Defining an Approximation to Formally Verify Cryptographic Protocols

Frédéric Enoha OEHL D.E.A.

Thesis for Degree of Master of Science

Dublin City University
School of Computing
Supervisor: Dr. David Sinclair

July 2006

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award Master of Science is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: 

ID No.: 50161423

Date: 22/09/06

Abstract

Electronic forms of communication are abundant in todays world, and much emphasis is placed on these methods of communication in every day life. In order to guarantee the secrecy and authenticity of information exchanged, it is vital to formally verify the cryptographic protocols used in these forms of communications. This verification does, however, present many challenges. The systems to verify are infinite, with an infinite number of sessions and of participants. As if this was not enough, there is also a reactive element to deal with: the intruder. The intruder will attack the protocol to achieve his goal: usurping identity, stealing confidential information, etc. His behavior is unpredictable!

This thesis describes a method of verification based on the verification of systems by approximation. Starting from an initial configuration of the network, an over-approximation of the set of messages exchanged is automatically computed. Secrecy and authentication properties can then be checked on the approximated system.

Starting from an existing semi-automatic proof method developed by Genet and Klay, an automatic solution is developed.

This thesis defines a particular approximation function that can be generated automatically and that guarantees that the computation of the approximated system terminates.

The verification by approximation only tells if properties are verified. When the verification fails no conclusion can be drawn on the property. Thus, this thesis also shows how the approximation technique can easily be combined with another verification technique to combine the strengths of both approaches.

Finally, the tool developed to validate these developments and the results of

cryptographic protocol verifications carried out in the course of this research are included.

Acknowledgements

Without the help and support of some people, I would not have been able to write this dissertation. This section gives me the opportunity to thank them.

Firstly, I'd like to thank my supervisor Dr. David Sinclair for his assistance and guidance over the last six years.

Secondly, I'd like to thank my examiners for having accepted to review this thesis.

I want also to thank all the researchers from the Computing School in Dublin City University and those from the Laboratoire Informatique de Franche-Comté for their comments and questions that helped me to go forward. I also want to thank Paula from the School Office for her precious help regarding administrative matters.

Finally, I cannot finish without thanking my girlfriend, my parents, my brother and my grand-parents for their ever present love and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the development of electronic communications (E-commerce, email, mobile phone services, etc.) it becomes vital to guarantee the secrecy and the authenticity of the information exchanged. Cryptographic protocols define precisely how messages using cryptographic primitives must be exchanged between agents participating in a transaction. They are used to secure the communications. Such protocols are already implemented in computer networks, ATM machines, commercial websites, etc.

Designing and verifying protocols for these contexts are not easy, however it is necessary as flawed protocols can have serious consequences. The following examples illustrate some of the implication.

The *"Yescards"* in France [Sci02] can be used to pay transactions under 91.47 euros. For transactions under 91.47 euros the system does not do a full authentication of the card. It only checks a 320 bit key and does not do any verification with the GIE (Groupement d'Intérêt Bancaire) Carte bancaire server. This 320 bit key was broken by Serge Humpich[1]. For amounts above 91.47 euros the equipment in the shop calls an authentication server of the merchant's bank to authenticate the card. Moreover, the merchant can manually force the authentication if he wants to decrease the payment time at the cash register and then reduce the waiting time at

[1] information about Mr Humpich's story can be found at http://www.parodie.com/humpich/

the lane. Nevertheless, at the end of the day all the payments made by card are sent to the merchant's bank computer. A full authentication of the cards used is then done over night, and identities of fraudulent cards are then reported to all the payment systems by the following morning. Serge Humpich, by studying the partial authentication process, was able to find how to generate valid keys. With this information, he was able to create chip cards that were authenticated by terminals every time. However, Serge Humpich's cards were valid for 24 hours at most.

An even more recent example of breaking protocols involved researchers from the Ecole Polytechnique Fédérale de Lausanne (EPFL), who broke the SSL (Secure Socket Layer) protocol and were able to recover usernames and passwords sent with Outlook Express 6 to an IMAP[2] server. Vaudeney [Vau02] explains how an intruder can recover encrypted information in Cipher Block Chaining mode[3] by exploiting error messages used in protocols. The sequence is as follow:

1. the attacker catches m which is the encryption of the information e exchanged between a client and a server.

2. the attacker builds a new message using m and some algorithms that take advantage of the properties of the CBC mode. The message is sent to the server.

3. the server sends an error message that the attacker analyzes to see if he made a good guess (and now has access to a piece of e) or not.

4. the attacker repeats the steps 2 and 3 till he recovers the information e.

To work, the method requires that the server does not end the communication after an error in the protocol. But as SSL does end the communication in the event of

[2]Internet Message Access Protocol (IMAP) is a method of accessing electronic messages kept on a (possibly shared) mail server.

[3]the Cipher Block Chaining mode is a particular encryption process. First the information that must be encrypted is split in blocks of same size. Then the first block is XORed with a particular message before being encrypted. The second block is XORed with the result of the previous encryption before it is encrypted; the same process is repeated to encrypt the remaining the blocks.

an error message, several sessions of SSL sending the same information must be studied in order to succeed in recovering the information. The researchers from EPFL succeeded in recovering username and password sent with Outlook Express 6 to an IMAP server using this flaw [CHSV03].

Cryptographic protocols are communication protocols that use cryptography. Messages are encrypted using encryption keys and cryptographic algorithms, and cannot be decrypted without the correct decryption keys in a reasonable timeframe.

1.1 Cryptography

Cryptography (from the Greek hidden writing) is the study of means of converting information from comprehensible forms into incomprehensible forms. Cryptography concerns itself mainly with four objectives:

1. Confidentiality: the information cannot be understood by anyone for whom it was not intended.

2. Integrity: the information cannot be altered in storage or transit between the sender and intended receiver without the alteration being detected.

3. Non-repudiation: the creator/sender of the information cannot deny at a later stage his or her intentions in the creation or transmission of the information.

4. Authentication: the sender and receiver can confirm each other's identity and the origin/destination of the information.

Procedures and protocols that meet some or all of the above criteria are known as cryptosystems.

Figure 1.1 introduces some basics terms used in cryptography. The plaintext refers to the comprehensible form of a piece of information while the ciphertext refers to the incomprehensible form. Encryption is the process of encoding the information, and decryption is the decoding process. The encryption and decryption keys are confidential, and are required for the corresponding process to work.

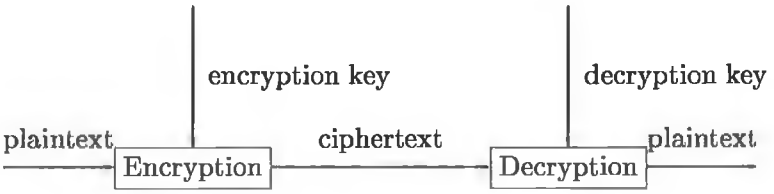


Figure 1.1: Cryptography

A well-known cryptosystem is the Caesar Number [Sue00]. Caesar encoded his mails by moving forward letters. Figure 1.2 shows an example with a gap of four between letters. The encryption is done by moving of 4 letters forward as the encryption key X is equal to 4. The decryption is done by moving of 22 letters forward as the decryption key Y is equal to 22.

