)$ on S reaches a fixpoint.

Lemma 4.4.10
Let $< P, \leq >$ be a finite poset with a top element $\top$. If F is a growing[6] map on P, then the series $F(x), F(F(x)), \ldots, F^n(x), \ldots$ reaches a fixpoint for any $x \in P$.

Proof:
Suppose the series $F(x), F(F(x)), \ldots, F^n(x), \ldots$ never reaches a fixpoint. Since F is a growing map, we know $F(x) < F(F(x)) < \ldots < F^n(x) < \ldots$ In particular, we cannot have $F^i(x) = F^{i+1}(x)$ because such an i would itself be a fixpoint. By definition of the top element, we know that $F^i(x) \leq \top$ for any i. Since the set P is finite by hypothesis, we cannot have infinitely many elements of P strictly smaller than $\top$. We then have a contradiction with the assumption that the series does not reach a fixpoint.

□

4.4.3 The convergence proof

If we consider the set $\wp(S^{(v,t)})$ together with the order relation $\subseteq$, it is easy to see that $< \wp(S^{(v,t)}), \subseteq >$ is a partially ordered set with bottom[7] element $\bot = \emptyset$ and top element

[6] A function F is called growing if $x \leq F(x)$ for any element x in the domain of F.
[7] The bottom element of a poset $< P, \leq >$ is an element $\bot$ such that $\bot \leq x$ for every $x \in P$

$\top = S^{(v,t)}$. We know that $\wp(S^{(v,t)})$ is finite from Lemma 4.4.4 and thus we need only to show that the restricted transformation function is indeed a growing map on $\wp(S^{(v,t)})$ to conclude that it reaches a fixpoint. This is done in Lemma 4.4.11

Lemma 4.4.11
$\Phi_{S^{(v,t)}}(\_)$ *is a growing map.*

Proof:
It is clear that $\Phi_{S^{(v,t)}}(\_)$ is a total map since for any $S' \in \wp(S^{(v,t)})$, there is a $S'' \in \wp(S^{(v,t)})$ such that $\Phi_{S^{(v,t)}}(S') = S''$. The fact that $\Phi_{S^{(v,t)}}(\_)$ is growing, that is $S' \subseteq \Phi_{S^{(v,t)}}(S')$, follows directly from the definition of Φ (considering the new definition of simplified inference system, see Definition 4.4.1 and the note on page 78).

□

Finally, we present the main theorem of our thesis, the transformation process converge when applied on a structured inference system.

Theorem 4.4.3 (The transformation process converges for any $S \in SIS$**)**
Given $S \in SIS$*, the transformation process converges when applied to* S*.*

Proof:
Let S be a structured inference system. It is sufficient to show that $\Phi_{S^{(v,t)}}(\_)$ has a fixpoint to conclude that the transformation process converges for S since the transformation process applied on S corresponds to the recursive application of the restricted transformation function $\Phi_{S^{(v,t)}}(S)$, $\Phi_{S^{(v,t)}}(\Phi_{S^{(v,t)}}(S)), \ldots$ Since $\wp(S^{(v,t)})$ is a finite poset with a top element and since $\Phi_{S^{(v,t)}}(\_)$ is a growing map from $\wp(S^{(v,t)})$ to $\wp(S^{(v,t)})$ (see Lemma 4.4.11), it follows from Lemma 4.4.10 that $\Phi_{S^{(v,t)}}(\_)$ has a fixpoint.

□

Now that the convergence proof has been established, another important proof remains to be done. We need to show, as claimed in 4.4.5, that $T(S)$ is a terminating inference system whenever S is a structured inference system. That is we need to show that the theory of $T(S)$ is decidable.

4.5 The resulting inference system is terminating

When applied to a structured inference system, the transformation process converges, as shown in Theorem 4.4.3. The transformation process will provide a resulting inference system S such that:

- S is a structured inference system since we showed in Theorem 4.4.1 that the function $\Phi(\_)$ preserves the structured property;
- $S \equiv T(S)$ since the transformation process returns a set of terminating rules.

From now on, when we talk about a resulting inference system, we mean a structured inference system composed only of terminating rules. We use *RIS* to denote the set of all resulting inference systems.

For any rule in a terminating inference system we can classify its premises in three different groups, these groups being defined by the non-atomic variables in the premises. A premise can always contain some atomic variables.

Definition 4.5.1 (Invariant premise)
A premise p is invariant when $NAVar(p) = \emptyset$ (the premise contains no non-atomic variable). Thus, whenever p is an invariant premise, we have $|\sigma(p)| = |p|$ for any σ.

Definition 4.5.2 (Free premise)
A premise p is said to be free in the rule R when $NAVar(p) = X$, but $NAVar(\mathcal{C}(R)) \neq X$.

Definition 4.5.3 (Linked premise)
A premise p is said to be linked in the rule R when $NAVar(p) = NAVar(\mathcal{C}(R)) = X$. Note that there is at most one linked premise in every rule of a resulting inference system.

In Definition 3.3.3 of the previous chapter, we established a sufficient condition to assure the termination of an inference system with respect to backward chaining. It is easy to see that we cannot find such a general terminating ordering for all resulting inference system. One of the reasons is that any ordering is not necessarily preserved under substitution, as explained in Example 4.5.1. Moreover, applying basic backward chaining could easily lead to infinite computation as shown in Example 4.5.2. The two examples are built within the simple-linear algebra of cryptographic protocols, as discussed in Example 4.2.1.

Example 4.5.1 (Substitution problem)
Take the following rule:

$$R = \frac{E(C(p_1, X_1), K(p_1, p_2)) \quad E(X_2, K(p_3, p_4))}{E(C(p_3, C(p_4, C(p_5, X_2)), K(p_3, p_5))}$$

Clearly, R can be part of a resulting inference system since it respects the conditions mentioned above. It is also clear that no ordering can be preserved under substitution, w.g. $\sigma = \{X_2 \leftarrow p_6, X_1 \leftarrow C(p_4, C(p_5, C(p_6, E(X_3, K(p_7, p_8)))))\}$. The message $\sigma(E(C(p_1, X_1), K(p_1, p_2)))$ will be bigger than $\sigma(E(C(p_3, C(p_4, C(p_5, X_2)), K(p_3, p_5)))$.

Example 4.5.2 (Infinite backward chaining computation)
Take the following rule:

$$R = \frac{E(C(p_1, C(p_2, p_3)), K(p_4, p_5)) \quad E(X_1, K(p_4, p_5))}{E(C(p_1, C(p_2, X_1)), K(p_4, p_5))}$$

Clearly, R can be part of a resulting inference system. Now suppose R is used in the search of a proof for m. That is $E(C(p_1, C(p_2, X_1)), K(p_4, p_5))$ is unified with m through the mgu σ. Now, the search of a proof for m is reduced to the search of a proof for both $\sigma(E(C(p_1, C(p_2, p_3)), K(p_4, p_5)))$ and $\sigma(E(X_1, K(p_4, p_5)))$. But to prove the message $\sigma(E(C(p_1, C(p_2, p_3)), K(p_4, p_5)))$, one can surely use the rule R with $\sigma' = mgu(\sigma(E(C(p_1, C(p_2, p_3)), K(p_4, p_5))), E(C(p_1, C(p_2, X_1)), K(p_4, p_5)))$. We now seek to prove $\sigma'(E(C(p_1, C(p_2, p_3)), K(p_4, p_5)))$ and $\sigma'(E(X_1, K(p_4, p_5)))$. But again, to prove $\sigma'(E(C(p_1, C(p_2, p_3)), K(p_4, p_5)))$, one can use rule R ... and so on.

In this section, we propose a proof-search procedure very similar to backward chaining. We argue that this modification is sufficient to yield a terminating proof-search procedure. From there, we conclude that the resulting inference system will always be a terminating one when the initial inference system is structured.

The intuition behind our modification of the backward chaining algorithm is based on the two examples above. First, we observe that the problem pointed by Example 4.5.1 is not present in the backward chaining procedure (the free premises do not grow when using a rule). Then, the problem pointed in Example 4.5.2 is addressed by adding a loop detection to the backward chaining algorithm (this is our modification).

4.5.1 Proof-search procedure

In Table 4.1 we present our proof-search procedure. The procedure takes as parameters: a list of objectives O to prove, a set of messages *Proving* that represent every messages

---

ProofSearch$(O, Proving, S)$:
  IF $(O = \emptyset)$
    RETURN TRUE
  ELSE
    LET $O =< o_1, o_2, \ldots, o_n >$ IN
      $O_1 = O - < o_1 >$
      $Proving = Proving \cup \{o_1\}$
      FOR ALL $R \in S$
        $(O_2, \sigma) = ApplyRule(o_1, R)$
        IF $(\sigma \neq$ NULL$)$ AND $(O_2 \cap Proving = \emptyset)$
          IF $(ProofSearch(O_2 \dot{+} O_1\sigma, Proving, S))$
            RETURN TRUE
      RETURN FALSE

ApplyRule(o, R):
  $\sigma = mgu(o, \mathcal{C}(R))$
  $O = \mathcal{P}(R)\sigma$
  RETURN (O, σ)

Table 4.1: The proof-search procedure

we are currently trying to prove (the latter is used to detect loops as explained in Example 4.5.2) and a resulting inference system S. When first invoking the procedure to prove the simple-linear term m, the set *Proving* is empty.

The operator $\dot{+}$ appends two lists and removes the multiple occurrences of an element (we only keep the first occurrence of every element).

To prove a simple-linear term t, the algorithm will try every available rule until either all rules have been tried or a proof for t has been found. If the utilization of a rule R results in proving a term t' that we are already trying to prove ($t' \in proving$), then it is useless to use R. This last point is the main difference between our proof-search procedure and the backward chaining, it will also be important to prove the termination of our proof-search procedure. The set *proving* has the following meaning. When we try to prove t, we use R in a backward manner and generate $\{p\sigma | p \in \mathcal{P}(R)\}$.

If we can prove those $p\sigma$, then t is proved. So we are trying to prove t, but the objectives we are working with are the $p\sigma$. What if there is a $p\sigma$ such that $p\sigma = t$, that is to prove t, using rule R, we need to prove t and some other messages? From there, if there is a proof for all $p\sigma$ (including the one equal to t), then there exists surely a proof for t without using R. That is why the algorithm backtracks when the application of a rule R forces us to prove a message m that we are currently trying to prove. Given a list of objectives $O =< o_1, o_2, \ldots, o_n >$, it is sufficient to try o_1 with all the rules. In particular, it is not necessary to check for o_2 in the case o_1 cannot be proved since if o_1 cannot be proved, it is clear that O cannot be proved.

From this discussion, it should be clear that the proof-search procedure is complete (if there is a proof for t in S, then the proof-search procedure will find one). The soundness (if the proof-search procedure finds a proof for t, then there is a proof for t in S) of the proof-search procedure follows directly from the soundness of the backward chaining algorithm.

We want to point to the reader a somewhat strange semantic associated with our proof-search procedure. When trying to prove a simple-linear term t such that $Var(t) = X$, one usually implicitly use a universal quantifier over X. In our case, we have to put an existential quantifier over X. That is, t will be provable iff there is a message t' such that $t\sigma$ is provable with $\sigma = \{X \leftarrow t'\}$. Although this is not the meaning one usually wants for t, it is precisely the meaning we want for every free premise. When a variable Y is present in a premise such that Y is not in the conclusion, we are content when Y can be replaced by anything (we do not require that Y be replaced with every possible message).

To prove that this procedure always terminates, we model it by a tree. The possible states of the algorithm will be the nodes of the tree, while an edge will consist in applying the recursive procedure to get from one state to another. We will show the tree is finite, thus implying the algorithm cannot run infinitely.

4.5.2 A tree model for the proof-search procedure

Given a resulting inference system S and a simple-linear term t, we define the tree explored by the proof-search procedure for t in S, denoted T_t^S, as:

Definition 4.5.4 (T_t^S)
T_t^S is an oriented tree where:

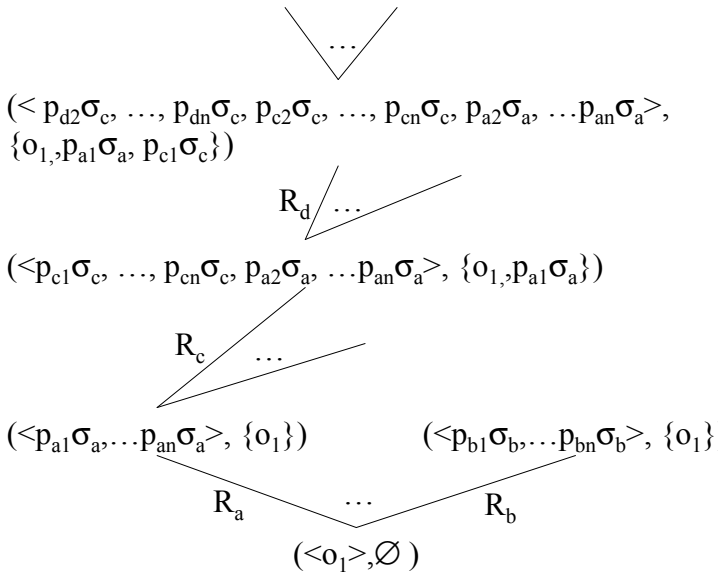

Figure 4.1: The proof-search tree

- *the nodes are pairs* (O, P) *where* O *is a list*[8] *of simple-linear terms (the objectives) and* P *is a set of simple-linear terms (the path);*
- *the root is* $(< t >, \emptyset)$*;*
- *an edge from* (O, P) *to* (O', P') *(denoted* $(O, P) \rightarrow (O', P')$*) represents the application of a rule (in a backward chaining fashion) to the first objective of* O*.* $(O, P) \rightarrow (O', P') \in T_t^S$ *iff there is a* $R \in S$ *and* $O =< o_1, o_2, \ldots, o_n >$ *such that:*
  - $\sigma = mgu(o_1, \mathcal{C}(R))$ *exists and*
  - $O' = \mathcal{P}(R)\sigma \dot{+} (O - < o_1 >)\sigma$ *and*
  - $P' = P \cup \{o_1\}$ *and*
  - $P' \cap \mathcal{P}(R)\sigma = \emptyset$*.*

Such a tree looks like Figure 4.1. When we need to emphasis the fact that node (O', P') is a child (in tree T_t^S) of node (O, P) by the application of rule $R \in S$, we write $(O, P) \rightarrow_R (O', P')$. Nodes will often be referred to by N, we then use N_O and N_P to denote respectively the list O and the set P of node N.

[8] A list is a set where every element has a position. Multiple occurrences of a same element never occurs in our case.

We will show the proof-search procedure terminates by showing the finiteness of T_t^S. Since the proof-search procedure simply explores T_t^S in a depth-first manner, the termination of the procedure is assured by the finiteness of the tree.

We first need to justify that every node (O, P) of T_t^S is such that O is a list of simple-linear terms and P is a set of simple-linear terms. This is done in the next proposition by showing that if a node (O, P) of T_t^S is such that O is a list of simple-linear terms and P is a set of simple-linear terms (as it is the case with the root of the tree), then all its children (O', P') are such that O' is a list of simple-linear terms and P' is a set of simple-linear terms. We use $\mathcal{L}(\mathcal{T})$ for the set of all lists formed with the terms of $\mathcal{T}$.

Proposition 4.5.1
If $(O, P) \in \mathcal{L}(\mathcal{T}) \times \wp(\mathcal{T})$ *and* $(O, P) \rightarrow_R (O', P') \in T_t^S$ *for any* $t \in \mathcal{T}$ *and* $S \in RIS$, *then* $(O', P') \in \mathcal{L}(\mathcal{T}) \times \wp(\mathcal{T})$.

Proof:
Let $O =< o_1, o_2, \ldots, o_n >$. It is clear that $P' \in \wp(\mathcal{T})$, since o_1 is simple-linear and $P \in \wp(\mathcal{T})$. Since O and $\mathcal{P}(R)$ contain only simple-linear terms, we need only to show that σ contains only simple-linear affectation to deduce that O' is a list of simple-linear terms (using Proposition 4.3.1). σ is the *mgu* of two simple-linear terms, namely o_1 and c (c is simple-linear since the rule R is in a resulting inference system), thus by Proposition 4.3.2, σ contains only simple-linear affectations.

□

The next proposition states that if the node (O', P') is a child of the node (O, P), then P' contains exactly one more element then P.

Proposition 4.5.2
If $(O, P) \rightarrow_R (O', P')$, *then* $|P'| = |P| + 1$.

Proof:
Let $O =< o_1, o_2, \ldots, o_n >$. It is sufficient to show that $o_1 \notin P$. Let N be the first node on the path from the root to (O', P') to contain o_1 as an objective. Clearly, $o_1 \notin N_P$, otherwise N cannot be the child of any node in the tree. If $o_1 \in P$, this that means there is a pair of nodes N^1 and N^2 such that $N^1 \rightarrow N^2$ on the path from N to (O, P) and o_1 is the first element of N_O^1. But then, a node N^3 on the path from N^2 to (O, P) inclusively can have o_1 as the first element of N_O^3 (because of the edge restriction in Definition 4.5.4); this is a contradiction with the fact that $O =< o_1, o_2, \ldots, o_n >$.

□

Given a set (or a list) $\mathcal{T}'$ of simple-linear terms and a resulting inference system S, we define the following size measure:

- $C_S(\mathcal{T}')$: $max(C_S, C')$ where C' is the size of the biggest invariant term in $\mathcal{T}'$.
- $V_S(\mathcal{T}')$: $max(V_S, V')$ where V' is the size of the biggest variant term in $\mathcal{T}'$.
- $T_S(\mathcal{T}')$: $max(C_S(\mathcal{T}'), 2 \times V_S(\mathcal{T}'))$.

Let m be a simple-linear term, S a resulting inference system and $R \in S$. The following proposition states that given $\sigma = mgu(m, \mathcal{C}(R))$, every $p \in \mathcal{P}(R)$ is such that if $NAVar(p\sigma) = \emptyset$ then $|p\sigma| \leq T_S(\{m\})$ while if $NAVar(p\sigma) \neq \emptyset$ then $|p\sigma| \leq V_S(\{m\})$.

Proposition 4.5.3
Let $m \in \mathcal{T}$, $S \in RIS$, $R \in S$ and $\sigma = mgu(m, \mathcal{C}(R))$. Then $\forall p \in \mathcal{P}(R)$, $p\sigma \in \mathcal{T}^{(V_S(\{m\}), T_S(\{m\}))}$.

Proof:
For every $p \in \mathcal{P}(R)$, we must show that: if $NAVar(p\sigma) = \emptyset$, then $|p\sigma| \leq T_S(\{m\})$; if $NAVar(p\sigma) \neq \emptyset$, then $|p\sigma| \leq V_S(\{m\})$.

Suppose $NAVar(p\sigma) = \emptyset$:

- In the case p is an invariant premise: since $NAVar(p) = \emptyset$, we know that $|p\sigma| = |p|$ and since $|p| \leq T_S \leq T_S(\{m\})$, it is clear that $|p\sigma| \leq T_S(\{m\})$.
- In the case p is a free premise: it is not possible since in this case we would have $NAVar(p\sigma) \neq \emptyset$.
- In the case p is a linked premise: $(NAVar(p) = NAVar(\mathcal{C}(R) = \{X\})$
  - If $NAVar(m) = \emptyset$: then $|p\sigma| < |c\sigma| = |m\sigma| = |m| \leq T_S(\{m\})$.
  - If $NAVar(m) = \{Y\}$: then σ contains both $X \leftarrow m'$ and $Y \leftarrow m''$ (to have $NAVar(p\sigma) = \emptyset$) such that $|m'| \leq |m|$ (see Proposition 4.3.3). Thus, $|p\sigma| = |p| - 1 + |m'| < |p| + |m| \leq 2 \times max(|p|, |m|) \leq 2 \times V_S(\{m\}$, $V_S(\{m\} = 2 \times V_S(\{m\} \leq T_S(\{m\}$.

Suppose $NAVar(p\sigma) \neq \emptyset$:

- In the case p is an invariant premise: it is not possible since in this case we would have $NAVar(p\sigma) = \emptyset$.
- In the case p is a free premise: let $\{X\} = NAVar(p)$. Since p is free, $X \not\sqsubseteq \mathcal{C}(R))$. Thus, $X \notin Dom(\sigma)$ and hence $|p\sigma| = |p|$. And clearly $|p| \leq V_S(\{m\})$.
- In the case p is a linked premise:$(NAVar(p) = NAVar(\mathcal{C}(R) = \{X\})$. We must have that $NAVar(m) = \{Y\}$ and $Dom(\sigma)$ does not contain both X and Y for $NAVar(p\sigma) \neq \emptyset$ to holds. In the case $X \notin Dom(\sigma)$, then $|p\sigma| = |p| \leq V_S(\{m\}$. In the case $X \in Dom(\sigma)$, then $|p\sigma| < |\mathcal{R}\sigma| = |m\sigma| = |m| \leq V_S(\{m\}$.

□

The next Lemma will be very important to prove that the proof-search procedure always terminates.

Lemma 4.5.1
Let $(O, P) \rightarrow (O', P') \in T_t^S$, for any given simple-linear term t and resulting inference system S. Then $O' \in \mathcal{L}(\mathcal{T})^{(V_S(O), T_S(O))}$.

Proof:
Let $o \in O$ and $R \in S$ be such that $(O, P) \rightarrow_R (O', P')$.
We will show that $V_S(O') \leq V_S(O)$ and that $C_S(O') \leq C_S(O)$. It will directly imply that $O' \in \mathcal{L}(\mathcal{T})^{(V_S(O), T_S(O))}$ since O' is known to be a list of simple-linear terms (see 4.5.1).
$\underline{V_S(O') \leq V_S(O)}$:
Since $V_S(O') = max(V_S, V')$ and clearly $V_S \leq V_S(O)$, we need only to show that $V' \leq V_S(O)$ (that is we need to show that for every $o \in (O'_V - O_V)$, $|o| \leq V_S(O)$). But this is easily deduced from Proposition 4.5.3.
$\underline{C_S(O') \leq C_S(O)}$:
Since $T_S(O') = max(C_S(O'), 2 \times V_S(O'))$ and that $V_S(O') \leq V_S(O)$ (see the case above), we have $T_S(O') = max(C_S(O'), 2 \times V_S(O))$. It is clear that $2 \times V_S(O)) \leq T_S(O)$ (by the definition of $T_S(O)$). Thus it remains only to show that $C_S(O') \leq T_S(O)$. Since $C_S(O') = max(C_S, C')$ and clearly $C_S \leq T_S(O)$, it is sufficient to show that $C' \leq T_S(O)$ (that is we need to show that for every $o \in (O'_C - O_C$, $|o| \leq T_S(O))$. But, again, this can be deduced from Proposition 4.5.3.

□

As a corollary of Lemma 4.5.1; we have, given a simple-linear term t and a resulting inference system S, every node (O, P) in the tree T_t^S is such that $V_S(O) \leq V_S(\{t\})$ and $T_S(O) \leq T_S(\{t\})$. This is proved below.

Corollary 4.5.1
Let (O, P) be any node in T_t^S, for any given simple-linear term t and resulting inference system S. Then $O \in \mathcal{L}(\mathcal{T})^{(V_S(\{t\}),T_S(\{t\}))}$.

Proof:
It follows inductively on the depth of the nodes from Lemma 4.5.1.

□

Now, for every node (O, P) in the tree T_t^S, we have that $V_S(P) \leq V_S(\{t\})$ and $T_S(P) \leq T_S(\{t\})$.

Lemma 4.5.2
Let (O, P) be any node in T_t^S, for any given simple-linear term t and resulting inference system S. Then $P \subseteq \mathcal{T}^{(V_S(\{t\}),T_S(\{t\}))}$.

Proof:
Let o be any element of P. It is clear that there is a node $(O', P') \in T_t^S$ such that $o \in O'$. By Corollary 4.5.1, we know that $o \in \mathcal{T}^{(V_S(\{t\}),T_S(\{t\}))}$. We then conclude that $P \subseteq \mathcal{T}^{(V_S(\{t\}),T_S(\{t\}))}$.

□

We now use the fact that P cannot grow infinitely to show that the depth of tree T_t^S is bounded. *Depth*(T) stands for the depth of the tree T.

Lemma 4.5.3
Depth$(T_t^S) \leq |\mathcal{T}^{(V_S(\{t\}),T_S(\{t\}))}|$, for any given simple-linear term t and resulting inference system S.

Proof:
It is clear that if the node (O, P) is at depth i in the tree T_t^S, $|P| = i$ holds (this is a consequence of Proposition 4.5.2 and the fact that if N is the root, then $|N_P| = 0$). Since we know by Lemma 4.5.2 that $P \subseteq \mathcal{T}^{(V_S(\{t\}),T_S(\{t\}))}$, we cannot have a node of depth greater than $|\mathcal{T}^{(V_S(\{t\}),T_S(\{t\}))}|$.

□

We now prove that the tree T_t^S is finite (it contains a finite number of nodes).

Theorem 4.5.1 (T_t^S is finite)
T_t^S has a finite number of nodes, for any given simple-linear term t and resulting inference system S.

Proof:
It should be clear from the facts that the depth of T_t^S is finite (see Lemma 4.5.3) and that every node has a finite number of children (a node has at most $|S|$ children).

□

We claim that $\sum_{i=0}^{|\mathcal{T}^{(v,t)}|} |S|^i$ is an upper bound on the number of nodes in T_t^S. It is easy to see by induction on i that there is at most $|S|^i$ nodes at depth i (there is only the root at depth 0, and every node has at most $|S|$ child). Since there is no node deeper than $|\mathcal{T}^{(v,t)}|$ (see Lemma 4.5.3), $\sum_{i=0}^{|\mathcal{T}^{(v,t)}|} |S|^i$ is effectively an upper bound on the number of nodes in T_t^S.

4.5.3 The proof-search procedure terminates

We are now ready to show that the proof-search procedure given in Section 4.5.1 terminates

Theorem 4.5.2 (The proof-search procedure terminates)
The proof-search procedure of Table 4.1 terminates when proving a simple-linear term in a resulting inference systems.

Proof:
Let S be a resulting inference system and t a simple-linear term. The tree T_t^S is finite by Theorem 4.5.1. Since the proof-search procedure simply searches (in a depth-first manner) through T_t^S, for a proof for t, then it cannot run infinitely.

□

4.6 Conclusion

We defined a class of inference systems (structured inference systems) and a way to partition the rules such that the transformation algorithm is shown to be convergent. We then proposed a proof-search procedure (a simple extension of backward chaining) to decide the theory of any resulting inference system. This procedure was shown to terminate. We can now claim this new result: *the theory of any structured inference system is decidable*. It is well known that inference systems are good models for many different problems. In the next chapter, we will see a direct application of this new result; that is, some security properties are decidable for every cryptographic protocol whose inference system is structured. This application of our result will illustrate its importance since, as we will see, a wide class of interesting protocols are modeled by structured inference systems. Of course, many other fields could benefit from this result.

Chapter 5

Decidability of Cryptoprotocols

Abstract

In this chapter, we propose a decision procedure for a class of cryptographic protocols with respect to some basic security properties. This decision procedure is an application of the result stated on the previous chapter *"the theory of any structured inference system is decidable"*.

5.1 Introduction

As we saw in Chapter 2 Section 2.2.1, a cryptographic protocol can be modeled by an inference system and some security properties by messages to be proved in the inference system. We use here the transformation process presented in Chapter 3 together with the convergence proof (for a restricted class of inference systems) of this transformation process to decide some basic security properties about a wide class of cryptographic protocols.

The rest of this chapter is structured as follows: we start by giving an overview of the whole decision procedure in Section 5.2; we then argue, in Section 5.3, that the structured property is too restrictive for cryptographic protocols, we relax this condition and show this does not affect the convergence; Section 5.4 argue that our decision procedure can be applied to a wide class of cryptographic protocols; Section 5.5, illustrates some of the basic security properties that can be decided with our procedure; Section 5.6 discusses the impact of an associative function in the algebra; finally, Section

5.7 presents our Java implementation of the inference system transformation algorithm presented in Section 3.5.

5.2 A decision procedure

In this section, we explain the overall decision procedure for the security of a class of cryptographic protocols. This method cannot be applied to every security property, in Section 5.5 we will see some security properties suitable for our method. The decision procedure we explain here is pictured in Figure 5.1.

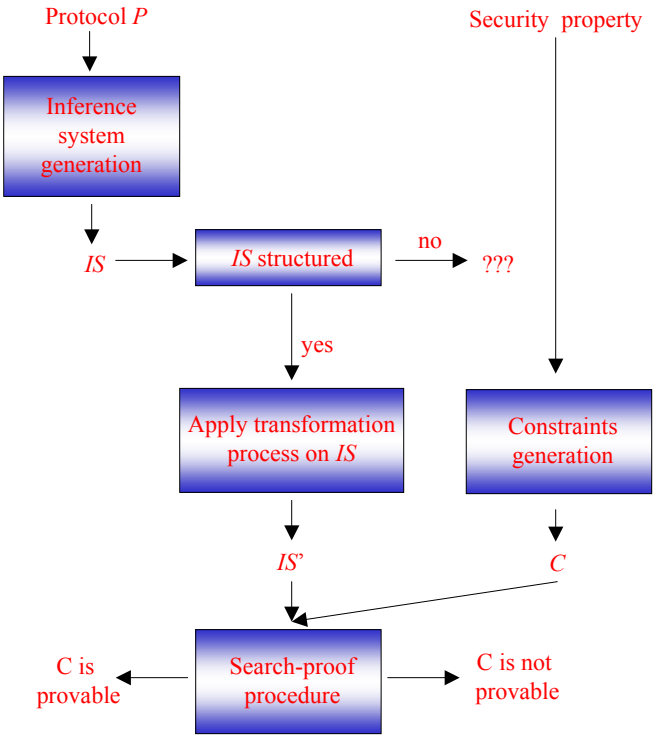

Figure 5.1: A decision procedure

We start by extracting, from a given protocol P and a given security property p, the inference system S together with a set of constraints C. This is done exactly as in the DYMNA approach: the inference system extraction is explained in Section 2.2.1, the constraints generation for the security property differ from one property to another, see

Section 5.5 for some examples. Next, we check if S is a structured[1] inference system. In the case S is not, we cannot decide whether or not P satisfies the security property p. We can either use an alternative method (one of those presented in Chapter 2) or apply our method risking non-termination. On the other hand, if S is structured, we can apply the transformation algorithm presented in Section 3.5 to get an equivalent inference system S'. The transformation algorithm converges, thanks to Theorem 4.4.3. The resulting inference system consists only of terminating rules as defined in 4.4.5. We can then use the proof-search procedure presented in Section 4.5.1 to check if the constraints C are provable in the inference system, remember that the proof-search procedure is known to terminate (see Theorem 4.5.2). The proof-search procedure decides whether or not every constraint of C is provable in S'. From there, we can decide if the protocol P respects the security property p.

5.3 Relaxing the structured requirement

As we explained in Section 5.2, we first check if the inference system underlying a cryptographic protocol is structured. In the affirmative, we can decide the security of the protocol. Unfortunately, as we will see here, no inference system modeling a protocol is structured due to the presence of some intruder's abilities. However, we can isolate the problematic rules and show that the presence of these rules does not alter neither the convergence of the transformation process nor the termination of the proof-search procedure. We first recall the message algebra used for cryptographic protocols, then we point which intruder's rules are not structured[2] and we show that the presence of these particular rules does not affect our results.

5.3.1 Cryptoprotocols message algebra

As in Example 4.2.1, we use the five types $Principals$, $Keys$, $Nonces$, Int and Msg with the partial order defined to be: $Principals < Msg$, $Keys < Msg$, $Nonces < Msg$ and $Int < Msg$; only $Principals$ and Int are atomic types. We use the binary functions:

- $E : Msg \times Keys \rightarrow Msg$;
- $C : Msg \times Msg \rightarrow Msg$.

[1] We will relax the requirement that S be structured in Section 5.3

[2] A rule is structured if it can be in a structured inference system.

- $K : \mathit{Principals} \times \mathit{Principals} \rightarrow \mathit{Keys}$.
- $N : \mathit{Principals} \times \mathit{Int} \rightarrow \mathit{Nonces}$.

to denote respectively the encryption, the concatenation, the key sharing and the nonce manipulation. We use X_i for variables of type *Msg* and P_i for variables of type *Principals*. The constant used are generally limited to A, I and S for *Principals* and m for *Msg*. Other constant are defined when needed. Note that we do not use variables of type *Keys* nor *Nonces*.

We assume no equational theory is associated with these functions; in particular C is not associative[3]. In order to lighten the notation, we use $\{m\}_k$ instead of $E(m,k)$, $m_1.m_2$ to mean $C(m_1.m_2)$, $K_{P_iP_j}$ is used instead of the formal $K(P_i,P_j)$ and $N^{\alpha}_{P_i}$ means $N(P_i,\alpha)$. Since C is not associative, we assume the term $m_1.m_2.m_3$ corresponds to $C(m_1,C(m_2,m_3))$. Note that the encryption function allows to encrypt a message with a key only. In particular, we cannot encrypt a message m with another message m' as long as m' is not typed as a key.