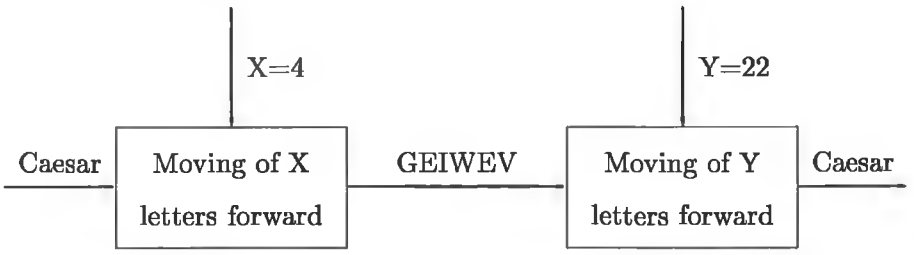


Figure 1.2: Caesar cipher

This encryption algorithm is very simple and would not resist a brute force attack (attack where all the possible gaps will be tested). Today, cryptography is mainly based on the number theory and takes advantage of the properties of specific classes of numbers. It is possible to distinguish between two types of cryptography: symmetric and asymmetric.

In symmetric cryptography, the keys used to encrypt and to decrypt are the same. In asymmetric cryptography, keys are different. An example of asymmetric cryptography is the public key cryptography where one key is usually available to everybody (for encryption), the public key, and only one person knows the other key (for decryption), the private key.

One of the most popular algorithms in cryptography is RSA [RSA78] (Rivest, Shamir and Adleman were the inventors of this algorithm). RSA is an asymmetric

algorithm based on the assumption that "factoring" is a difficult operation. Particular pairs of RSA keys have been broken over the years, most recently a 640 bit key pair in November 2005. More information about RSA and the contest to break the 2048 bit key pair (there are 200 000 dollars to win!) can be found online at http://www.rsasecurity.com/rsalabs/.

Breaking[4] an encryption scheme (algorithm) is difficult and requires lots of resources. For example, it took 4 months and 300 computers to break RSA with a 512 bit key [Zim99]. Most of the attacks found on cryptographic protocols assume a perfect encryption (unbreakable encryption) and take advantage of flaws in the protocols (e.g. replaying previous messages, etc.). Later on in the chapter the following attacks will be introduced in more details:

- man in the middle attack: the attacker uses the protocol to usurp identity and to get critical information;
- type attack: the attacker plays with the format of the messages to discover critical information;
- replay attack: the attacker replays old messages hoping that he can receive critical information.

1.2 Cryptographic Protocols

Protocols define how messages between agents participating in a transaction are exchanged. In this section, the protocol notation is introduced using the Needham-Schroeder protocol [NS78].

Protocols are used in possibly hostile networks as has been shown at the beginning of this chapter. Nevertheless, they should be able to achieve their goals regardless of the attacks. The encryption algorithms briefly introduced in section Section 1.1 can be attacked but it would require powerful computing resources. In fact, many attacks on protocols do not depend on the weaknesses of the encryption

[4] trying to recover the keys used in the cryptosystem

algorithm used. Examples of attacks that can be discovered at an abstract level are presented in this section. More attacks are described in [CJ97] or on the SPORE website [5]. Finally the properties that must be verified by protocols are presented.

1.2.1 Needham-Schroeder protocol

The Needham-Schroeder protocol defines the exchanges between two agents, Alice and Bob. The goal of this protocol is to establish mutual authentication between Alice and Bob. Figure 1.3 shows how this protocol is represented using the usual notation of cryptographic protocols. This protocol is composed of three messages[6].

Message 1:	$A \Longrightarrow B : \{Na, A\}_{Kb}$
Message 2:	$B \Longrightarrow A : \{Na, Nb\}_{Ka}$
Message 3:	$A \Longrightarrow B : \{Nb\}_{Kb}$

Figure 1.3: Needham-Schroeder protocol

In the first message, Alice sends her name A and a nonce Na (a nonce is an unguessable number that is typically randomly generated) encrypted with the public key of Bob, Kb. The encryption of a message m with the key k is represented by $\{m\}_k$. Therefore $\{Na, A\}_{Kb}$ in Figure 1.3 represents a message containing the nonce Na and the agent name A encrypted with the public key Kb.

When Bob receives $\{Na, A\}_{Kb}$, since he has the private key corresponding to the public key Kb, he has access to the encrypted information. He replies to Alice by sending her nonce, Na, and one he creates, Nb, back to her. This information is encrypted with Alice's public key, $\{Na, Nb\}_{Ka}$.

In the last step of the protocol, Alice receives Message 2. After decrypting the

[5] http://www.lsv.ens-cachan.fr/spore/

[6] The original protocol contains seven steps. The four extra steps explain that the agents get the public keys by contacting a server, which knows the public keys of all participants. As we are only interested in the authentication between the agents, we assume that each agent already knows the public key of the other and remove these four extra steps.

message with her private key, she recognizes her nonce Na, and replies to Bob with his nonce encrypted with his public key. When Bob receives Message 3, he recognizes his nonce and believes that he communicates with Alice. Both agents believe they are the only agents that know the nonce Nb and they will use it to authenticate themselves. Subsequently, when Alice receives any message with Nb inside, she will think that Bob sent it and vice versa.

Of course the protocol can be run several times by different agents with several nonces. One run of a protocol is called a session. For this protocol we can also identify two roles: initiator (sends Message 1 and Message 3) and responder (sends Message 2). Agents can play both roles.

1.2.2 Attacks

It is demonstrated here that by intercepting and possibly modifying the messages the intruder can catch secret information and usurp identities.

One of the common types of attacks is called the *"man in the middle attack"*. Figure 1.4 shows how this attack works on the Needham-Schroeder protocol using Lowe's assumption[7] that participants can be dishonest [Low95]:

Message 1 Alice initiates a communication with Yves by sending $\{Na, Alice\}_{Kyves}$.

Message 2 Yves usurps Alice's identity by sending $\{Na, Alice\}_{Kbob}$, the information he received encrypted with Bob's public key.

Message 3 When Bob receives Message 2, he thinks that Alice wants to communicate with him so he sends the message $\{Na, Nb\}_{Kalice}$ (Alice's nonce and his nonce) to Alice.

Message 4 Alice believes that Yves replied to her as she got back the nonce she used to communicate with Yves. Thus she sends back the nonce Nb to Yves ($\{Nb\}_{Kyves}$).

[7] without this assumption the protocol is secured

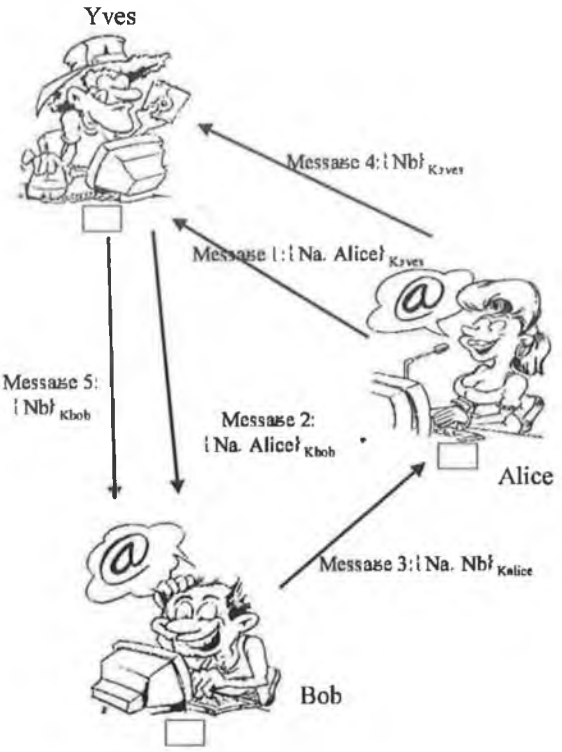


Figure 1.4: "Man in the middle attack"

Message 5 Yves can send Bob his nonce, $\{Nb\}_{Kbob}$. Henceforth each time that Bob receives a message with Nb, he will think that it comes from Alice but it will be from Yves.