The algebra must reflect the reality as much as possible. We believe only keys should be used to encrypt messages (see the definition of the encryption function in Section 1.2.1), and thus this restriction is realistic. But we think concatenation should be associative, and thus this restriction is important. We assume from now on that the implementation of a cryptographic protocol respects these restrictions.

Note that the way the key function is defined force the cryptosystem to be symmetric. We will concentrate only on such protocols but it should cause no problem to include asymmetric ones. More importantly, in the way keys are denoted, it is not permitted to have multiple keys shared between principals P and P'[4]. This is particularly annoying when dealing with key exchange protocols. This problem can be overcome easily by adding a number to the key representation ($K^i_{PP'}$ for some integer i).

Since our algebra is very simple, we can perform a simplification on the rule of our inference systems. If a rule R contains the premise p such that $p = X$, $X \not\sqsubseteq \mathcal{C}(R)$ and X is a non-atomic variable, then we can remove p from R without changing its meaning. It is clear that the intruder can provide a message to fill the premise p_i since: *Msg* is the only non-atomic type used for variables and if the type of X is *Msg*, then the intruder can provide a message as long as there is an axiom in the inference system (having no axiom implies an empty-theory).

[3] We discuss briefly in Section 5.6 the case where C is associative

[4] In particular, we should treat $K_{PP'}$ as being the same as $K_{P'P}$, a non-problematic commutative theory for the key function that we ignore for simplicity.

5.3.2 Problematic rules

In the set of rules modeling the intruder usual abilities (see the rules of Table 5.1), the encryption (R_e), decryption (R_d) and knowledge[5] ($R_{\mathcal{K}}$) rules cause no problem since they are clearly structured. However, the other three rules the concatenation (R_c) and both decomposition (R_{d_1} and R_{d_2}) rules are not structured. In fact, the terms forming the conclusion of R_c and the premise of R_{d_1} and R_{d_2} are not simple-linear.

$R_c = \frac{X \quad Y}{X.Y}$	$R_{d_1} = \frac{X.Y}{X}$	$R_{d_2} = \frac{X.Y}{Y}$
$R_e = \frac{X \quad Y}{\{X\}_Y}$	$R_d = \frac{\{X\}_Y \quad Y}{X}$	$R_{\mathcal{K}} = \frac{m \in \mathcal{K}}{m}$

Table 5.1: Intruder's rules

Since every inference system modeling a cryptographic protocol must consider these basic intruder's abilities, no such inference system can be structured. Fortunately, we will show that the presence of the concatenation and decomposition rules does not change the convergence result of the previous chapter.

5.3.3 Adapting the convergence result

We first adapt our definition of structured inference systems. Our new definition includes the problematic rules stated above. This new definition allows us to work with inference systems that model cryptographic protocols; we call them augmented-structured inference systems. We will show in this section that the main result of Chapter 4 on structured inference systems can be extended to augmented-structured inference systems.

Definition 5.3.1 (Augmented-structured inference system (SIS_+))
$S \in SIS_+$ if $S = S' \cup \{R_c, R_{d_1}, R_{d_2}\}$ where $S' \in SIS$.

Using this new class of inference system, one can define the class of cryptographic protocols for which we will have a decision procedure (for a restricted set of security

[5]Messages in the intruder knowledge are required to be invariant.

properties).

Definition 5.3.2 (Structured cryptographic protocols)
A protocol is structured if its underlying inference S is augmented-structured. That is if $S \in SIS_+$

As in the previous chapter, we need to work with an extension of augmented-structured inference system. This extension is very similar to the one of Definition 4.4.7, except that it includes the problematic rules specific to cryptoprotocols.

Definition 5.3.3 ($S_+^{(v,t)}$)
Let $S \in SIS$, we define $S_+^{(v,t)}$ to be $S^{(v,t)} \cup \{R_c, R_{d_1}, R_{d_2}\}$.

It is clear that $< \wp(S_+^{(v,t)}), \subseteq >$ is a finite poset with $S_+^{(v,t)}$ as its top element. If we can show that the power set of augmented-structured inference systems is closed under the transformation process, the convergence will follow exactly as in Section 4.4.3. Since we know that the power set of structured inference systems is closed under the transformation process, we need only to show that if R' is the result of a composition including at least one problematic rule, then $R' \in S^{(V_S,T_S)}$. This is done in Theorem 5.3.1 with the help of the following two lemmas.

Since the rules R_c, R_{d_1} and R_{d_2} cannot be classified terminating or non-terminating by Definition 4.4.5, we define R_c to be terminating while the two others are non-terminating.

Lemma 5.3.1
Let $S \in SIS$ and $R \in T(S)$. If $R' = R_{d_1} \Uparrow R$ exists, then $R' \in S^{(V_S,T_S)}$.

Proof:
We suppose R' exists. We know the rules R, R_{d_1} and R' looks like:

$$R = \frac{p_1 \dots p_m}{c} \quad R_{d_1} = \frac{X_1.X_2}{X_1} \quad R' = \frac{p_1\sigma \dots p_m\sigma}{X_1\sigma}$$

where $\sigma = mgu(c, X_1.X_2)$. For the message c to be unified with $X_1.X_2$, c must have one of the two following forms: $c = X_3$ or $c = m_1.m_2$. In the case $c = X_3$, since R is a structured rule, there must be an i such that $NAVar(p_i) = \{X_3\}$. Clearly, $|p_i| \geq 1$, thus $|p_i| \geq |c| = 1$, hence contradicting the fact that $R \in T(S)$. So it must be the case that $c = m_1.m_2$ and hence σ is forced to be $\sigma = \{X_1 \leftarrow m_1, X_2 \leftarrow m_2\}$. In the case R is the concatenation rule ($R = R_c$), we have $c = X_3.X_4$ and thus $R' = \frac{X_3 \ X_4}{X_3}$. But in this case, R' is redundant and eliminated in the simplification. We then suppose $R \neq R_c$ and we prove $R' \in S^{(V_S,T_S)}$.

- $\underline{NAVar(\mathcal{C}(R')) \subseteq \cup_{p \in \mathcal{P}(R')} NAVar(p)}$: If $NAVar(m_1) = \emptyset$, then $NAVar(\mathcal{C}(R')) = \emptyset$. Suppose $NAVar(m_1) = \{X_3\}$, then $NAVar(\mathcal{C}(R')) = \{X_3\}$. But we know, since R is a structured rule, that there is an i such that $NAVar(p_i) = \{X_3\}$. By the form of σ, $NAVar(p_i\sigma) = \{X_3\}$.

- $\underline{\text{If } i \neq j, \text{ then } NAVar(p_i\sigma) \cap NAVar(p_j\sigma) = \emptyset}$: Since R is structured, we must have $NAVar(p_i) \cap NAVar(p_j) = \emptyset$ for any $i \neq j$. By the form of σ, we conclude easily that $NAVar(p_i\sigma) \cap NAVar(p_j\sigma) = \emptyset$ for any $i \neq j$ since $NAVar(p_i\sigma) = NAVar(p_i)$.

- $\underline{X_1\sigma \in \mathcal{T}^{(V_S,T_S)}}$: By the form of σ, $X_1\sigma = m_1$. Clearly $|m_1| < |m_1.m_2|$. We know that $m_1.m_2 = c \in \mathcal{T}^{(V_S,T_S)}$. Since m_1 is simple-linear, it is the case that $m_1 \in \mathcal{T}^{(V_S,T_S)}$.

- $\underline{\text{For any } i,\ p_i\sigma \in \mathcal{T}^{(V_S,T_S)}}$: By the form of σ, it is clear that $|p_i\sigma| = |p_i|$. Since $p_i \in \mathcal{T}^{(V_S,T_S)}$, and since $p_i\sigma$ is simple-linear, we have that $p_i\sigma \in \mathcal{T}^{(V_S,T_S)}$.

□

The same result holds if we replace R_{d_1} with R_{d_2}.

Lemma 5.3.2
Let $S \in SIS$, $R \in NT(S)$ and t be a tuple of $T(S)$ rules such that $R_c \in t$. If $R' = R \Uparrow t$ exists, then $R' \in S^{(V_S,T_S)}$.

Proof:
Let $t = (R_1, \ldots, R_c, \ldots, R_m)$. We suppose, without lost of generality, that R_c occurs only once in t. The rules looks like:

$$R = \frac{p_1 \ldots p_m}{c} \quad R_c = \frac{X_1 \ X_2}{X_1.X_2} \quad R' = \frac{\mathcal{P}(R_1)\sigma \ldots X_1\sigma \ X_2\sigma \ldots \mathcal{P}(R_m)\sigma}{c\sigma}$$

Let p_i be the premise of R to be unified with the rule R_c, that is p_i is unified with $X_1.X_2$. For this to be possible, p_i is either X_3 or of the form $m_1.m_2$. In the case $p_i = X_3$, two cases are possible:

- If $NAVar(c) = \{X_3\}$: Since $R \in NT(S)$, it must be the case that $|p_i| \geq |c|$. But to have $|p_i| \geq |c|$ when $p_i = X_3$ and $NAVar(c) = \{X_3\}$, is must be the case that $c = X_3$. In this case, R is redundant and has been removed.

- If $NAVar(c) \neq \{X_3\}$: Then the premises $p_i = X_3$ is useless in R since clearly the intruder can provide a message to fill the variable. Thus p_i has been removed from R[6].

[6]This simplification was discussed in Section 5.3.1

It remains only that $p_i = m_1.m_2$ and here σ_i is forced to be $\sigma_i = \{X_1 \leftarrow m_1, X_2 \leftarrow m_2\}$. We now show that $R' \in S^{(V_S,T_S)}$. Note that we are only interested in the messages $X_1\sigma$, $X_2\sigma$ and $c\sigma_i$, since all other messages of R' can be treated exactly as in lemmas 4.4.5 to 4.4.9 and we know they behave correctly.

- $\underline{NAVar(\mathcal{C}(R')) \subseteq \cup_{p\in\mathcal{P}(R')} NAVar(p)}$: Suppose $NAVar(c\sigma) = \{X_3\}$ which implies $NAVar(c) = \{X_4\}$. Since R is structured, there is a k such that $NAVar(p_k) = \{X_4\}$.
  - If $p_k = p_i$ (p_i is the premise of R unified with $X_1.X_2$) then $NAVar(X_1\sigma) = \{X_4\}$ or $NAVar(X_2\sigma) = \{X_4\}$ (and $X_3 = X_4$).
  - If $p_k \neq p_i$, then since p_k is unified with c_k of rule R_k, one of the p_{kj} will be such that $NAVar(p_{kj}\sigma) = \{X_3\}$ (see Lemma 4.4.6).
- $\underline{\forall p \neq p' \in \mathcal{P}(R'),\ NAVar(p) \cap NAVar(p') = \emptyset}$: The fact that p_i is linear implies $NAVar(X_1\sigma) \cap NAVar(X_2\sigma) = \emptyset$. Since $NAVar(p_i) \cap NAVar(p_j) = \emptyset$ for any j different from i, we will have that $NAVar(X_1\sigma) \cap NAVar(p) = \emptyset$ for any $p \in \mathcal{P}(R_j)\sigma$. The same holds for X_2. For the disjointness between other pairs of premises, the reasoning behind Lemma 4.4.7 can be applied.
- $\underline{c\sigma_i \in \mathcal{T}^{(V_S,T_S)}}$: By the form of σ_i we have $|c\sigma_i| = |c|$. Since c is simple-linear, it is clear that $c\sigma_i$ is also simple-linear. Moreover, since $c \in \mathcal{T}^{(V_S,T_S)}$, we conclude $c\sigma_i \in \mathcal{T}^{(V_S,T_S)}$.
- $\underline{X_1\sigma, X_2\sigma \in \mathcal{T}^{(V_S,T_S)}}$: Since $p_i = m_1.m_2$ is simple-linear, m_1 and m_2 are both simple-linear (and thus $X_1\sigma$ and $X_2\sigma$ are both simple-linear). It is clear that $|m_1| < |p_i|$ and $|m_1| < |p_i|$. Since $p_i \in \mathcal{T}^{(V_S,T_S)}$, we conclude $m_1, m_2 \in \mathcal{T}^{(V_S,T_S)}$ and thus $X_1\sigma, X_2\sigma \in \mathcal{T}^{(V_S,T_S)}$.

□

The next theorem states that the power set of augmented-structured inference systems is closed under the transformation process.

Theorem 5.3.1
Let $S \in SIS$, $S' \in \wp(S_+^{(V_S,T_S)})$. Then $\Phi(S') \in \wp(S_+^{(V_S,T_S)})$.

Proof:
We know by Theorem 4.4.2 that $\wp(S^{(V_S,T_S)})$ is closed under the transformation process. Lemmas 5.3.1 and 5.3.2 provide the remaining arguments to conclude $\wp(S_+^{(V_S,T_S)})$ is closed under the transformation process.

□

From there, we can use the same argument as in Section 4.4.3 to show that the transformation process converges over augmented-structured inference systems. We define a function $\Phi'(\_)$ modeling the transformation over augmented-structured inference systems. The augmented-structured property is preserved by the function as stated in Theorem 5.3.1. It is clear that $\Phi'(\_)$ is an order-preserving map from $\wp(S_+^{(V_S,T_S)})$ to $\wp(S_+^{(V_S,T_S)})$ (see the justification done in Lemma 4.4.11) and since $\wp(S_+^{(V_S,T_S)})$ forms a a finite poset with a top element (the top here is $S_+^{(V_S,T_S)}$), Lemma 4.4.10 assures us that the recursive application of $\Phi'(\_)$ on any augmented-structured inference system reaches a fixpoint and thus that the transformation process converges. The details are not given since this would only be a repetition of the work done in the previous chapter.

5.3.4 Adapting the proof-search procedure termination result

Since the rule R_c is considered as oriented, it can (and will) be part of the resulting inference system. The proof-search procedure presented in Section 4.5.1 must be adapted to avoid a pathetic chance of infinite looping. When trying to prove a term $t = X_3$ where X_3 is a non-atomic variable, the proof-search procedure could use the rule R_c thus generating two new terms $t_1 = X_1, t_2 = X_2$ to prove. But to prove both t_1 and t_2, the proof-search procedure can again use R_c, ... thus looping infinitely. To avoid this problem, we put the following restriction on the use of the rule in the proof-search procedure. If the message to prove is like t (it is simply a non-atomic variable), then t is removed from the list of terms to prove (t is considered as being always provable). This restriction is sufficient to avoid non-termination, and it is correct (it does not change the provability nor the non-provability of any term). The correctness comes from the following reasoning: In a proof tree for the term t', if we are required to prove the term $t = X_3$ (with X_3 a non-atomic variable[7]), it is clear that X_3 can be proved[8], otherwise the theory of the inference system is empty. Thus we can simply remove t from the list of terms to prove, since it is clearly provable and its proof will not affect the other terms to be proved. If we can remove t without trying to prove it, we surely avoid the non-termination problem mentioned above.

It is clear that the messages to be proved after the use of the rule R_c on the term t will be:

[7]Remember that only non-atomic variables of type *Msg* are used.
[8]Remember that variables are existentially quantified

- simple-linear;
- of size smaller than t.

These two properties are sufficient to prove that the proof-search procedure always terminates. The proof is exactly the same as the one done in Section 4.5.3 to prove the termination of the proof-search procedure without the R_c rule.

In the rest of this chapter, especially in Section 5.5, we often talk about decidable security properties, for example we say it is possible to decide if a protocol is confidential. When we do so, we implicitly talk about structured protocols and not protocols in general.

5.4 The class of protocols is wide enough

For our method to be interesting, the class of protocols concerned (namely the class of structured protocols) must be wide. We present here some structured protocols; these protocols are not necessarily built over the algebra we define above, they may include asymmetric keys or multiple keys shared between two principals (both extensions to our algebra were discussed and should cause no problem). In addition to the protocol explicitly mentioned in this section, we can add the three passes protocol of Table 1.20 and the protocol of Table 5.9.

Among the *Woo & Lam* family, we mention that the protocols π_1, π_2 and π_3 presented in [16], the original version (see Table 1.5) as well as its corrected version and the corrected version of π_1 are all structured protocols.

Both the *Needham-Schroeder* and the *Needham-Schroeder-Lowe* public-key protocols (see [31]) are structured.

The *Andrew* secure RPC protocol (see [16]) is structured. However, the message algebra would have to be improved to deal with arithmetic expression. The protocols of tables 5.2 to 5.3 are structured.

$$
\begin{array}{ll}
1 & A \rightarrow S : \{A.B.N_A\}_{K_{AS}} \\
2 & S \rightarrow A : \{A.B.N_A.K_{AB}\}_{K_{AS}} \\
3 & S \rightarrow B : A \\
4 & B \rightarrow S : N_B \\
5 & S \rightarrow B : \{A.B.N_B.K_{AB}\}_{k_{BS}}
\end{array}
$$

Table 5.2: Protocol A

$$
\begin{array}{ll}
1 & A \rightarrow S : A.B \\
2 & S \rightarrow A : \{A.B.K_{AB}\}_{K_{AS}} \\
3 & S \rightarrow B : \{A.B.K_{AB}\}_{K_{BS}}
\end{array}
$$

Table 5.3: Protocol B

Regarding the number of literature protocols we found to be structured (and there is probably many more), we can conclude that the class of protocol that can be decided by our method is wide enough to be interesting.

5.5 Some decidable security properties

We give here some security properties that are decidable with the method presented in Section 5.2. We see how we can decide if a structured protocol respects the secrecy property or if a structured protocol is chaotic. We also discuss why our method cannot be used (in its present form) to decide authentication even if DYMNA can deal with it. However, we will give some possible modifications allowing to decide authentication.

5.5.1 Chaotic property

In this section, we explain how our method can be used to test if a structured protocol is chaotic or not. We also explain how it is possible to build an attack scenario from a proof.

As we discussed in Section 1.3, a protocol is chaotic for a given key K if the intruder is able to both encrypt any message with the key K and decrypt any message encrypted

under K without ever learning the value of K. In [59], Mejri explains how to verify if a protocol is chaotic regarding K using the DYMNA approach. It is required to check if the two constraints $c_1 = \mathcal{K}_I \cup \{m\} \models \{m\}_K$ and $c_2 = \mathcal{K}_I \cup \{\{m\}_K\} \models m$ are provable in their respective inference system. Constraint c_1 states that the intruder can encrypt any given message m with the key K while c_2 states the intruder can decrypt any message of the form $\{m\}_K$. We will see how to decide if a structured protocol is chaotic through the example of the *Woo & Lam* protocol (see Table 1.5). In Section 2.2.1 we computed the inference system modeling the *Woo & Lam* protocol (see tables 2.1 and 2.2).

Lemma 5.5.1
In the Woo & Lam protocol, the intruder can encrypt any given message m using the key K_{AS}.

Proof:
We take $\mathcal{K}_I = \{K_{IS}\} \cup \mathit{Principals} \cup \{m\}$ to be the initial knowledge of the intruder. In Table 5.4 we can see the inference system modeling the protocol separated into terminating and non-terminating rules as stated in Definition 4.4.5. We use the transformation algorithm to get an equivalent inference system containing only terminating rules, see Table 5.5 (the complete transformation steps are given in Appendix B). From the resulting inference system, we apply the proof-search procedure to check if $\{m\}_{K_{AS}}$ is a theorem. The derivation

$$\frac{\frac{\square}{m} R_3}{\{m\}_{K_{AS}}} \sigma(R_4)$$

with $\sigma = \{X_1 \leftarrow m, P_2 \leftarrow A\}$, provides an easy proof. We conclude, since there is a proof for $\{m\}_{K_{AS}}$, that the intruder can encrypt any given message under the key K_{AS}.

□

Lemma 5.5.2
In the Woo & Lam protocol, the intruder can decrypt any message $\{m\}_{K_{AS}}$ previously encrypted under the key K_{AS}.

Proof:
We take $\mathcal{K}_I = \{K_{IS}\} \cup \mathit{Principals} \cup \{\{m\}_{K_{AS}}\}$ to be the initial knowledge of the intruder. The inference system modeling the protocol is the same as the one in Table 5.4 except that rule R_3 is replaced by

$$\frac{\square}{\{m\}_{K_{AS}}}$$

$T(S)$	$NT(S)$
$R_1 = \frac{\square}{P_1} \quad R_2 = \frac{\square}{K_{IS}}$	$R_8 = \frac{X_6.X_7}{X_6} \quad R_9 = \frac{X_8.X_9}{X_9}$
$R_3 = \frac{\square}{m} \quad R_4 = \frac{X_1}{\{X_1\}_{K_{P_2S}}}$	$R_{11} = \frac{\{X_{10}\}_{K_{P_9P_{10}}} \quad K_{P_9P_{10}}}{X_{10}}$
$R_5 = \frac{P_3 \quad X_2}{\{P_3.X_2\}_{K_{P_4S}}} \quad R_6 = \frac{X_3 \quad X_4}{X_3.X_4}$	$R_{12} = \frac{\{P_{11}.\{X_{11}\}_{K_{P_{11}S}}\}_{K_{P_{12}S}}}{\{X_{11}\}_{K_{P_{12}S}}}$
$R_7 = \frac{X_5 \quad K_{P_5P_6}}{\{X_5\}_{K_{P_5P_6}}} \quad R_{10} = \frac{P_7}{N^{\alpha}_{P_8}}$	$R_{19} = \frac{p_{14}.\{X_{12}\}_{K_{p_{14}S}}}{\{X_{12}\}_{K_{p_{15}S}}}$
	$R_{20} = \frac{p_{16} \quad \{X_{13}\}_{K_{p_{16}S}}}{\{X_{13}\}_{K_{p_{17}S}}}$

Table 5.4: *Woo* & *Lam* initial inference system

$T(S')$		
$R_1 = \frac{\square}{P_1}$	$R_2 = \frac{\square}{K_{IS}}$	$R_3 = \frac{\square}{m}$
$R_4 = \frac{X_1}{\{X_1\}_{K_{P_2S}}}$	$R_5 = \frac{P_3 \quad X_2}{\{P_3.X_2\}_{K_{P_4S}}}$	$R_6 = \frac{X_3 \quad X_4}{X_3.X_4}$
$R_7 = \frac{X_5 \quad K_{P_5P_6}}{\{X_5\}_{K_{P_5P_6}}}$	$R_{10} = \frac{p_7}{N^{\alpha}_{P_8}}$	

Table 5.5: *Woo* & *Lam* resulting inference system 1

Using the transformation process, as detailed in Appendix C, we get an equivalent inference system containing only terminating rules, see Table 5.6. From the resulting inference system, we apply the proof-search procedure to check if m is derivable. Axiom R_{27} provides a direct proof for m and we conclude the intruder can decrypt any message previously encrypted under K_{AS}.

□

$T(S')$		
$R_1 = \frac{\square}{P_1}$	$R_2 = \frac{\square}{K_{IS}}$	$R_4 = \frac{X_1}{\{X_1\}_{K_{P_2S}}}$
$R_5 = \frac{P_3 \quad X_2}{\{P_3.X_2\}_{K_{P_4S}}}$	$R_6 = \frac{X_3 \quad X_4}{X_3.X_4}$	$R_7 = \frac{X_5 \quad K_{P_5P_6}}{\{X_5\}_{K_{P_5P_6}}}$
$R_{10} = \frac{P_7}{N^{\alpha}_{P_8}}$	$R_{26} = \frac{\square}{\{m\}_{K_{P_{18}S}}}$	$R_{27} = \frac{\square}{m}$

Table 5.6: *Woo & Lam* resulting inference system 2

Let's now see how we can build the attack associated with each lemma, that is the attack scenario allowing the intruder to encrypt a message m with K_{AS} and the one allowing him to decrypt any message of the form $\{m\}_{K_{AS}}$.

For Lemma 5.5.1 (the intruder can encrypt any message m with K_{AS}), the proof only use R_4 to form $\{m\}_{K_{AS}}$ from m. This rule was extracted from $\mathcal{GR}(A)$. The attack associated with this rule (as shown in Table 2.1) is detailed in Table 5.7.

$$
\begin{array}{ll}
1 & A \rightarrow I(B) : A \\
2 & I(B) \rightarrow A : m \\
3 & A \rightarrow I(B) : \{m\}_{k_{AS}}
\end{array}
$$

Table 5.7: Attack scenario

For Lemma 5.5.2 (the intruder can decrypt any message of the form $\{m\}_{K_{AS}}$), the proof is provided by rule R_{27}. But since R_{27} is not present in the initial inference system, we cannot directly extract an attack from this rule. What we do is decompose R_{27} using the compositions we did in Appendix C. In this way, we get:

$$
\begin{aligned}
& R_{27} \\
= \\
& R_{11} \Uparrow (R_{26}, R_2) \\
= \\
& R_{11} \Uparrow ([R_{20} \Uparrow (R_1, R_3)]R_2) \\
= \\
& R_{11} \Uparrow ([[R_{12} \Uparrow R_5] \Uparrow (R_1, R_3)]R_2)
\end{aligned}
$$

Now we know that the proof tree $\frac{\square}{m}R_{27}$ can also be represented by:

$$\dfrac{\dfrac{\dfrac{\frac{\square}{A}R_1 \quad \frac{\square}{\{m\}_{K_{AS}}}R_3}{\{A.\{m\}_{K_{AS}}\}_{K_{IS}}}R_5}{\{m\}_{K_{IS}}}R_{12} \qquad \frac{\square}{K_{IS}}R_2}{m}R_{11}$$

This proof tree uses only rules in the initial inference system. Among the rules used in the proof tree, only R_5 and R_{12} are rules extract from the protocol, the others are intruder abilities or knowledge. The attack we can mount is shown in Table 5.8.

1.1	$I(A) \rightarrow I : A$
1.2	$I \rightarrow I(A) : N_I^1$
1.3	$I(A) \rightarrow I : \{m\}_{k_{AS}}$
1.4	$I \rightarrow I(S) : \{A.\{m\}_{k_{AS}}\}_{K_{IS}}$
2.4	$I \rightarrow S : \{A.\{m\}_{k_{AS}}\}_{K_{IS}}$
2.5	$S \rightarrow I : \{m\}_{k_{IS}}\}$

Table 5.8: Attack scenario

Although the transformation process was not designed for this purpose, it could be easily adapted to allow **automatic** construction of attack scenario from a proof. It is only necessary to keep, for any rule R, how it has been generated ($R = R' \Uparrow R''$). This additional information allows us to build an attack scenario from a proof. However, the resulting attack is not always the most elegant, as we can see with the attack of Table 5.8.

5.5.2 Secrecy property

Through the next example, we illustrate two interesting features of our method. On one hand, we show how we can decide confidentiality and on the other hand, we show that we can easily model the intruder as being either a legitimate agent of the system or an exterior entity. The example is based on the protocol of Table 5.9. In this protocol, we use m_A for a constant of type *Msg* generated by agent A. We will first check that the message m_A is confidential when the intruder is not a legitimate agent of the system, then we will show that it is no longer the case when the intruder is a legitimate agent (when he posses a key K_{IS}).

$$\begin{array}{ll} 1 & A \rightarrow B : A.\{m_A\}_{K_{AS}} \\ 2 & B \rightarrow S : B.\{A.\{m_A\}_{K_{AS}}\}_{K_{BS}} \\ 3 & S \rightarrow B : \{A.m_A\}_{k_{BS}} \end{array}$$

Table 5.9: Protocol C

First, we model the protocol of Table 5.9 to reflect the fact that the intruder is not part of the system. This gives the initial inference system S of Table 5.10 where we can see the intruder does not know the key K_{IS} and where the intruder identifier I is not present in the set *Principals*.

$T(S)$	$NT(S)$
$R_1 = \frac{\square}{P_1}$ $R_2 = \frac{X_1 \;\; X_2}{X_1.X_2}$	$R_6 = \frac{X_5.X_6}{X_5}$ $R_7 = \frac{X_7.X_8}{X_8}$
$R_3 = \frac{X_3 \;\; K_{P_2P_3}}{\{X_3\}_{K_{P_2P_3}}}$	$R_8 = \frac{\{X_9\}_{K_{P_7P_8}} \;\; K_{P_7P_8}}{X_9}$
$R_4 = \frac{\square}{P_4.\{m_{P_4}\}_{K_{P_4S}}}$	$R_9 = \frac{P_{10}.\{P_9.\{X_{10}\}_{K_{P_9S}}\}_{K_{P_{10}S}}}{\{P_9.X_{10}\}_{K_{P_{10}S}}}$
$R_5 = \frac{P_5.X_4}{P_6.\{P_5.X_4\}_{K_{P_6S}}}$	

Table 5.10: Initial inference system S for the protocol C

When applying the transformation process to the initial inference system S of Table 5.10, we get the resulting inference system $T(S')$ presented in Table 5.11. This resulting

inference system is equivalent to the initial one but is composed only of terminating rules (in the sense of Definition 4.4.5). From this resulting inference system, it is easy to see that $m_A \notin TH(T(S'))$ which imply $m_A \notin TH(S)$. We conclude that the intruder is not able to learn the message m_A, thus m_A is confidential, when the intruder is not a legitimate agent of the system.

$T(S')$		
$R_1 = \frac{\square}{P_1}$	$R_2 = \frac{X_1 \quad X_2}{X_1.X_2}$	$R_3 = \frac{X_3 \quad K_{P_2P_3}}{\{X_3\}_{K_{P_2P_3}}}$
$R_4 = \frac{\square}{P_4.\{m_{P_4}\}_{K_{P_4S}}}$	$R_5 = \frac{P_5.X_4}{P_6.\{P_5.X_4\}_{K_{P_6S}}}$	$R_{14} = \frac{\square}{\{m_{P_{11}}\}_{K_{P_{11}S}}}$
$R_{15} = \frac{P_{12}.X_{11}}{\{P_{12}.X_{11}\}_{K_{P_{13}S}}}$	$R_{17} = \frac{P_{16} \quad \{X_{13}\}_{K_{P_{15}S}}}{\{P_{16}.X_{13}\}_{K_{P_{16}S}}}$	$R_{18} = \frac{\square}{\{P_{17}.m_{P_{17}}\}_{K_{P_{18}S}}}$
$R_{19} = \frac{P_{19}.X_{14}}{\{P_{20}.P_{19}.X_{14}\}_{K_{P_{21}S}}}$		

Table 5.11: Resulting inference system S' for the protocol C

Now, we model the protocol of Table 5.9 to reflect the fact that the intruder is part of the system. This gives the initial inference system S of Table 5.12 where we can see the intruder does know the key K_{IS}.

$T(S)$	$NT(S)$
$R_1 = \frac{\square}{P_1} \quad R_2 = \frac{\square}{K_{IS}}$	$R_7 = \frac{X_5.X_6}{X_5} \quad R_8 = \frac{X_7.X_8}{X_8}$
$R_3 = \frac{X_1 \quad X_2}{X_1.X_2} \quad R_4 = \frac{X_3 \quad K_{P_2P_3}}{\{X_3\}_{K_{P_2P_3}}}$	$R_9 = \frac{\{X_9\}_{K_{P_7P_8}} \quad K_{P_7P_8}}{X_9}$
$R_5 = \frac{\square}{P_4.\{m_{P_4}\}_{K_{P_4S}}}$	$R_{10} = \frac{P_{10}.\{P_9.\{X_{10}\}_{K_{P_9S}}\}_{K_{P_{10}S}}}{\{P_9.X_{10}\}_{K_{P_{10}S}}}$
$R_6 = \frac{P_5.X_4}{P_6.\{P_5.X_4\}_{K_{P_6S}}}$	

Table 5.12: Initial inference system S for the protocol C

When applying the transformation process to the initial inference system S of Table 5.12, we get the resulting inference system $T(S')$ presented in Table 5.13. This resulting inference system is equivalent to the initial one but is composed only of terminating rules (in the sense of Definition 4.4.5). From this resulting inference system we can see that $m_A \in TH(T(S'))$, by the rule R_{27}, which imply $m_A \in TH(S)$ by equivalence. In fact, the attack presented in Table 5.14 can be mounted by the intruder to learn the message m_A.