This flaw was found and corrected by G. Lowe [Low95] in 1995. The correction is simple; the sender of the second message adds his name in the message. So after correction the third message looks like $\{Na, Nb, Bob\}_{Kalice}$ for our example. In the later chapters, it is proven that the Lowe corrected version of the Needham-Schroeder protocol is safe, assuming that "type attacks" are prevented.

A second type of attack is called the *"type attack"*. In this attack, the intruder tricks the honest agents by changing the format of messages. Figure 1.5 presents a

slightly modified version of the Needham-Schroeder-Lowe protocol [Low96]; in the first message the agent's names becomes the first element of the list of elements encrypted.

Message 1:	$A \Longrightarrow B : \{A, Na\}_{Kb}$
Message 2:	$B \Longrightarrow A : \{Na, Nb, B\}_{Ka}$
Message 3:	$A \Longrightarrow B : \{Nb\}_{Kb}$

Figure 1.5: Modified Needham-Schroeder-Lowe protocol

Figure 1.6 shows how this protocol is vulnerable to a type attack:

Message 1 Yves sends Alice's name and his name, $\{Alice, Yves\}_{Kbob}$, to Bob hoping that Bob will believe that Alice wants to start a communication.

Message 2 When Bob receives Message 1, he thinks that Alice initiated a communication. For him the information $Yves$ of the first message is Alice's nonce. Thus as he received a name and a nonce by following the protocol, Bob sends to Alice $\{Nb, Bob, Yves\}_{Kalice}$.

Message 3 When Alices receives Message 2, she interprets the concatenation of the two pieces of information Nb and Bob as the nonce created by Yves. Thus she thinks that Yves initiated communication with her since he sent his name, $Yves$, and a nonce, (Nb, Bob), to her. She then replies to his request of communication by sending $\{Nb, Bob, Na, Alice\}_{Kyves}$, and thus Yves can recover the nonce Nb.

Nevertheless, [HLS03] proves that by tagging each field of the message with some information indicating its intended type, the "type attack" can easily be prevented. Moreover [HLS03] justifies the widely used assumption within the verification techniques that all agents can identify the type of the information sent.

Another form of attack is the *"replay attack"*. Here the intruder replays old messages to get the information he wants or to usurp an identity. To illustrate this

Figure 1.6: "Type attack"

attack, we pick another protocol, the BAN-Yahalom protocol [BAN89, Syv94] (cf. Figure 1.7). In this protocol, Alice and Bob trust a server, S, to generate the shared key that they will later use to exchange their confidential information.

Message 1:	$\mathrm{A} \Longrightarrow \mathrm{B} : A, Na$
Message 2:	$\mathrm{B} \Longrightarrow \mathrm{S} : B, Nb, \{A, Na\}_{Kbs}$
Message 3:	$\mathrm{S} \Longrightarrow \mathrm{A} : Nb, \{B, Kab, Na\}_{Kas}, \{A, Kab, Nb\}_{Kbs}$
Message 4:	$\mathrm{A} \Longrightarrow \mathrm{B} : \{A, Kab, Nb\}_{Kbs}, \{Nb\}_{Kab}$

Figure 1.7: BAN-Yahalom protocol

Figure 1.7 introduces the protocol, first A sends his name, A, and a nonce, Na to B (Message 1). In the second message B sends his name, B, a nonce, Nb and information encrypted with the shared key between S and B, $\{A, Na\}_{Kbs}$. When the server S receives the message, it creates the key for a session between A and B, Kab and sends Message 3 to A. To end the communication, A sends B the Message 4 that contains the cypher text encrypted with the share key, Kbs and his nonce encrypted with the session key, $\{A, Kab, Nb\}_{Kbs}$.

Figure 1.8 presents the attack:

Message 1 Alice initiates a communication with Bob by sending her name and her nonce.

Message 2 Bob follows the protocol and sends the second message (Bob, Nb, $\{Alice, Na\}_{Kbs}$) to the server. Yves intercepts and forwards the message.

Message 3 Yves starts his attack by sending $Alice$, (Na, Nb), a message containing Alice's name, and the nonces created by Alice and Bob.

Message 4 When Bob receives Message 3, he thinks that Alice wants to communicate with him and he interprets the concatenation of Na and Nb as one nonce if type flaw attacks are possible. Bob still follows the protocol and sends $Bob, Nb1, \{Alice, Na, Nb\}_{Kbs}$.

Message 5 Yves intercepts and blocks Message 4, and with the information he caught he builds the message $\{Alice, Na, Nb\}_{Kbs}, \{Nb\}_{Na}$. When Bob receives this message, he thinks that he and Alice can communicate safely. But in fact, what he interprets as his shared key with Alice is in reality a nonce. And Yves can use this nonce to usurp Alice's identity.

This attack differs from the man-in-the-middle one because the intruder re-sends information that he caught over the network and he is not replaying information sent to him.

Thus it has been shown that it is not always necessary for the intruder to break encryption algorithms to undermine protocols.

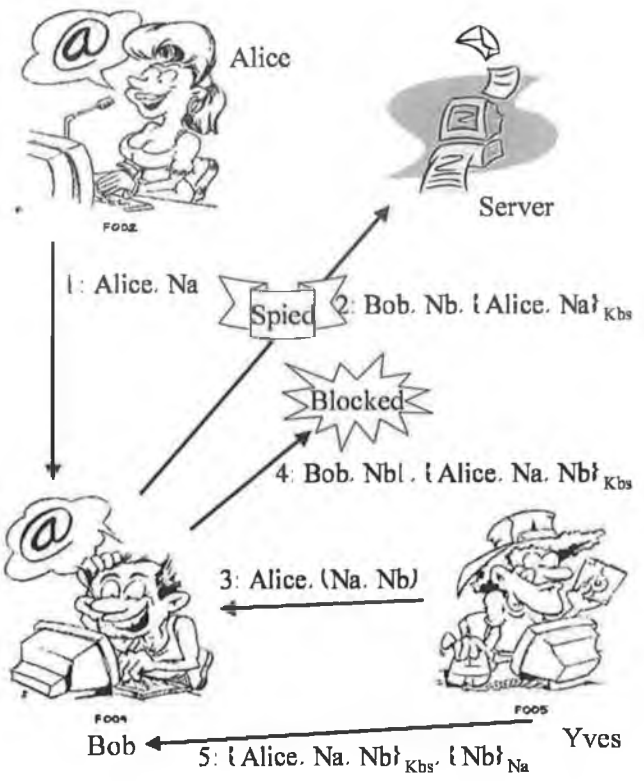


Figure 1.8: "Replay attack"

1.2.3 Properties

When people think of exchanging information over open networks, two properties come to mind.