$T(S')$		
$R_1 = \frac{\square}{P_1}$	$R_2 = \frac{\square}{K_{IS}}$	$R_3 = \frac{X_1 \;\; X_2}{X_1.X_2}$
$R_4 = \frac{X_3 \;\; K_{P_2P_3}}{\{X_3\}_{K_{P_2P_3}}}$	$R_5 = \frac{\square}{P_4.\{m_{P_4}\}_{K_{P_4S}}}$	$R_6 = \frac{P_5.X_4}{P_6.\{P_5.X_4\}_{K_{P_6S}}}$
$R_{15} = \frac{\square}{\{m_{P_{11}}\}_{K_{P_{11}S}}}$	$R_{16} = \frac{P_{12}.X_{11}}{\{P_{12}.X_{11}\}_{K_{P_{13}S}}}$	$R_{21} = \frac{P_{16} \;\; \{X_{13}\}_{K_{P_{16}S}}}{\{P_{16}.X_{13}\}_{K_{P_{17}S}}}$
$R_{22} = \frac{\square}{\{P_{18}.m_{P_{18}}\}_{K_{P_{19}S}}}$	$R_{23} = \frac{P_{20}.X_{14}}{\{P_{21}.P_{20}.X_{14}\}_{K_{P_{22}S}}}$	$R_{25} = \frac{\square}{P_{24}.m_{P_{24}}}$
$R_{26} = \frac{P_{25}.X_{16}}{P_{26}.P_{25}.X_{16}}$	$R_{29} = \frac{\square}{m_{P_{27}}}$	$R_{31} = \frac{X_{17} \;\; K_{P_{28}S}}{P_{28}.X_{17}}$
$R_{34} = \frac{P_{29} \;\; \{X_{18}\}_{K_{P_{29}S}}}{P_{30}.P_{29}.X_{18}}$	$R_{35} = \frac{\square}{P_{31}.P_{32}.m_{P_{32}}}$	$R_{36} = \frac{P_{33}.X_{19}}{P_{35}.P_{34}.P_{33}.X_{19}}$
$R_{46} = \frac{\{X_{21}\}_{K_{P_{38}S}} \;\; K_{P_{38}S}}{\{P_{38}.X_{21}\}_{K_{P_{39}S}}}$	$R_{47} = \frac{\{X_{22}\}_{K_{P_{40}S}} \;\; K_{P_{40}S}}{P_{40}.X_{22}}$	$R_{48} = \frac{\{X_{23}\}_{K_{P_{42}S}} \;\; K_{P_{42}S}}{P_{43}.P_{42}.X_{23}}$

Table 5.13: Resulting inference system S' for the protocol C

$$
\begin{array}{ll}
1 & A \rightarrow I(B) : A.\{m_A\}_{K_{AS}} \\
2 & I \rightarrow S : I.\{A.\{m_A\}_{K_{AS}}\}_{K_{IS}} \\
3 & S \rightarrow I : \{A.m_A\}_{k_{IS}}
\end{array}
$$

Table 5.14: Attack scenario

5.5.3 Authentication property

Unfortunately, we cannot use the method presented in Section 5.2 to decide if a structured protocol respects the authentication property. In this section, we first discuss the problem of authentication toward our method and we propose some simple modifications to our method to achieve the decidability of authentication.

Authentication is not yet decidable

The reason why authentication is not currently decidable in our model is that the transformation process (on inference system) only preserves the theory between inference systems. But if there are many proofs for a term in S, there is possibly only one proof for the same term in S', even with $S \equiv S'$. This equivalence does not affect the confidentiality nor the chaotic property since they only require the existence of a proof. Unfortunately, this is not true for the authentication property. The constraint generated for the authentication property are always provable in the inference system modeling the protocol (using the forwarding attack); but to conclude the protocol does not respect authentication, we must find a proof that yields an attack different from the forwarding attack.

In the example of Section 2.2.1, we saw that the authentication property constraints are extracted from the generalized role to attack. For the one-way authentication over B in the *Woo & Lam* protocol, we saw that the constraints are: $\{\mathcal{K}_I \vdash A, \mathcal{K}^1 \vdash Y, \mathcal{K}^2 \vdash \{N_b^\beta\}_{k_{bs}}\}$, where $\mathcal{K}^1 = \mathcal{K}_I \cup \{N_b^\beta\}$ and $\mathcal{K}^2 = \mathcal{K}^1 \cup \{\{A.Y\}_{k_{bs}}\}$. These constraints are clearly provable in the underlying inference system since we can mount the attack of Table 5.15 (this is the forwarding attack). Clearly, this attack does not yield an authentication flaw in the protocol.

$$
\begin{array}{ll}
1.1 & A \rightarrow I(B) : A \\
2.1 & I(A) \rightarrow B : A \\
2.2 & B \rightarrow I(A) : N_b^2 \\
1.2 & I(B) \rightarrow A : N_b^2 \\
1.3 & A \rightarrow I(B) : \{N_b^2\}_{K_{as}} \\
2.3 & I(A) \rightarrow B : \{N_b^2\}_{K_{as}} \\
2.4 & B \rightarrow I(S) : \{A.\{N_b^2\}_{K_{as}}\}_{K_{bs}} \\
3.4 & I(B) \rightarrow S : \{A.\{N_b^2\}_{K_{as}}\}_{K_{bs}} \\
3.5 & S \rightarrow I(B) : \{N_b^2\}_{K_{bs}} \\
1.5 & I(S) \rightarrow B : \{N_b^2\}_{K_{bs}}
\end{array}
$$

Table 5.15: Forwarding attack

Thus to affirm a protocol does not respects the authentication property, we must find a proof in the underlying system that mounts an attack different from the forwarding attack. It is not necessarily possible to do so in a resulting inference system since we may have lost some proofs and we may end up with only the forwarding attack remaining. However, we can modify the transformation process in a simple way to avoid this problem and enable the decidability of authentication.

Modifying the transformation process

The main problem with the transformation process comes from the equivalence relation between inference systems (two inference system are equivalent if they have the same theory). For the authentication, we need a somewhat stronger[9] equivalence (two inference system are strongly equivalent if they have the same theory and for every terms in their theory, these terms have the same set of proof trees). We do not give a formal definition of this strong equivalence but simply the intuition we have for the authentication property.

We must be able to show that the transformation process preserves the strong equivalence of inference systems, that is $S_1 \cup S_2 \equiv_s (S_2 \looparrowright S_1)_{\downarrow}$. To achieve this, we must work again on the definition of a simplified inference system and we need to change the structure of a rule. A rule R is now composed of two parts, the first part is exactly what a rule was before (some premises and a conclusion), the second part is a set of path we refer to it by $Path(R)$. The set of paths corresponds to every possible path that can be used to build a rule. The rules extracted from the protocols have themselves as path. Now when we simplify an inference system and we eliminate a rule R because it is a renaming of another rule R', we update the path of R' to be $Path(R) \cup Path(R')$. Thus when removing R (because R is a renaming of a rule R'), we still allow the inference system to use R (in all its different form) through R'. When we build a new rule $R' = R \Uparrow (R_1, \dots R_n)$, we set $Path(R')$ to be $R \Uparrow (R_1, \dots R_n)$.

From there, we conjecture that the transformation process preserves not-only the theory of its initial inference system, but also the complete set of proofs for any term. That is, $S = S_1 \cup S_2 \equiv_s (S_2 \looparrowright S_1)_{\downarrow} = S'$. The proof of such a conjecture should be done in the same way as the proof of Lemma 5.4.1 in [57]. However, it would be tedious and would require many new formal definitions (path, proof tree,...).

It is clear that these modifications do not change the convergence in the case of a structured inference system. However, the proof-search procedure needs to be redone. It is not sufficient anymore to get simply the yes/no answer to whether a term is

[9] We use $\equiv_s$ to denote this strong equivalence

provable in the resulting inference system. We now need to get the complete set of proof associated with a term, and from this set of proofs, we can check if there is at least one which does not correspond to the forwarding attack. Since we showed in Section 4.5.2 that the tree to search in for proofs in finite, there is clearly a terminating procedure to enumerate all possible proofs for a given term.

With these modifications, we can decide if a structured inference system respects the authentication or not.

5.6 Into associative unification

Since concatenation is naturally considered to be an associative binary function, we believe the algebra would be a lot more interesting if the function C would be associative. We discuss here the impact of having an associative function in our algebra. As we will see, treating the associativity will create some new problems. These problems are very difficult, almost as much as the problem of termination of inference systems itself.

This section should be seen as an insight to some ongoing and even future work. We do not aim to give interesting results here but simply expose the problems we face and give some ideas and intuitions about how these problems could be solved.

5.6.1 Associative unification

A-unification is unification modulo the associativity theory. Two terms t_1 and t_2 are A-unifiable if there exists a substitution σ such that $t_1\sigma =_A t_2\sigma$. That is, $t_1\sigma$ and $t_2\sigma$ are either syntactically equal (as it is the case with two $\emptyset$-unifiable terms) or $t_1\sigma$ can be rewritten has $t_2\sigma$ simply by applying the associativity law. It is often the case that when considering A-unification, only one function is taken to be associative. We make this assumption in this whole section. We do not survey the field of A-unification, however, information on this can be found in [4, 34]. A-unification is a special case of the general E-unification (where E describe an equational theory), see [5, 26, 40, 46, 64, 65].

Decidability of A-unification was first conjectured by *Plotkin* in [69] and then proven by *Makanin* in [50]. Thus given any two terms, we can decide if they are A-unifiable.

5.6.2 A-unification is not unitary

While it is the case that two $\emptyset$-unifiable terms have a unique *mgu*, two A-unifiable terms may have many "*mgu*". Since it is contradictory to talk about "many" "most general unifier", authors usually adopt the more correct expression complete-minimal set of A-unifiers. The complete-minimal set of A-unifiers of two terms t_1 and t_2 is defined to be the least set such that if σ is a A-unifier of t_1 and t_2, then either σ is in the complete-minimal set of A-unifiers or there is a substitution θ in the complete-minimal set of A-unifiers such that $\sigma = \theta \circ \theta'$ for a substitution θ'. We still use *mgu* here to denote the complete-minimal set of A-unifiers. Not being unitary is not necessarily a problem, but we know from [78] that A-unification is infinitary[10]. The fact that A-unification is infinitary is a problem for the convergence of the transformation process. If an inference system contains the two rules $R_1 = \frac{p_1}{c_1}$ and $R_2 = \frac{p_2}{c_2}$ such that R_1 is terminating and R_2 is not. Suppose $c_1 = C(a, X)$ and $p_2 = C(X, a)$, since c_1 and p_2 are A-unifiable and they have infinitely many *mgu* (as shown in Example 5.6.1), then the composition of R_2 with R_1 would yield possibly infinitely many new rules and thus lead non-termination.

Example 5.6.1 (Infinitely many A-*mgu*)
Let $t_1 = C(a, X)$ and $t_2 = C(X, a)$. Then t_1 and t_2 have infinitely many A-*mgu* (here C is an associative function). They are the following:

- $\{X \leftarrow a\}$
- $\{X \leftarrow C(a, a)\}$
- $\{X \leftarrow C(C(a, a), a)\}$
- ...

In studying the work of *Clerin-Debart & Enjalbert* (see [17, 28]), we found that their result could possibly solve the infinitary problem of A-unification in the special case of simple-linear terms. In [28] they prove that if two terms respect the *unique prefix property* (UPP for short), then these two terms have a finite number of A-*mgu*. The *unique prefix property* states that every variable must possess a unique prefix in the terms to be A-unified. By observing the behavior of A-unification when applied of UPP terms, we have come to believe that we can generalize this result in the following way. The UPP needs not to hold for all variables, but only for variables that may be replaced by the term $C(X_1, X_2)$ where C is the associative function. With this new

[10] Two A-unifiable terms can have infinitely many incomparable *mgu*.

definition of the UPP, it is clear that every pair of simple-linear respects it since only *Msg* variables(or equivalently non-atomic variables) can be instantiated by $C(X_1, X_2)$ and because a non-atomic variable appears at most once in the two terms (it must then have a unique prefix).

So the first problem of A-unification seems to be solved for simple-linear terms thanks to the result of *Clerin-Debart & Enjalbert*.

5.6.3 Associative unification can increase the size of terms

To assure the convergence of the transformation process, it is not sufficient to have a finite number of A-*mgu*, we also need some properties to hold for A-unification. In Section 4.3, we saw some properties of $\emptyset$-unification, we believe they all still hold for A-unification except property 4.3.5. Property 4.3.5 states that if t is the result of $\emptyset$-unifying t_1 and t_2 and if t contains a non-atomic variable, then $|t| \leq max(|t_1|, |t_2|)$. As we can see in Example 5.6.2, this property does not hold for A-unification.

Example 5.6.2
Let $t_1 = C(C(X_1, a), b)$ and $t_2 = C(C(d, e), X_2)$. They are A-unifiable with the following A-*mgu*:

- $\sigma_1 = \{X_1 \leftarrow C(d, e), X_2 \leftarrow C(a, b)\}$
- $\sigma_2 = \{X_1 \leftarrow C(C(d, e), X_3), X_2 \leftarrow C(C(X_3, a), b)\}$

The substitution σ_1 causes no problem, but σ_2 will cause a problem. $t = t_1\sigma_2 = t_2\sigma_2 = C(C(C(C(d, e), X_3), a), b)$. It is the case that t contains a non-atomic variable, but $|t| = |t_1| - 1 + |t_2| > max(|t_1|, |t_2|)$.

This property is important to assure the convergence, it limits to a finite number the number of rule that can be created by the transformation process.

To be able to reuse the convergence proof done in Chapter 4 in the case of A-unification, we need to further restrict the terms to be A-unified to avoid size explosion. Due to time limitation, we have not placed much effort on finding a wide (and interesting) subclass of simple-linear terms for which A-unification is not problematic. However, we propose a restriction on terms that we believe sufficient to keep convergence when modeling the associativity of concatenation.

Definition 5.6.1 (Simple-linear-right term)
A term built over the simple-linear algebra of cryptographic protocol (considering here C to be associative) is right if the non-atomic variable of this term (if it has one) is the last term of a concatenation

Example 5.6.3
Among the following simple-linear terms, t_1, t_2, t_3, t_6 and t_7 are right:

- $t_1 : \{X_1\}_k$
- $t_2 : \{a.X_2\}_k$
- $t_3 : X_3$
- $t_4 : X_4.a$
- $t_5 : \{X_5.\{a\}_k\}_{k'}$
- $t_6 : \{a.\{c.X_6\}_k.b.\{d.e\}_{k'}\}_{k''}$
- $t_7 : \{a.\{c.f.g\}_k.b.\{X_7\}_{k'}\}_{k''}$

We conjecture that given two simple-linear-right terms t_1 and t_2, $t = t_1\sigma = t_2\sigma$ is such that if $NAVar(t) \neq \emptyset$, then $|t| \leq max(|t_1|, |t_2|)$ where σ is any A-mgu of t_1, t_2.

We came up with this while observing terms as special trees. The father-son relation (vertical lines) expresses the encryption while the brother relation (horizontal lines) represents concatenation. The tree of every simple-linear-right term from Example 5.6.3 is pictured in Figure 5.2 (we omitted the keys used for encryption in the tree representation for the sake of readability).

Given two simple-linear-right A-unifiable terms t and t', let t'' be the result of $t\sigma$ with σ any A-mgu of t, t'. If t has no non-atomic variables, then t' must be a subtree[11] of t and clearly $|t''| = |t|$. If they both have a non-atomic variable and these are on different level of the tree, then either t' is a subtree of t (as it is the case when A-unifying the terms t_2 and t_3 illustrated in Figure 5.2), or t'' will end up having no non-atomic variables (as it is the case when A-unifying the terms t_6 and t_7 illustrated in Figure 5.2). The remaining case is when t and t' both have a non-atomic variable and these are on the same level. In this case, one of t, t' is a subtree of the other (as it is the case with the terms t_1 and t_2 in Figure 5.2).

[11] By subtree, we regard here the tree's structure only, not the content of the nodes.

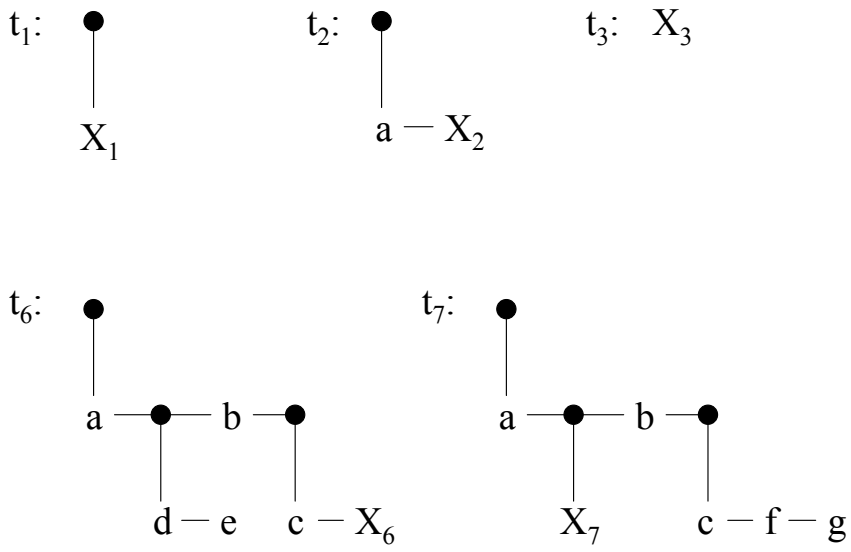

Figure 5.2: Trees for simple-linear-right terms

If the above conjecture holds, we can add the associativity to the concatenation function without losing the convergence at the cost of restricting the class of protocols considered to structured protocol (where structured must be redefined to take simple-linear-right terms only).

5.7 Implementation

This section is dedicated to IST (Inference System Transformation), the Java implementation of the transformation algorithm presented in Section 3.5. IST is available on the web page of the LSFM research group at: www.ift.ulaval.ca/∼lsfm/ or by contacting the author at "francois.gagnon@ift.ulaval.ca".

IST consists mainly of four configuration steps, the transformation engine and the result output. Since the transformation engine is the implementation of the above algorithms, this section rather focuses on the configuration steps and the result output. In addition to the description of IST, we present experimental results of applying IST to some cryptographic protocols. We also present some additional features that could be available in future versions of IST.

5.7.1 The program

We present here the four configuration steps, that is: type configuration, algebra configuration, inference system configuration and time configuration; as well as the result outputting of IST.

Type configuration

The user can choose an untyped algebra or a typed one. In the former case, no additional configuration needs to be done for the type, while in the latter case, a type ordering must be provided. At this point, the user is forced to provide an order such that the types will be structured as a forest (i.e. $\tau_1 \geq \tau \leq \tau_2$ implies either $\tau_1 \geq \tau_2$ or $\tau_1 \leq \tau_2$). This requirement is justified by the unification problem as discussed in Section 4.3. The user can either manually enter all pairs defining the ordering, load the type structure from a text file or load the type structure from a previously saved type structure. If there is a need to distinguish between atomic and non-atomic types (as it is the case in this thesis, see Section 4.2) the user can enter t_i meaning the type named i is atomic or T_i meaning the type named i is non-atomic. Once the user finishes the type configuration, the program checks if the type structure is indeed a forest one. If not, an error message appears and the user must change the structure before taking the next step.

Algebra configuration

Once the type structure, if any, is correctly entered, the user can create his algebra. The algebra configuration panel is pictured in Figure 5.3. The user must choose the kind of algebra considered. It can either be the free algebra (see Definition 4.2.3), the simple-linear algebra (the one used in this thesis, see Definition 4.2.7) or a custom algebra. Selecting the custom algebra requires that an empty function be replaced with the correct Java code and the program be recompiled (see the program documentation for more information). The user can either load a previously saved algebra (note that loading an algebra will automatically load the type setting associated with it) or manually create a new algebra by adding the needed variables and functions. Variable creation, or edition, is done through the interface presented in Figure 5.4. The user must first select a name for its variable and then a type (only in the case typed algebra was selected in the type configuration). The function creation, or edition, is similar to the one for variable except that the arity of the function is asked to the user and in the case of a typed algebra, the type for every parameter as well as for the function output is required. Once the algebra configuration is finished, the user will be asked for the inference system configuration.

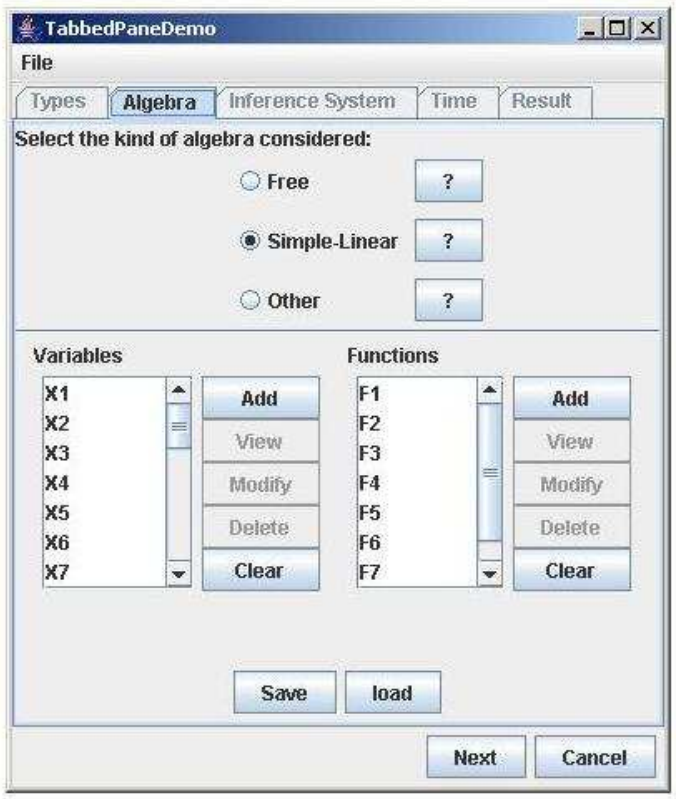

Figure 5.3: Algebra configuration

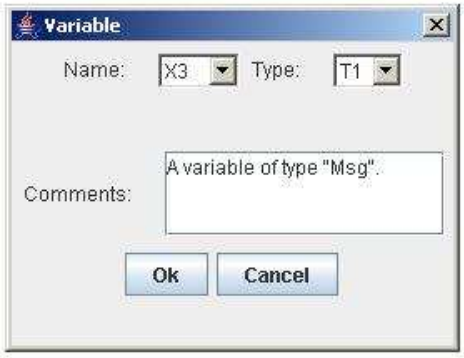

Figure 5.4: Variable configuration

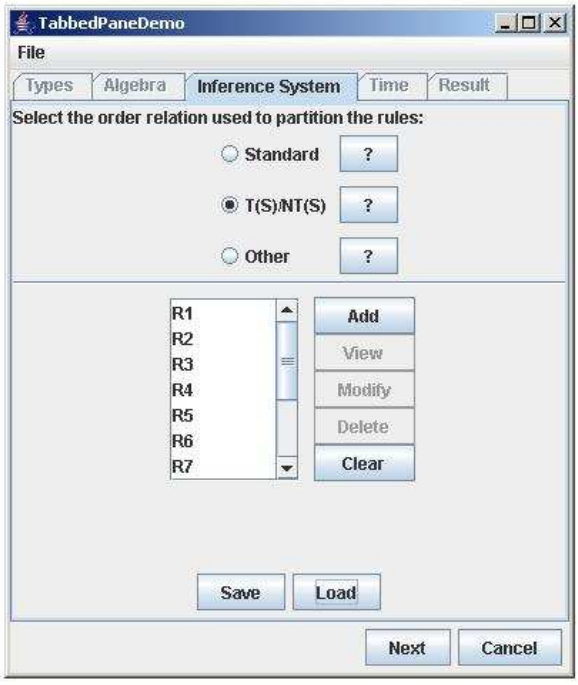

Figure 5.5: Inference system configuration

Inference system configuration

Again here, the user can either manually create an inference system or load a previously saved one (note that loading an inference system will automatically load the algebra for this inference system). The user is required to select how the rule separation is done (which rules are considered to be oriented). The standard separation requires that every premise be smaller than the conclusion for a rule to be oriented, the $T(S)/NT(S)$ separation is defined in 4.4.5, for the custom separation, Java code must be added to an empty function and the program recompiled (see the program documentation for more information). Rules are added with the Rule addition/edition interface pictured in Figure 5.6. The user must specify the name for the rule as well as the number of premises this rule has. Once the number of premises is selected, the corresponding premise fields become editable. The user must properly fill in all editable fields (the premises and the conclusion). By properly, we mean that the terms entered must be well-formed (with respect to the selected algebra kind, the variables and the functions available for this algebra) and well typed (with respect to the type structure and the type for the variables and the functions) if the algebra is typed. Once confirming the inference system configuration is done, the program will split the rules according to the order relation selected.

Figure 5.6: Rule configuration

Time configuration

Before starting the transformation process over the specified inference system, the user must select a time limit for executing the program. It can be either unlimited or limited to a given amount of time. This option is important since termination of the transformation process is not assured in the general case.

Results

When the transformation process stops, either because the convergence is achieved or the time limit is up, the resulting inference system (or the last inference system if the algorithm stopped because of the time limit) is showed to the user.

From the "File" menu, the user can directly load an algebra without configuring the type (loading an algebra will automatically load its associated type structure), or load an inference system without configuring the type and the algebra (loading an inference system will automatically load its associated algebra). It is also possible to have the resulting inference system (even the whole series of inference systems) written in a LaTeX file.

5.7.2 The results

We experimented the IST with some inference systems to check if we can count on a good convergence time. As we can see with our results (see Table 5.16), the time required by the transformation algorithm to produce the resulting inference system is pretty good. The tests were done on a Pentium M 1.5 GHz processor laptop with 512 Mo RAM running Windows XP. Transformation time required is the average time over five repetitions.

Inference system	Termination assured	Transformation time (ms)
Woo & Lam Π (Table 1.5)	yes	174
Appendix B	yes	184
Appendix C	yes	410
Table 5.2	yes	1079
Table 5.3	yes	371
Table 5.9	yes	577
Woo & Lam Π_f [16]	no	N/A
Otway-Rees [16]	no	N/A
Table 5.17	no	160

Table 5.16: Transformation process time result

The inference systems from Appendices B and C were extracted from the *Woo & Lam* Π protocol. We simply added some axioms (intruder knowledge) to check different security properties.

Because of the result exposed in this thesis, we were assured of the termination for the transformation process applied on the first 6 inference systems (the termination was not assured for the three others). The transformation over the *Woo & Lam* Π_f inference system could not be completed[12]; the system ran out of memory after about 23 minutes of computation (while computing S^{10}). The transformation over the *Otway-Rees* inference system never finished[13]; after 15 hours (computing S^{7}[14]) it was still running (no memory problems in this case) but has not reached a convergence point.

[12]Note that it does mean the transformation does not converge but simply that it could not converge with the resources allowed

[13]Note that it does mean the transformation does not converge but simply that it could not converge within the allowed time

[14]Surprisingly, it took only 6 minutes to compute S^6, but S^7 could not be computed in the 15 hours allowed.

$T(S)$	$NT(S)$
$R_2 = \frac{X_2}{F(F(F(X_2)))}$ $R_3 = \frac{\square}{a}$	$R_1 = \frac{F(F(X_1))}{F(X_1)}$

Table 5.17: Inference System

5.7.3 The next version

Additional features for an eventual next version should be:

- A more user friendly and flexible graphical interface ;
- An optimization of the algorithms;
- A plug-in for the special case of modeling cryptographic protocols with inference systems (taking an inference system in the standard notation an automatically creating and transforming the associated inference system). This features should also provide a customizable algorithm for testing if a term belongs to the theory of the resulting inference system;
- Adding a more general treatment to unification thus allowing a less restrictive treatment of the type structure.

5.8 Conclusion

In this chapter, we used the result of the previous chapter: *the theory of any structured inference system is decidable*, to build a decision procedure for a class of cryptographic protocols. We argued that this class is wide since many well-known protocols are in this class. We also gave some examples of security properties that can be decided with our method. Decision procedures, like this one, are quite rare since most of the existing formal methods either aim to prove the correctness of a protocol (see Section 2.3) or to refute its correctness (see Section 2.2).

Conclusion

As a conclusion, we sum up the results we got and contributions we made with this thesis, we compare our results with those of similar approaches for cryptographic protocols analysis and we sketch some interesting future work that would enrich our results.

Contributions

Our contributions to the field of protocol analysis are:

- First, we presented in Chapter 2 a classification of formal methods dedicated to cryptoprotocols analysis. This classification provides good insights about the strengths and weaknesses of these formal methods. It also helps to compare new formal methods with existing ones.
- Then, in Chapter 4, we established a strong result: *The theory of a structured inference system is decidable*. To do this, we presented, among other things, a proof-search procedure for resulting inference systems.
- In Chapter 5 we proposed a decision procedure for a wide class of cryptoprotocols.
- Finally, as told in Section 5.7, we built a Java implementation of the transformation process. This implantation confirms that inference systems are good models for protocols.

We tickled the problem of non-empty unification in the analysis of cryptographic protocols. This problem is by itself, very hard. Although we could not come up with some interesting results, we mentioned some intuitions that may be investigated to get those results. Very few methods, if any, consider the fact that concatenation should be associative (however, not all methods needs to do a special treatment for the associativity of concatenation).

Related work

Blanchet

The worst drawback of *Blanchet*'s approach is the approximation introduced during the idealization process. While we use our method as a decision procedure, it is not possible with *Blanchet*'s. The fact that *Blanchet*'s transformation algorithm does not terminate on the *Needham-Schroeder* secret-key protocol as well as on many of the *Woo & Lam* family protocol, while our's terminate assure that *Blanchet*'s result cannot be more general than our's. Both approaches can treat the fact that old keys have been compromised, however, it seems easier to do so in our's. *Blanchet* uses tuples, instead of a binary function, to represent concatenation. This causes the problem that $x.m_1$ cannot be unified with $m_2.m_3.m_1$ when regarding concatenation as tuples while it is unifiable when concatenation is a binary function. On the other hand, $X.m_1.m_2.Y$ can be unified with $X'.m_3.m_1.Y'$ in tuples while it cannot in binary functions (although it would in associative binary function). The main advantage of *Blanchet* is that no restriction is imposed on the keys used to encrypt messages while we restrict keys to be independent of any other messages (we have a special type of messages only for keys).

Kindreed & Wing

Since the approach of *Kindreed & Wing* is based on existing logic modeling protocols, like the BAN logic, it may suffer from the idealization problem. It is known, see Section 2.5.1, that transforming a protocol in the BAN logic is error prone. Moreover, in [43], the authors had to add some extension to the BAN logic to achieve the desired goals. They say: "We added most of the extension above after some verification failed". Thus, this approach suffers from the possible incompleteness of the logics. In their restrictions imposed on rules, they need every variable in the premise of a G-rule to be in its conclusion. We do not have this restriction. In [45], they show that an inference system composed of $R_1 = \frac{g(f(X))}{g(X)}$ and $R_2 = \frac{g(X)}{g(f(X))}$ cannot be treated by their algorithm (it may cause non-termination), while it causes no problem with our algorithm. Thus it is clear that their method cannot be strictly more general than our's. However, we are aware that they can consider a lot more of security properties since their language (the logics themselves) is far more expressive than our's (simple-linear terms).

Future work

The principal thing to enhance the results presented here would be a formal and deeper consideration of A-unification. It would be interesting to define the biggest subset of structured inference systems that remains decidable when considering associativity. Although this problem is probably as hard, if not more, then the ones solved here, it would greatly enrich our results. A formal proof that authentication can be decided (to replace the informal discussion presented in section 5.5.3) is surely an interesting next step. The enumeration of some other security properties that can be decided with our procedure would also be interesting. Of course, the elaboration of other, different, classes of inference systems for which we can have the convergence of the transformation process would be of a good practical interest, namely in the field of protocols analysis. Surely, the implementation of the transformation process should be embedded in a more specialized program for protocol analysis. This program should take as input a protocol, in the standard notation, together with a security property, and produce as output the yes/no answer to the question "does the protocol respect the security property?". Such a program would be terminating on, at least, structured inference systems.

Bibliography

[1] M. Abadi and A.D. Gordon. Calculus for cryptographci protocols: The spi-calculus. *In proceedings of the 4th ACM Conference on Computer and Communications Security*, pages 36–47, January 1997. Link.

[2] L. Aceto and K.G. Larsen. An introduction to milner's ccs. Link, November 2004.

[3] K. Adi. *Formal Specification and Analyses of Security Protocols*. PhD thesis, Université Laval, May 2002.

[4] F. Baader and K.U. Schulz. General A- and AX-unification via optimized combination procedures. *in proccedings of the second International Workshop on Word Equation and Related Topics (IWWERT'91) - LNCS*, 677:23–42, October 1991.

[5] F. Baader and K.U. Schulz. Unification in the union of disjoint equational theories: Combining decision procedures. *Journal of Symbolic Computation*, 21(2):211–243, 1996.

[6] F. Baader and W. Snyder. *Handbook of Automated Reasoning (Chapter 8: Unification Theory)*. Elsevier Science, 2001.

[7] C. Bennet, G. Brassard, and A. Ekert. Quantum cryptography. *Scientific American*, 267(4):50–57, 1992.

[8] C. Bennet, G. Brassard, and A. Ekert. La cryptographie quantique. *Dossier pour la science, Hors-série*, pages 114–118, October 2002.

[9] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. *In proceedings of the 14th Computer Security Foundation Workshop (CSFW'01)*, pages 82–97, June 2001. Link.

[10] B. Blanchet and A. Podelski. Verification of cryptographic protocols: Tagging enforces termination. *In proceedings of the 2003 Foundations of Software Science and Computation Structures (FoSSaCS'03) - LNCS*, 2620:136–152, April 2003. Link.

[11] D. Bouwmeester, A. Ekert, and A. Zeilinger. *The Physics of Quantum Information*. Springer, 2000.

[12] A.R. Bull, W.S. Choi, C.E. Landweher, and J.P. McDermott. A taxonomy of computer program security flaws, with examples. Technical Report 9591, Navel Research Laboratory, November 1993. Link.