The first property usually required of a cryptographic protocol is the *secrecy property*. Here the verification that critical information remains secret during protocol runs is required. Two type of secrecy can be distinguished [Aba00]:

- information m is secret when the intruder cannot find m regardless what he does (sender fraudulent messages, usurping identities, etc.).

- information m is secret when it is impossible to distinguish the session that uses m, from another one that uses a different piece of information m' instead of m.

Also two levels of secrecy can be distinguished, an absolute level and a temporary level. In the case of the absolute level, the information must be unknown by the intruder at any time. For the temporary level, we might want the secrecy of specified elements for a certain amount of time (for example only for one session of the protocol, etc.). In this thesis we will verify only secrecy properties with an absolute level of confidentiality.

The second property is the *authentication property*. There are many definitions for this property, Lowe [Low97a] gives 4 different definitions of the authentication while Schneider [Sch97] identifies 10 different definitions. The main idea is to ensure that at the end of the protocol the agents really communicate with the persons they intend to. As the intruder has the ability to replay all previous messages, authentication guarantees are usually expressed in the form "if Alice receives a message m_1 believed to be from Bob then Bob sent a message identical to m_1".

Depending on the protocol's goals, other properties must also be studied. For example for a protocol that is used to do online shopping, it may be required to ensure that the session keys used to encrypt credit card numbers cannot be re-used in another transaction instead of fresh session keys (*freshness property*) [Gon93, AN95, BCF02, PSW+01].

For other protocols such as telecommunications protocols, it may be required to check that the protocol guarantees that the intruder is unable to deduce the identity of the sender or receiver of messages (*privacy/anonymity property*). Abadi [Aba02] gives two protocols that satisfy that property. Onion Router [GRS99] and Crowds [RR98] systems are designed to prevent an intruder from determining the origination or destination of requests to servers.

For protocols used to sign contracts over the Internet, it may be required to ensure that the protocol satisfies a *fairness property* [MGK02]. The property makes

sure that an agent does not have an advantage over the others during the transaction. Kremer [KR02] presents an interesting approach based on the game theory, where a protocol is fair if a participant in collaboration with the communication subsystem does not have a strategy to receive a signed contract without the other participant also having a strategy to receive a signed contract.

1.3 Problems Raised by Protocol Verification

The verification of protocols is difficult because:

- the number of sessions is potentially infinite;
- the number of participants is potentially infinite;
- the message sizes could be undefined (as it was shown with type flaw attacks);
- after each step the intruder can learn new information and send fraudulent messages.

Even by assuming perfect encryption, the problem is still hard to deal with. Several methods have been developed to solve this problem; some methods are more efficient than others to verify specific properties. The verification of cryptographic protocols is equivalent to the verification of an infinite system and is undecidable [EG83].

1.4 Thesis

1.4.1 Contribution

The formal verification of cryptographic protocols is a very challenging area of research, in light of the problems it raises and the critical role protocols play in modern life.

Being able to prove that protocols are safe in a reasonable time and for a wide range of properties is vital for the development of new cryptographic protocols. When this work started only few automatic proof techniques were available for cryptographic protocols verification. One interesting approach among the user driven

approaches was the Genet and Klay's technique [GK00a]. Instead of trying to prove properties on concrete traces of protocols, they build abstract models of protocols' traces and then they check the safety of the protocols on the abstract systems. On the approximation of the reachable traces, it is possible to verify security properties such as no information got by the intruder or the participants communicate with the right person. The proof by approximation is very efficient as it allows us to only keep the information we are interesting in and then to quickly verify the desired properties. At the end of their article, they indicate that the approach can be automated. The challenge is to find an approximation that guarantees the termination of the computation, that does not require user interactions and that is suitable for secrecy and authentication verification. This thesis investigates the development of such approximation. Nevertheless, the proof by approximation is not perfect, and when the verification fails no conclusion can be drawn on the property. Another approach must then be used to check the property. This thesis also investigates the combination of the new approach with another verification technique also presents the ease with which the new approach can be integrated into an existing technique.

1.4.2 Outline of the Thesis

Chapter 2 reviews the literature on the formal verification of cryptographic protocols and motivates the work presented in this thesis. Chapter 3 introduces the basic definitions and details the method developed by Genet and Klay. Chapter 4 details the improvements made to the original method. It explains how the computation of the abstract model can become automatic. It also presents a combination with Paulson's technique [Pau98]. A prototype IS2TiF was developed to test the improvements made to [GK00a], Chapter 5 briefly introduces this tool. To validate the work done several protocol verifications were conducted, the results of which are presented in Chapter 5. Chapter 6 outlines the conclusions of this research and identifies future directions.

Chapter 2

Verification of Cryptographic Protocols: State of the Art

The formal verification of cryptographic protocols is a critical stage in the design and development of cryptographic protocols. A number of models and tools have been developed to formally verify cryptographic protocols over the last twenty years.

In this section, some of those models and tools are briefly introduced. Three types of approaches are distinguished, which correspond to "ways of explaining" to a user why a protocol is unflawed. This is done by:

- showing that what the user believes in (secrecy, authenticity) matches the reality;
- showing that messages or sequence of message that compromise the protocol cannot be found;
- describing the protocol as a number of little boxes. If the interactions between those boxes are free of mistakes the protocol is safe.

2.1 Belief approaches

One way to verify protocols is to reason about the beliefs of the participants involved in the communications.

One of the first approaches developed for the verification of cryptographic protocols was "the logic of authentication"[BAN89] (also called BAN logic). Michael Burrows, Martin Abadi and Roger Needham developed this logic in 1989. In this method, the verification is done by reasoning about the beliefs of the agents in the network and the evolution of these beliefs after each protocol step. An example of this reasoning would be: "If Paul has received a message encrypted with the key K, and he believes that only Alice and he share K, then he believes that Alice has sent the message".

To verify a protocol with this method, first the initial beliefs of all the protocol's actors are defined. Then after each protocol step, the receiving agent may be able to derive information previously unknown to him. With this new information and the logic inference rules, new beliefs are found by derivation. If the set of beliefs fits with the beliefs desired for the protocol, it is assumed that the protocol has been proven correct. Otherwise, a security flaw might have been discovered in the protocol. The BAN logic has found flaws and redundancies in several protocols [BAN89].

[Nes90] was one of the first to criticize the BAN logic. He created an example to prove the logic was flawed. In that example, the verification concluded that authentication could be established with a compromised key. Burrows, Abadi and Needham position was that the BAN logic was incapable of detecting an unauthorized release of information. Those criticisms were the first bases of new logics:

- GNY logic [GNY90]: this logic makes the distinction between what the participants believe in and what they possess. Thus it is possible to reason at a lower level than the BAN logic. Unfortunately, the 40 inference rules of this logic make it difficult to use.

- Abadi and Tuttle's logic [AT91]: gives a formal semantic of the BAN by improving the logic's syntax and inference rules. The modifications lead to a simpler logic. Nevertheless, [SvO94] proved that the logic was not sound.

- van Oorschot's logic [vO93]: this logic offers an extension of the BAN and the GNY to verify protocols with key agreement.

- AUTLOG logic [KG94]: this logic introduces a simulated spy who can detect information leaks. This logic contains 42 inference rules.

- SvO logic [SvO94]: this logic unifies the [BAN89], [GNY90], [AT91] and [vO93]. The logic contains the negation, makes the distinction between belief/possession and does not contain the idealisation step. It captures all the desirable features of the other logics without introducing new rules/axioms.

With the exception of the Abadi and Tuttle, and SvO logics, these logics are more complicated and more difficult to use than the BAN logic as they have more inference rules. The BAN logic only has 19 inference rules.

One can also refer to Kailar's logic [Kai95], and Kessler and Neumann's logic [KN98], that were used to verify e-commerce protocols. These logics are still based on the BAN logic, but they introduce the notion of accountability. Accountability is the property whereby the association of a unique originator with an object or action can be proved to a third party (i.e.: a party who is different from the originator and the prover). Kailar's logic [Kai96] has been used to verify two versions of Carnegie Mellon's Internet Billing Server protocol, the University of Southern California Information Science Institute's anonymous payment protocol and the SPX protocol. The SET [Gro96a](Secure Electronic Transaction) and the Payword protocols have been studied with Kessler and Neumann's logic [KN98].

Those logics are usually decidable. In [Mon99b], belief logics are automatically transformed in another logic on which a forward chaining search can be launched (the completeness and termination of the algorithm are given in the paper). The BAN logic was implemented in the theorem-provers SETHEO [Sch96] and EVES [CS96] (these two tools produce fully automatic proofs of protocols). Kindred generated automatic checkers for both Kailar's logic and AUTLOG logic using Revere [Kin99]. Nevertheless, if there is no tool yet developed for the logic required it can still be implemented in a theorem-prover like HOL [Bra96] (Higher Order Logic), SETHEO or any other suitable theorem-prover. Kessler and Neumann started to implement their logic in this tool, however the workload was very heavy, since SETHEO is actually not suited to this logic and the run-time for most of their proofs was so

long that they preferred to carry out proofs by hand.

An interesting work in that area was [BG00]. [BG00] introduces a model-checking approach that is able to deal with beliefs.

Each agent is seen as a process having beliefs about itself and other agents. They define a "view" as the evolution of a principal's beliefs over time. They extend the temporal logic CTL to use model-checking on belief formulae. To each view a language is associated to express properties about the process associates to the view. When authors consider the temporal evolution, formulae expressing beliefs are treated as atomic propositions.

CTL deals with finite state machines. Nevertheless, their model deals with an infinite number of views and belief atoms. So they have to create the MultiAgent Finite State Machine that is an extension of finite state machines for their model by:

- fixing the number of views. They only consider the views of an external observer and associate a finite state machine to each view.
- introducing the notion of *explicit belief atoms*; the only atoms which are explicitly represented in a finite state machine.

A MultiAgent Finite State Machine can then be seen as a set of finite state machines.

They had to extend also the notion of satisfiability in a MultiAgent Finite State Machine because of the *implicit belief atoms* (the atoms that are not explicit). The finite state machine satisfiability does not work with those atoms. The explicit belief atoms induce a "compatibility relation" between states of different views. They use this relation to express the satisfiability of implicit belief atoms (explicit atoms are used to study the veracity of implicit atoms).

The protocol's properties can then be checked under a temporal aspect or a belief aspect. This approach has been used to study the Andrew protocol [Sat89].

2.2 Trace approaches

Cryptographic protocols can also be verified by studying and reasoning about their traces. A trace is a sequence of *information* that is derived from the protocol steps or operations on messages. The nature of the information depends on what the verifier is interested in; it can be:

- a message, if the user only wishes to look at all messages sent during protocol runs.
- a message + a status of the knowledge of the intruder if the user wishes to follow the protocols and also to look at the information which the intruder learns after each protocol step.

2.2.1 Dolev Yao's model

Dolev and Yao [DY83] were among the first to offer an approach to verify cryptographic protocols. In their work, they specify a model of the intruder that has since been re-used in most other approaches. The intruder is in full control of the network. He can read, modify, delete messages and create fraudulent messages. Moreover, the other participants see him as a legitimate participant; he can then follow the protocol and establish valid communication with them. Initially, he does not know the initial secret information such as encryption keys belonging to honest agents. Since their intruder can intercept and replay any message, and can also create his own messages, Dolev and Yao consider any message sent as sent to the intruder and any message received as received by the intruder. Thus the network is a machine used by the intruder to generate messages (words). Messages follow rewrite rules, such as encryption and decryption with the same key cancel each other. The goal of the intruder is to find information that should be secret. If he succeeds then the protocol is flawed. Thus the verification of secrecy properties can be seen as a search problem in a term-rewriting system, which means proving that a certain message is not in the set of messages that can be generated (sent or received) by the intruder.

This model is quite limited as it considers only secrecy, and models only encryp-

tion and decryption of messages, and the addition and removal of agents.

In [DY83], two classes of protocols are also defined and studied:

- cascade protocols where only encryption and decryption operators are used by the participants;
- name-stamp protocols where the participants can encrypt and decrypt but can also append, delete and check names encrypted with the plaintext.

Sufficient and necessary conditions for two-party cascade protocols to be secure are provided in the paper [DY83]. A polynomial algorithm to decide if a two-party name-stamp protocol is secure is also introduced.

In [Mea92], Meadows extends the Dolev-Yao model to cover the intruder's ability to cheat a principal. Meadows' tool can find attacks where for example the intruder learns a word K and convinces an honest agent that K is a session key. Nevertheless this model was not able to deal with the freshness of the information.

This ability was added later in the NRL Protocol Analyser [Mea94]. In this tool, the protocols are specified as a set of transitions of state machines. The user then queries the program by entering words/information known by the intruder and values of local variables. The program takes each subset of the words and variables and for each transition rule uses a narrowing algorithm to find a complete set of substitutions that make the output of the rule reducible to that subset. The program returns the complete description of the states that may precede the specified states. The user can prove that a protocol is flawed by repeating this process. The idea is to start from insecure configuration (for example, one that violates a secrecy property), to look for a sequence of messages that lead to a valid message of the protocol (any message exchanged between two participants). If a sequence is found the protocol is flawed. The two main drawbacks of this approach are firstly, that the expertise of the user is important to find a flaw, and secondly that the computation may not terminate.

2.2.2 Paulson's model

Paulson [Pau98] developed a method based on the proof by induction on traces (messages sent). In this technique, the protocols are modeled by the set of all possible traces that they can generate. It is assumed in this method that there is an intruder or bad agent in the network. This intruder has access to all traces, can decrypt messages if he has caught the right decryption keys and finally, can build and send fraudulent messages if he has the correct encryption keys. To verify the protocol properties, we prove that each protocol step preserves the desired properties by induction on traces.

This method allows the verification of a large range of properties. But in this approach the secrecy and authenticity properties/theorems are very difficult to prove. The proofs require an experienced user to introduce the correct lemma at the correct time to ensure the proofs terminate. The proofs of the remaining properties (freshness, regularity, ...) are more simple and are generally similar for all protocols.

This method was implemented in the theorem-prover Isabelle [Pau94] and used to verify the Internet protocol TLS [Gro96b, Pau99] and Kerberos protocol [BP98a, BP97, BP98b] amongst others. The proofs of these protocols are available on the Isabelle website[1]. This technique has also been used to verify the SET protocol [Gro96a] in a project "Verifying E-Commerce Protocols at the University of Cambridge.