[13] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. Technical Report 39, DEC SRC, Link, 1989.

[14] U. Carlsen. Cryptographic protocols flaws. *In proceedigs of the IEEE Computer Security Foundations Workshop VII*, pages 192–220, June 1994.

[15] I. Cervesato, N.A. Durgin, P.D. Lincoln, J.C. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. *In proceedings of the 12th IEEE Computer Security Foundations Workshop (CSFW'99)*, 1999.

[16] J. Clark and J. Jacob. A survey of authentication protocol literature. Link, November 1997.

[17] F. Clerin-Debart. *Théorie équationnelles et de contraintes pour la déduction automatique en logique modale*. PhD thesis, Laboratoire d'informatique de l'université de Caen, Janvier 1992.

[18] M. Debbabi, M. Mejri, N. Tawbi, and I. Yahmadi. Formal automatic verification of authentication cryptographic protocols. *In proceedings of the 1st International Conference on Formal Engineering Methods (ICFEM'97)*, pages 50–59, November 1997.

[19] M. Debbabi, M. Mejri, N. Tawbi, and I. Yahmadi. From protocol specifications to flaws and attack scenarios: An automatic and formal algorithm. *In proceedings of the 6th Workshop on Enabling Technologies Infrastructure for Collaborative Enterprises (WET-ICE'97)*, pages 256–262, June 1997.

[20] M. Debbabi, M. Mejri, N. Tawbi, and I. Yahmadi. A new algorithm for the automatic verification of authentication protocols: From specifications to flaws and attack scenarios. *In proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, September 1997. Link.

[21] D.E. Denning. *Cryptography and Data Security*. ADDISON-WESLEY, January 1983.

[22] N. Dershowitz. Termination of rewriting. *J. Symbolic Computation*, 3:69–116, 1987.

[23] N. Dershowitz. Natural termination. *Theoretical Computer Science*, 142:179–207, 1995.

[24] W. Diffie and M. Hellman. New directions in cryptography. *in proceedings of the IEEE Transactions on Information Theory*, 22(6):644–654, November 1976. Link.

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

[26] D.J. Dougherty and P. Johann. An improved general E-unification method. *International Conference on Automated Deduction (CADE'10) - LNCS*, 449:261–276, July 1990.

[27] N.A. Durgin, P.D. Lincoln, J.C. Mitchell, and A. Scedrow. Undecidability of bounded security protocols. *In proccedings of the Workshop on Formal Methods and Security Protocols (FMSP'99)*, 1999.

[28] P. Enjalbert and F. Clerin-Debart. A case of terminaison for associative unification. *In proccedings of the second International Workshop on Word Equation and Related Topics (IWWERT'91) - LNCS*, 677:79–89, October 1991.

[29] S. Even and O. Goldreich. On the security of multi-party ping-pong protocols. Technical Report 285, Computer Science Departement, Technion, Haifa 32000, Israel, June 1983. Link.

[30] F.J. Thayer Fábrega, J.C. Herzog, and J.D. Guttman. Honest ideals on strand spaces. *In proceedings of the 1998 Computer Security Foundations Workshop*, 1998.

[31] F.J. Thayer Fábrega, J.C. Herzog, and J.D. Guttman. Stand spaces: Why is a security protocol correct? *1998 IEEE Symposium on Security and Privacy*, 1998.

[32] F.J. Thayer Fábrega, J.C. Herzog, and J.D. Guttman. Strand space pictures. *In proceedings of the 1998 Workshop on Formal Methods and Security Protocols*, 1998.

[33] F.J. Thayer Fábrega, J.C. Herzog, and J.D. Guttman. Strand space: Proving security protocols correct. *Journal of Computer Security*, 7:191–230, 1999. Link.

[34] F. Fages. Associative-commutative unification. *LNCS*, 170:194–209, 1984.

[35] J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. *In proceedings of the 13th Computer Security Foundations Workshop*, pages 255–268, 2000. Link.

[36] N. Heintze and J.D. Tygar. A model for secure protocols and their compositions. *IEEE Transaction on Software Engineering*, 22(1):16–30, January 1996. Link.

[37] C.A.R. Hoare. *Communication Sequential Processes*. Prentice Hall International, 1985. Link.

[38] H. Houmani. Vers la correction des protocoles cryptographiques. Master's thesis, Université Laval, July 2003.

[39] H. Houmani and M. Mejri. Secure protocols for secrecy. In *Foundations of Computer Security: proceedings of the LICS'03 workshop on Foundations of Computer Security*, pages 59–68, Ottawa, Canada, 26–27 June 2003.

[40] J. Jaffar. Minimal and complete word unification. *Journal of the ACM*, 37(1):47–87, January 1990.

[41] J. Jürjens. Composability of secrecy. *International Workshop on Mathematical Methods, Models and Architectures for Computer Networks Security (MMMACNS'2001) - LNCS*, 2052:28–38, 2001. Link.

[42] D. Kindred. *Theory Generation for Security Protocols*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1999. Link.

[43] D. Kindred and J.M. Wing. Fast, automatic checking of security protocols. *In proccedings of the 2nd USENIX Workshop on Electronic Commerce*, pages 41–52, November 1996. Link.

[44] D. Kindred and J.M. Wing. Closing the idealization gap with theory generation. *In proceedings of the DIMACS Workshop on Cryptogaphic Protocol Design and Verification*, September 1997. Link.

[45] D. Kindred and J.M. Wing. Theory generation for security protocols, July 1999. Link.

[46] C. Kirchner. A new equational unification method: a generalisation of martelli-montanati's algorithm. *LNCS*, 170:224–248, May 1984.

[47] T. Kyntaja. A logic of authentication by burrows, abadi and needham, 1995. Link.

[48] G. Lowe. Toward a completeness result for model checking protocols. *In proceedings of the 11th Computer Security Foundation Workshop (CSFW'98)*, pages 96–105, June 1998. Link.

[49] G. Lowe. Toward a completeness result for model checking protocols. Technical Report 6, Departement of Mathematics and Computer Science, University of Leicester, 1998. Link.

[50] G.S. Makanin. The problem of solvability of equations in a free semigroup. *Math. Sbornik (English transl. in Math. USSR Sbornik 32)*, 103:147–236, 1977.

[51] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages ans Systems*, 4(2):258–282, April 1982.

[52] F. Massicotte. Une thorie des types pour les classes de failles dans les cryptoprotocoles. Master's thesis, Université Laval, July 2000.

[53] P. Mateus, J. Mitchell, and A. Scedrov. Composition of cryptographic protocols in a probabilistic polynomial-time process calculus. *In proceedings of the 14th International Conference on Concurrency Theory - LNCS*, 2761:327–349, September 2003. Link.

[54] C. Meadows. The nrl protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996. Link.

[55] C. Meadows. Formal methods for cryptographic protocol analysis: Emerging issues and trends. 2003. Link.

[56] C.A. Meadows. Formal verification of cryptographic protocols: A survey. *International Conference on the Theory and Application of Cryptology (ASIACRYPT'94) - LNCS*, 917:133–150, 1995. Link.

[57] M. Mejri. A formal automatic verification of authentication cryptographic protocols. Master's thesis, Université Laval, December 1997.

[58] M. Mejri. *From Type Theory to the Verification of Security Protocols*. PhD thesis, Université Laval, Febuary 2001.

[59] M. Mejri. Chaotic protocol. *In proceedings of the 2004 International Conference on Computational Science and its Applications (ICCSA'2004) - LNCS*, 3043:938–948, May 2004.

[60] A.J. Menezes, P.C. Van Oorschot, and S.A. Vanstone. *Handbook of Apllied Cryptography*. CRC Press, 5 edition, August 2001. Link.

[61] R. Milner. The polyadic π-calculus: a tutorial. *Logic and Algebra of Specification, Springer-Verlag*, 1993. Link.

[62] R.A. Mollin. *An Introduction to Cryptography*. Discrete Mathematics and Its Applications. CRC Press, October 2000.

[63] R.A. Mollin. *RSA and public-key cryptography*. 2003.

[64] P. Narendran and F. Otto. Some results on equational unification. *International Conference on Automated Deduction (CADE'10) - LNCS*, 449:276–292, July 1990.

[65] T. Nipkow. Unification in primal algebras, their powers and their varieties. *Journal of the ACM*, 37(4):742–777, October 1990.

[66] L.C. Paulson. Proving properties of security protocols by induction. *In proceedings of the 10th Computer Security Foundations Workshop (CSFW'97)*, pages 70–83, June 1997. Link.

[67] L.C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998. Link.

[68] A. Perrig and D. Song. A first step towards the automatic generation of security protocol. *In proceedings of Network and Distributed System Security*, pages 73–83, February 2000. Link.

[69] G. Plotkin. Building-in equational theories. *In Proceedings of the 7th Annual Machine Intellignece Workshop)*, 7:73–90, 1972.

[70] T. Renji. On finite automaton one-key cryptosystems. *Lecture Notes in Computer Science*, 809:135–148, 1994.

[71] T. Renji and S. Chen. On finite automaton public-key cryptosystem. *Theoretical Computer Science*, 226:143–172, 1999.

[72] G. Robin. *Algorithmique et Cryptographie*, volume 8 of *Mathmatiques et applications*. ellipses, 1991.

[73] C. Rudolph. *A Model for Secure Protocols and its Application to Systematic Design of Cryptographic Protocols*. PhD thesis, J.-W. Goethe University Frankfurt, December 2001. Link.

[74] H. Saïdi. Toward automatic synthesis of security protocols. *In proceeding of the Logic-Based Program Synthesis Workshop*, March 2002. Link.

[75] A. Salomaa. *Public-Key Cryptography*. 1996.

[76] B. Schneier. *Applied Cryptography*. WILEY, second edition, 1996.

[77] J. H. Siekmann. Universal unification. *LNCS*, 170:1–42, May 1984.

[78] J. H. Siekmann. *Unification Theory*. Number 221 in A. Linguistic Agency University of Duisburg, March 1988.

[79] S. Singh. *The Code Book*. Doubleday, August 2000.

[80] W. Snyder. *A Proof Theory for General Unification*, volume 11 of *Progress in Computer Science and Applied Logic*. Birkhäuser, birkhäuser edition, 1991.

[81] D.X. Song. Athena: a new efficient automatic checker for security protocol analysis. Link.

[82] D.X. Song, A. Perrig, and D. Phan. AGVI - automatic generation, verification and implementation of security protocols. *In proceedings of the 13th International Conference on Computer Aided Verification (CAV'01) - LNCS*, 2001.

[83] M. J. Toussaint. *Verification of Cryptographic Protocols.* PhD thesis, Université de Liège, 1991.

[84] C. Walther. Many-sorted unification. *Journal of the ACM*, 35(1):1–17, January 1988.

[85] J.M. Wing. A symbiotic relationship between formal methods and security. *In proceedings pf the Workshops on Computer Security, Fault Tolerance, and Software Assurance: From Needs to Solution. Technical report CMU-CS-98-188*, 1998. Link.

Appendix A

Every rule should have finitely many premises

We argue here that under some conditions, for every rule R with infinitely many premises there exists another rule R' with finitely many premises such that $R \equiv R'$. Two rules R, R' are equivalent when R can be used iff R' can be used as well. We do not aim to provide a formal structured proof. Instead, we give the intuitions with which anyone should be convinced. We will illustrate that there exists a bound α on the number of useful premise for a rule. We are not yet able to compute such a bound, nor automatically remove some premises from a rule with more than α premises. This will possibly be a problem when implementing our theory[1]. However, from the theoretical point of view, everything will be fine.

The conditions we need are:

- the set of term structures is finite;
- the set of constants is finite;
- rules are premise-disjoint[2];

Suppose R has infinitely many premises. Since the set of term structures is finite, it must be the case that infinitely many premises of R are derived from the same term structure s, let $\mathcal{P}(R)_s$ be those premises. If the terms of $\mathcal{P}(R)_s$ contain no atomic

[1] Note that the problem never arisen in the tests we did with our program described in Section 5.7.
[2] Remember that this only concerns non-atomic variables.

variables, then, since they are disjoint over their set of non-atomic variables and there is finitely many constants, it is clear that infinitely many of them are equal modulo renaming. Example A.0.1 takes the case where we have only non-atomic variables.

Example A.0.1
Consider the following term structure $s = f(\tau_1, h(\tau_2, \tau_3))$ where τ_1 is non-atomic while τ_2, τ_3 are atomic (we suppose, for simplicity, that these types are incomparable). Let the set of constants be $\{c_2, c'_2, c_3, c'_3\}$ where c_i and c'_i are of type τ_i. Since we consider the case where terms have no atomic variables, the following terms are possible derivations from s:

- $t_1 = f(X_1, (h(c_2, c_3)))$
- $t_2 = f(X_2, (h(c_2, c'_3)))$
- $t_3 = f(X_3, (h(c'_2, c_3)))$
- $t_4 = f(X_4, (h(c'_2, c'_3)))$

Any other term t' derived from s will be a renaming of one of the above, say t. Since the conclusion of a rule is simple-linear, it may contain at most one non-atomic variable. Thus either t' or t is useless.

Unfortunately, it is much more complicated when atomic variables come in. The problem with atomic variables is that premises are not required[3] to be disjoint over atomic variables. Thus two premises of a same rule can share an atomic variable. This sharing creates possible links that are not easily analyzed. Example A.0.2 shows how such links may cause problems.

Example A.0.2
Consider the following term structure $s = h(\tau_1, \tau_1)$ where τ_1 is atomic. One might create a rule with infinitely many premises which are all essential by using the following pattern (we present only the premises and leave the conclusion aside):

- $h(x_1, x_2)$
- $h(x_1, x_2)\ h(x_2, x_3)$
- $h(x_1, x_2)\ h(x_2, x_3)\ h(x_3, x_4)$
- $h(x_1, x_2)\ h(x_2, x_3)\ h(x_3, x_4)\ h(x_4, x_5)$

[3] We do not want to add this requirement since it will cost a lot on our expressivity.

- ...
- $h(x_1, x_2)\ h(x_2, x_3)\ h(x_3, x_4)\ \ldots h(n, n+1)$
- ...

It seems that the dependencies between the premises prevent us from removing any of them. However, if we consider the problem with more information, this will change. Let $\{c_1, c_2\}$ be the set of constants of type τ_1. With this, we are able to show that if we have three premises derived from s, then one of them is useless. This can be proved as follows:

Proof:
The set of ground[4] term formed from s is $\{h(c_1, c_1), h(c_1, c_2), h(c_2, c_1), h(c_2, c_2)\}$. Suppose we have the following three premises in a rule: $h(x_1, x_2)\ h(x_2, x_3)\ h(x_3, x_4)$. Then:

- If $h(x_1, x_2)$ is provable[5] by $h(c_1, c_2)$, then $h(x_2, x_3)$ has three choices
  - If $h(x_2, x_3)$ is provable by $h(c_2, c_1)$, then $h(x_3, x_4)$ is useless since it is provable by $h(c_1, c_2)$.
  - If $h(x_2, x_3)$ is provable by $h(c_2.c_2)$, then $h(x_3, x_4)$ is useless since it is also provable by $h(c_2.c_2)$.
  - Else, $h(c_2, x_3)$ cannot be proved. Thus either the rule cannot be used or we must find another way to prove $h(x_1, x_2)$.
- The same reasoning applies for the case where $h(x_1, x_2)$ is provable by $h(c_2, c_1)$.
- If $h(x_1, x_2)$ is provable by $h(c_1, c_1)$, then $h(x_2, x_3)$ and $h(x_3, x_4)$ are both provable by $h(c_1, c_1)$ and thus both useless.
- The same reasoning applies for the case where $h(x_1, x_2)$ is provable by $h(c_2, c_2)$
- Else, $h(x_1, x_2)$ is not provable and hence the rule cannot be used. The whole rule is thus useless.

□

[4] A term is ground if it contains no variables.

[5] $h(x_1, x_2)$ is provable by $h(c_1, c_2)$ means that $h(x_1, x_2)$ can be proved by replacing x_1 with c_1 and x_2 with c_2.

Although the above example is far from being a formal proof for the existence of a bound on the number of premises for a rule, we can use some of its ideas to yield a more general argument.

Consider a set of rules S, where each rule has finitely many premises (this would correspond to the initial inference system of our transformation process[6]). We will take every rule $R \in S$ and generate the set of semi-ground[7] rules formed from R, noted $\mathcal{SG}(R)$. $\mathcal{SG}(R)$ is clearly finite since R has finitely many terms and we have finitely many constants. It should be clear that $R \equiv \mathcal{SG}(R)$ (replacing rule R is S by the set of rules $\mathcal{SG}(R)$ does not change the theory of S). Now, we can form

$$S' = \bigcup_{R \in S} \mathcal{SG}(R)$$

such that $TH(S) = TH(S')$. It is easy to see that any rule R' built in the transformation process over S' will be semi-ground. Using the discussion related to Example A.0.1, we can conclude that a rule such as R' cannot have infinitely many premises.