2.2.3 Rewriting techniques

[CDL+99] explains how a multiset rewrite rule model can be applied to protocol verification. Here, a system state is a finite set of ground terms: states of protocol agents, messages transmitted and information saved by the intruder. The protocol rules, modeled as rewrite rules, express how system states are updated (the left hand side of the rewrite rule is replaced by the right hand side of the rule). The model also contains rules to model the intruder's abilities. One interesting feature is the use of the existential quantifier in rules to express the freshness of information. The

[1] http://www.cl.cam.ac.uk/Research/HVG/Isabelle/

model is not a method to verify protocols rather a formalism to model them. One advantage of the formalism is that it allows the modeling of unbounded runs of a protocol.

This model has been incorporated into the CAPSL [DM00] specification language for authentication protocols. This language is translated to an intermediate language that can be exploited by the model-checker Maude [DMT98]. In Maude, the verification of the properties is carried out on an exhaustive search of the reachable states. If a state does not satisfy the property, the protocol is flawed and the trace of the attack is available. A depth-first algorithm is used to compute the reachable states. Thus it has the advantage of being able to verify a large range of protocols, however, it is limited to small protocols and a small number of participants. The approach was improved by typing the messages and by placing priority on rules to apply.

Jacquemard and al. [JRV00] also introduced an automatic tool that has been successfully tested on simple protocols [CJ97]. The protocol, the intruder's initial knowledge and the intruder's abilities described in an input file, are transformed into rewrite rules by their compiler CASRUL. For Jacquemard and al., protocols and intruders are rewrite rules executed on initial data by applying a variant of ac-narrowing [Hul80]. [JRV00] shows how these "narrowed" rules can be used by the theorem-prover daTac [Vig95]. A protocol execution is a sequence of terms describing the messages sent, the messages expected and the knowledge of the participants. When the prover arrives at an inconsistency (a term that models a violation of a property, for example the intruder knows information that should be secret) then the protocol is flawed. The verification is limited to small numbers of participants and of protocols' runs.

The rules generated by CASRUL have been used in theorem-provers for first-order logic, on-the-fly model-checking or SAT-based state exploration in the European Union Project AVISS[2] (Automated Verification of Infinite State Systems) [ABB+02].

[2]http://www.informatik.uni-freiburg.de/~softech/research/projects/aviss/

In [GK00a], tree automata are used to model the network (traces of the protocol + capabilities of the intruder) as well as the current intruder's knowledge. A term rewriting system is used to model the protocol steps and some intruder abilities. In this approach the properties are verified on an over-approximation of the reachable states of the network. At the end of the computation (if the computation terminates), the tree automaton contains all messages sent by the participant and all information the intruder has acquired during the runs. Properties are checked by checking that from all the information recognized by the automaton nothing violates the properties. As the set of information is an over-approximation of the concrete execution, if the property is verified on the over-approximation it is also verified on the concrete execution. Otherwise nothing can be said about the protocol. In this approach the authors try to prove that properties are verified and not to find an attack. The approach is not limited by the number of participants or the number of sessions. However, it does require user interactions to make the computation terminate.

2.2.4 Horn clauses

[Bla01, CLC03] use Horn clauses to model protocols. This approach is closely related to the rewriting approach. Predicates are used to model:

- basic information: honest agents, dishonest agents, agent names, numbers, nonces.
- a message: basic information is a message, a composition of messages is a message, keys are messages, etc.
- a trace is a possibly empty sequence of events, where an event is a message sent by an agent or a view of an agent's internal memory.

There are also predicates to model the intruder and any auxiliary functionality. Horn classes are used to model the abilities of the intruder, freshness, symmetry, etc with their predicates. The protocol steps and its properties are also modeled as Horn clauses. The advantage of such approach is that techniques that have

been developed to deal with Horn clauses over the years can be re-used for the computation of the reachable states of the system and the proof of its properties. The proofs are automatically done on the trace, however, to make the computation of a trace tractable, approximations are applied (those approximations are introduced in Section 5.3.3).

2.2.5 Automata model

In [Bol96], Bolignano introduced a method based on the idea of trustable and untrustable agents. A set of trustable agents and one intruder are defined. This intruder stores all the information exchanged between the agents, decrypts messages if he has the appropriate decryption keys, and builds and sends fraudulent messages if he has the appropriate encryption keys. Protocols are formalized as automata where each state is a n-tuple of the agents' current states, and the transitions are the protocols steps. The protocols properties are verified by induction on the automaton states.

Bolignano used the theorem-prover Coq [BBC+97] to implement this method, and presented an extension of his method for e-commerce protocols [Bol97]. Like [Pau98], this technique imposes no limits on the size of the messages and the number of sessions, but it requires user interactions.

Monniaux was the first to use tree automata to verify cryptographic protocols [Mon99a]. His idea was to model with the set of messages that the intruder can create after each protocol step using tree automata. For tractability reasons, a superset of the attacks is computed using an abstract interpretation technique [CC92]. The verification of protocol properties is carried out on the superset. If no attacks are found, then the properties are verified, however if an attack is found, these may not be any attack in reality. Moreover, even though the verification is fully automatic, the results are valid for a concrete model with bounded numbers of participants and sessions.

[GL00] extends Monniaux's idea by using an extension of tree automata that

integrates some deductive abilities (V-parameterized tree automata) and also by using some ideas from [Bol96] (protocol model) and [DMTY97] (honest agents can be seen as accomplices of intruders). This technique also uses abstract interpretation to build an abstract model of the intruder's knowledge. The result is a fully automatic technique. The main drawback of the approach is the exponential complexity of the exploration algorithm. Depending on the mode the protocol is running; mono-session or multi-session, and on the complexity of the protocol, the run time can go from few seconds to few minutes to return a result. A nice feature of this approach is that the computation does not stop after the first possible flaw. Rather, it explores all possible exchanges between the participants.

2.2.6 Strand Space model

[THG99] presents a new model for the verification of cryptographic protocols: Strand Spaces. A strand is a sequence of messages sent and received by an agent. A strand space is a set of strands (agents' strands and intruder's strands). A bundle consists of a number of strands hooked together, where one strand sends a message and another one receives that same message. A protocol will be correct when each bundle consists of one strand for each agent and each agent agrees on the participants, nonces, and session keys. Intruder strands are also included in a bundle, so long as they do not prevent honest agents agreeing on a secret, or from keeping their secrets.

Athena [SBP01] is a tool based on this model. It uses model-checking and theorem proving approaches. [SBP01] refines the strand spaces model and defines a logic to specify security properties. Their logic is a propositional logic in which strands are used as constants and bundles as variables. Secrecy and authentication properties are specified as well-formed formulae. If Athena computation terminates, it either provides a counterexample if the formula does not hold, or generates a proof of the correctness of the security. The verification is fully automatic and holds for an infinite number of protocol runs. The main drawback is that the computation might not terminate. Nevertheless, the termination can be forced by bounding the number of concurrent sessions of protocol and the size of the messages sent.

2.2.7 Murφ

Murφ [DDHY92] is a tool especially designed for protocol verification by state exploration. It has been successfully used to verify multiprocessor cache coherence protocols and multiprocessor memory models. The user models his protocol in the Murφ language and adds to this model the desired properties. The language is based on a collection of guarded commands (condition/action rules) to model the protocol steps. Then the possible traces (sequence of rules) are computed and the Murφ system checks if the rules of the traces satisfy the desired properties.

[MMS97] explains how Murφ can be used to verify cryptographic protocols. Firstly, they model the protocol, the intruder and the properties they intend to check for this protocol. Murφ is launched for exploration for four to five participants and three to five runs of the protocol. If a property is not satisfied, they have the trace that leads to the flaw.

2.2.8 Marrero, Clarke and Jha's model

In [MCJ97], protocols are modeled by a sequence of commands such as SEND, RECEIVE, NEWNONCE, etc. The principals and the intruder are also modeled by a sequence of commands. By interleaving these sequences, traces of the protocol are built. Then from these traces it is possible to investigate whether one leads to a configuration that violates the protocol's properties. This approach only works for a finite number of protocol runs.

2.2.9 Challenging model

[DMTY97] presents an automatic method to verify authentication properties using inference rules. After having extracted the role of each participant (information they are sending and receiving), their intruder will try to usurp identities. In addition to the usual abilities of the intruder (decryption and encryption of information with already known keys, composition and decomposition of message, etc.), the intruder also has some abilities regarding the protocol (For example if the intruder knows some information then, by encrypting and sending that information to an agent he

gets the cipher text of that information). For each role, an agent is expecting a particular message to launch the next one. If the intruder is able to make an agent launch all his messages then there is an authentication flaw in the protocol. The search is not guaranteed to terminate.

2.3 Process algebraic approaches

The final method of dealing with cryptographic protocols, reviewed within this section, considers protocols as processes with well-defined algebra. Each role (sender, receiver, and intruder) is seen as an independent process running together with other processes (roles). A process will be able to send, receive and create information, etc. With process algebra, systems can be interpreted using two main semantics:

- operational semantics; Operational semantics describes how protocols are performed. It shows the evolution of the protocol's configuration in time. A useful and well used paradigm to describe protocols from an operational point of view is the state transition systems. It gives transitions between states where states are process descriptions. Thus starting from an initial configuration of the processes, the set of the possible configurations can be computed and the properties can be checked on those configurations.

- denotational semantics; Denotational semantics describes the meaning of a protocol as a mathematical object in some domain. The protocol P is mapped by a valuation function F into its meaning (denotation) D:

$$F(P) = D.$$

From the process algebraic description the following information is built:

1. syntactic domain which is the set of mathematical objects,
2. semantic functions which map object from the process algebra into object in the semantic domain,
3. semantic equations which specify how functions act on each process.

Then protocols are proven safe if their meaning verify properties.

The language of Communicating Sequential Processes (CSP) [Hoa85] has been used for the verification of cryptographic protocols. This language can describe interactions of processes in a system where they are interacting via message passing.

Due to the ambiguity of the informal specifications, passing from a protocol specification to a CSP model is difficult [Sch98] and requires design decisions to be taken. An attempt to simplify this process was the work of Roscoe [Ros95] where a CSP protocol model is presented. The model consists of a communication medium with two channels that link it to each participant. One channel is for messages sent and the other channel is for those received. The abilities of the intruder are modeled by adding extra-channels. In order to simplify the transformation specification to CSP model, Lowe implemented Casper [Low97b], a protocol compiler, which produces CSP description from protocol specifications.

Once a CSP model is available, the Failures Divergences Refinement (FDR) model checker is used to discover attacks on the protocol. One of the main drawback of model-checkers generally, and FDR is no exception, is that they are inefficient on systems with infinite number of states to explore. To restrict the behaviour of the model, limitations on the numbers of participants, nonces and protocol runs are applied. Thus the model-checker guarantees the safety of the protocol for the "reduced" model but it does not mean that a bigger model will be safe. Research has been carried on CSP and FDR to achieve more complete results. [BLR00] applied *data independence* techniques which allow the allocation of an infinite number of fresh information but still maintain a finite number of values in FDR for unbounded runs. To deal with an infinite number of nonces, an extra-process must be added to the model. This process delivers fresh nonces to participants when required and checks the mapping of the nonces onto a finite set of values. A subset of those values is allocated to the fresh nonces used by trustable participants. The remaining values are mapped onto by all the nonces known to the intruder and those redundant in the current session. To work, the technique requires messages to be typed. One drawback of the approach is that every agent is limited to one session at a time;

otherwise the number of states to explore will increase drastically.

In [Low99], Lowe presents sufficient conditions on protocols and their environments, such that, if there is no secrecy breach on a small model of the protocol then the protocol is secure. Thus, there is no attack possible on any bigger system. His small model restricts the role of the agents, the intruder's initial knowledge and the data type of the messages. Conditions are placed on the requirements of distinct text values of encrypted components, the ability to determine the identities running the protocol from the messages and the exclusion of temporary secrets. This analysis is limited to secrecy properties and Lowe admitted there was a small risk that a flaw could be found when in fact the protocol was safe.

Despite their limitations, CSP and FDR are very efficient for the verification of cryptographic protocols. In [Low96], Lowe used FDR model-checker to find his flaw in the Needham-Schroeder public key authentication protocol [NS78]. A combination of Casper and FDR has been proven to be a very powerful tool for the verification of cryptographic protocols. In [DNL99], this approach was used to verify 50 protocols of [CJ97]. The results showed that they were unable to verify one protocol, failed to rediscover attacks on five protocols, found new attacks on 10 protocols assumed to be secure and new attacks on 6 flawed protocols.

In [CJM00] introduced a tool, known as Brutus, which was developed to simplify the protocol designers work. Here the protocol is modeled as an asynchronous composition of a set of named communicating processes which model honest agents and the intruder. In their model, all messages sent by honest agents are caught by the intruder and the intruder sends all the messages received by honest agents. To make the model finite, the amount of time a participant may execute the protocol is limited. Each attempt is a session and for each session an agent can play the role of initiator or responder. To complete this model, an intruder is added and they are able to explore the different states of their system. The protocol properties are expressed and verified with a first-order logic that includes a past-time modal operator.

The *spi-calculus* [AG98] is another process algebra developed for the cryptographic protocol verification. It extends the *π-calculus* with cryptographic primitives. In this approach, each actor is modeled as a process, and the protocol is an instance of those processes running in parallel. In this framework it is possible to express secrecy and authenticity properties. In order to verify whether a property is satisfied, the equivalence[3] between two instances of the protocol must be checked. If *Inst(M)* describes an instance of the whole protocol parameterized by the message M and if *F(x)* is an instantiation of the abstraction F on x, then:

> **Authentication property:** *Inst(M)* is equivalent to $Inst_{spec}(M)$, for any M; $Inst_{spec}$ is a magical version of *Inst* where processes react as if they have received the correct information.
>
> **Secrecy property:** *Inst(M)* is equivalent to *Inst(M')* if *F(M)* is equivalent to *F(M')*, for any M and M'.

In the spi-calculus, there is no explicit definition of an intruder's abilities. The intruder is assumed to be able to carry out any of the actions defined in the language, and carries out attacks on a protocol using only these actions. Therefore, the verifier must be wary that this could result in missing particular attacks on the protocol due to the limitations of the calculus. The language has been used to verify small protocols [AG98], and the proofs were carried out by hand. Because of the use of the *quantification over all possible contexts* in their definition, the implementation of an automatic tool for this approach is difficult. Nevertheless, work to develop tools and techniques based on this model can be found [Aba99, GLL00, FA01, Azi03].