Although the idea of generating semi-ground rules works well in theory, it may not be as good in practice. There is a possible overhead of rule compositions created by rule duplication (one rule replaced by many semi-ground rules) that may cause time and memory problems. However, we can now affirm that a rule is not allowed to have infinitely many premises. How to implement this is currently an open question. Other interesting, and unanswered, questions are:

- Given a rule R with infinitely many premises. What is the number of premises we should keep to yield an equivalent rule with finitely many premises?
- Given a rule R with infinitely many premises. Which premises could be removed without changing the rule?

[6] It is realistic and quite natural to impose the restriction that the initial inference system is composed only of rules having finitely many premises.

[7] We call semi-ground a term without atomic variables. A semi-ground rule is a rule formed only with semi-ground terms.

Appendix B

An example of complete transformation

We present here the complete transformation steps used in the proof of Lemma 5.5.1 on the inference system of Table B.1. We use the transformation algorithm presented in Section 3.5, in particular, we use the simplification of inference system defined in 3.4.4.

$T(S^0)$	$NT(S^0)$
$R_1 = \frac{\square}{P_1}$ $R_2 = \frac{\square}{K_{IS}}$	$R_8 = \frac{X_6.X_7}{X_6}$ $R_9 = \frac{X_8.X_9}{X_9}$
$R_3 = \frac{\square}{m}$ $R_4 = \frac{X_1}{\{X_1\}_{K_{P_2S}}}$	$R_{11} = \frac{\{X_{10}\}_{K_{P_9P_{10}}} \quad K_{P_9P_{10}}}{X_{10}}$
$R_5 = \frac{P_3 \quad X_2}{\{P_3.X_2\}_{K_{P_4S}}}$ $R_6 = \frac{X_3 \quad X_4}{X_3.X_4}$	$R_{12} = \frac{\{P_{11}.\{X_{11}\}_{K_{P_{11}S}}\}_{K_{P_{12}S}}}{\{X_{11}\}_{K_{P_{12}S}}}$
$R_7 = \frac{X_5 \quad K_{P_5P_6}}{\{X_5\}_{K_{P_5P_6}}}$ $R_{10} = \frac{P_7}{N^a_{P_8}}$	

Table B.1: S^0

The possible compositions of a rule from $NT(S^0)$ with rules from $T(S^0)$ are:

$$R_{13} = R_8 \Uparrow R_6 = \frac{X_3 \;\; X_4}{X_3} \qquad R_{14} = R_9 \Uparrow R_6 = \frac{X_3 \;\; X_4}{X_4}$$

$$R_{15} = R_{11} \Uparrow (R_4, R_2) = \frac{X_1}{X_1} \qquad R_{16} = R_{11} \Uparrow (R_5, R_2) = \frac{P_3 \;\; X_2}{P_3.X_2}$$

$$R_{17} = R_{11} \Uparrow (R_7, R_2) = \frac{X_5 \;\; K_{IS}}{X_5} \qquad R_{19} = R_{12} \Uparrow R_4 = \frac{P_{11}.\{X_{11}\}_{K_{P_{11}S}}}{\{X_{11}\}_{K_{P_{12}S}}}$$

$$R_{20} = R_{12} \Uparrow R_5 = \frac{P_{11} \;\; \{X_{11}\}_{K_{P_{11}S}}}{\{X_{11}\}_{K_{P_{12}S}}} \qquad R_{21} = R_{12} \Uparrow R_7 = \frac{P_{11}.\{X_{11}\}_{K_{P_{11}S}}}{\{X_{11}\}_{K_{P_{12}S}}}$$

When eliminating redundant rules, we eliminate R_{13}, R_{14}, R_{15} and R_{17} since they are all self-redundant. We also eliminate R_{16} and R_{21} since they are instances of R_6 and R_{19} respectively. Finally, R_{10} is made redundant by R_{18}. Once the redundant rules have been removed and the variables in the new rules renamed, we get the inference system S^1 given in Table B.2.

$T(S^1)$	$NT(S^1)$
$R_1 = \frac{\square}{P_1} \quad R_2 = \frac{\square}{K_{IS}}$	$R_8 = \frac{X_6.X_7}{X_6} \quad R_9 = \frac{X_8.X_9}{X_9}$
$R_3 = \frac{\square}{m} \quad R_4 = \frac{X_1}{\{X_1\}_{K_{P_2S}}}$	$R_{11} = \frac{\{X_{10}\}_{K_{P_9P_{10}}} \;\; K_{P_9P_{10}}}{X_{10}}$
$R_5 = \frac{P_3 \;\; X_2}{\{P_3.X_2\}_{K_{P_4S}}} \quad R_6 = \frac{X_3 \;\; X_4}{X_3.X_4}$	$R_{12} = \frac{\{P_{11}.\{X_{11}\}_{K_{P_{11}S}}\}_{K_{P_{12}S}}}{\{X_{11}\}_{K_{P_{12}S}}}$
$R_7 = \frac{X_5 \;\; K_{P_5P_6}}{\{X_5\}_{K_{P_5P_6}}} \quad R_{10} = \frac{P_7}{N^{\alpha}_{P_8}}$	$R_{19} = \frac{p_{14}.\{X_{12}\}_{K_{p_{14}S}}}{\{X_{12}\}_{K_{p_{15}S}}}$
	$R_{20} = \frac{p_{16} \;\; \{X_{13}\}_{K_{p_{16}S}}}{\{X_{13}\}_{K_{p_{17}S}}}$

Table B.2: S^1

The possible compositions of a rule from $NT(S^1)$ with rules from $T(S^1)$ are (we omit the composition already checked in the previous step):

$$R_{22} = R_{19} \Uparrow R_6 = \frac{P_{14} \ \{X_{12}\}_{K_{P_{14}S}}}{\{X_{12}\}_{K_{P_{15}S}}} \qquad R_{23} = R_{20} \Uparrow (R_1, R_4) = \frac{X_{13}}{\{X_{13}\}_{K_{P_{17}S}}}$$

$$R_{24} = R_{20} \Uparrow (R_1, R_5) = \frac{P_3 \ X_2}{\{P_3.X_2\}_{K_{P_{17}S}}} \quad R_{25} = R_{20} \Uparrow (R_1, R_7) = \frac{X_5 \ K_{P_2S}}{\{X_5\}_{K_{P_{17}S}}}$$

The four new rules are found to be redundant. Rules R_{23} and R_{25} are both made redundant by R_4, R_{22} is made redundant by R_{20} and R_{24} is an instantiation of R_5. Once the redundant rules are removed, we have $S^2 = S^1$. We can take $T(S^2)$ (which is the same as $T(S^1)$) as an inference system equivalent to the initial inference system.

Appendix C

Another example of complete transformation

We present here the complete transformation steps used in the proof of Lemma 5.5.2 on the inference system of Table C.1. We use the transformation algorithm presented in Section 3.5, in particular, we use the simplification of inference system defined in 3.4.4.

$T(S^0)$	$NT(S^0)$
$R_1 = \frac{\square}{P_1}$ $R_2 = \frac{\square}{K_{IS}}$	$R_8 = \frac{X_6.X_7}{X_6}$ $R_9 = \frac{X_8.X_9}{X_9}$
$R_3 = \frac{\square}{\{m\}_{K_{AS}}}$ $R_4 = \frac{X_1}{\{X_1\}_{K_{p_2S}}}$	$R_{11} = \frac{\{X_{10}\}_{K_{P_9P_{10}}} \quad K_{P_9P_{10}}}{X_{10}}$
$R_5 = \frac{P_3 \quad X_2}{\{P_3.X_2\}_{K_{P_4S}}}$ $R_6 = \frac{X_3 \quad X_4}{X_3.X_4}$	$R_{12} = \frac{\{P_{11}.\{X_{11}\}_{K_{P_{11}S}}\}_{K_{P_{12}S}}}{\{X_{11}\}_{K_{P_{12}S}}}$
$R_7 = \frac{X_5 \quad K_{P_5P_6}}{\{X_5\}_{K_{P_5P_6}}}$ $R_{10} = \frac{P_7}{N^{\alpha}_{P_8}}$	

Table C.1: S^0

The possible compositions of a rule from $NT(S^0)$ with rules from $T(S^0)$ are:

$$R_{13} = R_8 \Uparrow R_6 = \frac{X_3 \;\; X_4}{X_3} \qquad R_{14} = R_9 \Uparrow R_6 = \frac{X_3 \;\; X_4}{X_4}$$

$$R_{15} = R_{11} \Uparrow (R_4, R_2) = \frac{X_1}{X_1} \qquad R_{16} = R_{11} \Uparrow (R_5, R_2) = \frac{P_3 \;\; X_2}{P_3.X_2}$$

$$R_{17} = R_{11} \Uparrow (R_7, R_2) = \frac{X_5 \;\; K_{IS}}{X_5} \qquad R_{19} = R_{12} \Uparrow R_4 = \frac{P_{11}.\{X_{11}\}_{K_{P_{11}S}}}{\{X_{11}\}_{K_{P_{12}S}}}$$

$$R_{20} = R_{12} \Uparrow R_5 = \frac{P_{11} \;\; \{X_{11}\}_{K_{P_{11}S}}}{\{X_{11}\}_{K_{P_{12}S}}} \qquad R_{21} = R_{12} \Uparrow R_7 = \frac{P_{11}.\{X_{11}\}_{K_{P_{11}S}}}{\{X_{11}\}_{K_{P_{12}S}}}$$

When eliminating redundant rules, we eliminate R_{13}, R_{14}, R_{15} and R_{17} since they are all self-redundant. We also eliminate R_{16} and R_{21} since they are instances of R_6 and R_{19} respectively. Once the redundant rules have been removed and the variables in the new rules renamed, we get the inference system S^1 given in Table C.2.

$T(S^1)$	$NT(S^1)$
$R_1 = \frac{\square}{P_1} \quad R_2 = \frac{\square}{K_{IS}}$	$R_8 = \frac{X_6.X_7}{X_6} \quad R_9 = \frac{X_8.X_9}{X_9}$
$R_3 = \frac{\square}{\{m\}_{K_{AS}}} \quad R_4 = \frac{X_1}{\{X_1\}_{K_{p_2S}}}$	$R_{11} = \frac{\{X_{10}\}_{K_{P_9P_{10}}} \;\; K_{P_9P_{10}}}{X_{10}}$
$R_5 = \frac{P_3 \;\; X_2}{\{P_3.X_2\}_{K_{P_4S}}} \quad R_6 = \frac{X_3 \;\; X_4}{X_3.X_4}$	$R_{12} = \frac{\{P_{11}.\{X_{11}\}_{K_{P_{11}S}}\}_{K_{P_{12}S}}}{\{X_{11}\}_{K_{P_{12}S}}}$
$R_7 = \frac{X_5 \;\; K_{P_5P_6}}{\{X_5\}_{K_{P_5P_6}}} \quad R_{10} = \frac{P_7}{N^{\alpha}_{P_8}}$	$R_{19} = \frac{p_{14}.\{X_{12}\}_{K_{p_{14}S}}}{\{X_{12}\}_{K_{p_{15}S}}}$
	$R_{20} = \frac{p_{16} \;\; \{X_{13}\}_{K_{p_{16}S}}}{\{X_{13}\}_{K_{p_{17}S}}}$

Table C.2: S^1

The possible compositions of a rule from $NT(S^1)$ with rules from $T(S^1)$ are (we omit the composition already checked in the previous step):

$$R_{22} = R_{19} \Uparrow R_6 = \frac{P_{14} \quad \{X_{12}\}_{K_{P_{14}S}}}{\{X_{12}\}_{K_{P_{15}S}}} \qquad R_{23} = R_{20} \Uparrow (R_1, R_4) = \frac{X_{13}}{\{X_{13}\}_{K_{P_{17}S}}}$$

$$R_{24} = R_{20} \Uparrow (R_1, R_5) = \frac{P_3 \quad X_2}{\{P_3.X_2\}_{K_{P_{17}S}}} \qquad R_{25} = R_{20} \Uparrow (R_1, R_7) = \frac{X_5 \quad K_{P_2S}}{\{X_5\}_{K_{P_{17}S}}}$$

$$R_{26} = R_{20} \Uparrow (R_1, R_3) = \frac{\square}{\{m\}_{K_{P_{17}S}}}$$

The first four new rules are found to be redundant. Rules R_{23} and R_{25} are both made redundant by R_4, R_{22} is made redundant by R_{20} and R_{24} is an instantiation of R_5. R_3 is also made redundant by R_{26}. The inference system S^2 is found in Table C.3.

$T(S^2)$	$NT(S^2)$
$R_1 = \frac{\square}{P_1} \quad R_2 = \frac{\square}{K_{IS}}$	$R_8 = \frac{X_6.X_7}{X_6} \quad R_9 = \frac{X_8.X_9}{X_9}$
$R_4 = \frac{X_1}{\{X_1\}_{K_{p_2S}}}$	$R_{11} = \frac{\{X_{10}\}_{K_{P_9P_{10}}} \quad K_{P_9P_{10}}}{X_{10}}$
$R_5 = \frac{P_3 \quad X_2}{\{P_3.X_2\}_{K_{P_4S}}} \quad R_6 = \frac{X_3 \quad X_4}{X_3.X_4}$	$R_{12} = \frac{\{P_{11}.\{X_{11}\}_{K_{P_{11}S}}\}_{K_{P_{12}S}}}{\{X_{11}\}_{K_{P_{12}S}}}$
$R_7 = \frac{X_5 \quad K_{P_5P_6}}{\{X_5\}_{K_{P_5P_6}}} \quad R_{10} = \frac{P_7}{N^{\alpha}_{P_8}}$	$R_{19} = \frac{p_{14}.\{X_{12}\}_{K_{p_{14}S}}}{\{X_{12}\}_{K_{p_{15}S}}}$
$R_{26} = \frac{\square}{\{m\}_{K)_{p_{18}S}}}$	$R_{20} = \frac{p_{16} \quad \{X_{13}\}_{K_{p_{16}S}}}{\{X_{13}\}_{K_{p_{17}S}}}$

Table C.3: S^2

The possible compositions of a rule from $NT(S^2)$ with rules from $T(S^2)$ are (we omit the composition already checked in the previous step):

$$R_{27} = R_{11} \Uparrow (R_{26}, R_2) = \frac{\square}{m} \quad R_{28} = R_20 \Uparrow (R_1, R_{26}) = \frac{\square}{\{m\}_{K_{p_{p_{17}S}}}}$$

The rule R_{28} is made redundant by R_{26}, while R_{27} is not redundant. The inference system S^3 is as shown in Table C.4.

$T(S^3)$	$NT(S^3)$
$R_1 = \frac{\square}{P_1}$ $R_2 = \frac{\square}{K_{IS}}$	$R_8 = \frac{X_6.X_7}{X_6}$ $R_9 = \frac{X_8.X_9}{X_9}$
$R_4 = \frac{X_1}{\{X_1\}_{K_{p_2S}}}$	$R_{11} = \frac{\{X_{10}\}_{K_{P_9P_{10}}} \quad K_{P_9P_{10}}}{X_{10}}$
$R_5 = \frac{P_3 \quad X_2}{\{P_3.X_2\}_{K_{P_4S}}}$ $R_6 = \frac{X_3 \quad X_4}{X_3.X_4}$	$R_{12} = \frac{\{P_{11}.\{X_{11}\}_{K_{P_{11}S}}\}_{K_{P_{12}S}}}{\{X_{11}\}_{K_{P_{12}S}}}$
$R_7 = \frac{X_5 \quad K_{P_5P_6}}{\{X_5\}_{K_{P_5P_6}}}$ $R_{10} = \frac{P_7}{N^{\alpha}_{P_8}}$	$R_{19} = \frac{p_{14}.\{X_{12}\}_{K_{p_{14}S}}}{\{X_{12}\}_{K_{p_{15}S}}}$
$R_{26} = \frac{\square}{\{m\}_{K)_{p_{18}S}}}$ $R_{27} = \frac{\square}{m}$	$R_{20} = \frac{p_{16} \quad \{X_{13}\}_{K_{p_{16}S}}}{\{X_{13}\}_{K_{p_{17}S}}}$

Table C.4: S^3

Since there is no new composition possible from a rule in $NT(S^3)$ and rules in $T(S^3)$, it is clear that $S^4 = S^3$. We can take $T(S^4)$ (which is the same as $T(S^3)$) as an inference system equivalent to the inference system S^0 of Table C.1.

Index