In [Aba99], Abadi explains how to verify secrecy properties by typechecking the protocols. He distinguishes three possible types for the data sent: *Public*, *Secret* and *Any*. *Public* data can be known by anyone. *Secret* data must be kept secret. *Any* data is an arbitrary type but it should not be leaked as *Any* data could be of *Secret* type. On those three types, he builds a set of typing rules for the spi-calculus.

[3]Two processes P and Q are equivalent if the behaviours of processes P and Q are indistinguishable from each other. A third process cannot distinguish running in parallel with P from running in parallel with Q.

He also guarantees that if the protocol typechecks then the secrecy of messages is protected. He shows that the proofs are more simple to do with typechecking than without. Moreover, the rules specified are neither necessary nor sufficient for the security of protocols. Not necessary as they are incomplete, and not sufficient as they take only secrecy issues into account.

In [GLL00], Gnesi et al. modify the syntax of the spi-calculus (by removing mobility and by embedding the "let" and "case" constructs into the output and input primitives respectively) and define a semantics for their language based on labeled transition systems. They then show how security properties can be specified with the Brutus logic [CJM00]. To make their transition systems finite, they used results of previous work in this domain [MMS97, HLS03] (such as typing input messages).

In [FA01], symbolic techniques for studying the traces of particular cryptographic protocols (incorporecting shared key encryption/decryption that use arbitrary messages as keys) are presented. They use a dialect close to the spi-calculus to describe the protocols. The symbolic computation leads to a finite representation of models embodying the interactions between the agents and the intruder. These models can then be used to analyze protocol properties.

[Azi03] introduces a denotational semantics for the pi-calculus and the spi-calculus. Using an abstract analysis approach on his models, he explains the verification of the secrecy and authentication properties. Automatic tools for each of the calculus are also presented in this thesis. The results are guaranteed for unbounded numbers of sessions and participants.

2.4 Summary

As is demonstrated in this chapter, large numbers of models and tools are available for the verification of cryptographic protocols. Choosing the most appropriate model is difficult. Moreover, the properties verified might differ from one model to another. Nevertheless these approaches can be compared on the basis of:

- whether they try to find attacks or to prove that properties are satisfied;

- whether an automatic tool is available to do the verification.

Table 2.1 presents the comparison of the methods studied in of this section. From this table, two conclusions can be drawn:

- more "attack search" techniques were available;
- "attack search" techniques were more accessible as automatic verification tools were available.

Techniques	**Proof**	**Attack**	**Automatic**
BAN [BAN89]	√		√
Dolev Yao [DY83]		√	√
Meadows [Mea92]		√	√
Paulson [Pau98]	√		
Maude [DMT98]		√	√
CASRUL [JRV00]		√	√
Genet and Klay [GK00a]	√		
Bolignano [Bol96]	√		
Monniaux [Mon99a]		√	√
Murφ [MMS97]		√	√
Marrero and al. [MCJ97]		√	√
Debbadi and al. [DMTY97]		√	√
Casper+FDR [Low97b]		√	√
spi-calculus [AG98]	√		

Table 2.1: Starting point of the thesis

Verifications using proof techniques are very powerful as their results are guaranteed for an infinite number of sessions and an infinite number of participants. But such methods require many user interactions to do the proofs.

On the other hand, the "attack search" techniques look for attacks on small models of the protocol (bounded sessions, bounded number of agents, etc.) and offer no guarantees for the whole protocol, if no attack is found. Nevertheless the search can be done automatically.

To make the problem of the search for attacks tractable and to develop automatic tools, the following two approaches have been used.

The first approach consists of fixing the parameters that can be infinite to reduce the size of the system. For example, the messages exchanged can be typed [MMS97, HLS03] to fix their size. The number of agents in the network can also be bounded. [CLC03] proved that two agents are sufficient for the analysis of security properties of cryptographic protocols when the protocols allow an agent to talk to himself. If the protocol does not allow "agents to talk to themselves" and there is an attack involving n agents, then there is an attack involving at most $k + 1$ agents (k is the number of roles that an agent can play). With this approach if a flaw is found on the small model then the larger model, from which the smaller was derived, is also flawed. The lack of flaws on the small model does not necessarily mean that the original model is safe.

The second approach is to build an abstraction-based approximation of the concrete model [Cou01]. Here the idea is to build a finite model by compounding information regarding the system properties to be checked or by making abstraction of irrelevant information to the properties to be checked. With this approach, if a flaw is not found in the abstract model, then the concrete model is safe. Flaws in the abstract model do not necessarily mean that the concrete model is flawed.

To illustrate these points consider Figure 2.1, 2.2 and 2.3. Figure 2.1 models the system that must be verified. Figure 2.2 gives an idea of the system to verify when the number of agents are bounded [CLC03]. In Figure 2.3, Agent 2 to Agent n are compounded into an Agent R. The communications between these agents are then modeled by a transition from Agent R to himself; that is the approximation chosen by [GK00a].

With the abstraction technique, some proofs of cryptographic protocols have been carried out since this research work started [Bla01, CLC03, BLP02, Azi03]. Moreover by choosing this technique, the tool implemented should be easily combined with another approach in order to get a result when the proof fails (i.e. the property is not satisfied on the abstract model).

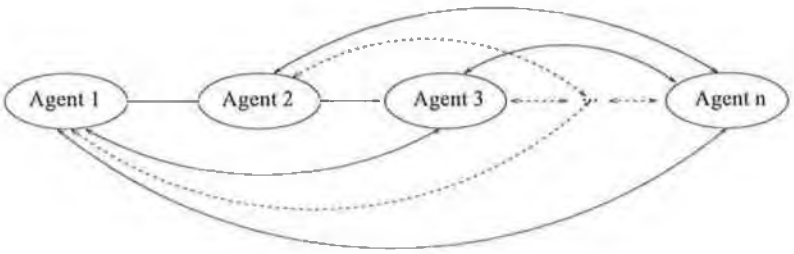


Figure 2.1: Infinite system

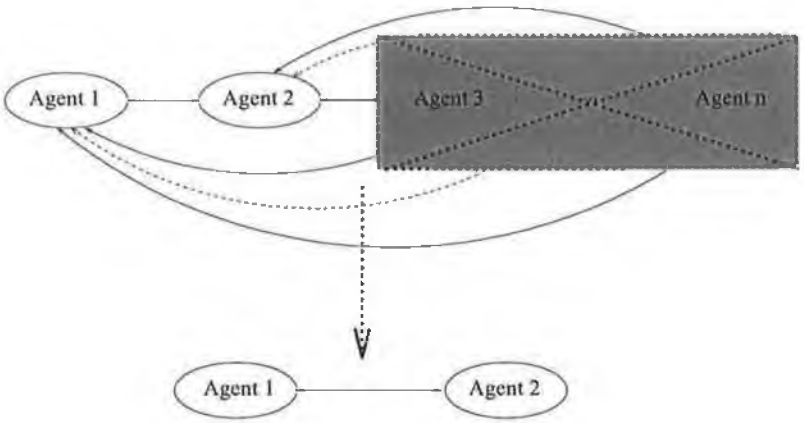


Figure 2.2: Fixing the unbound parameters

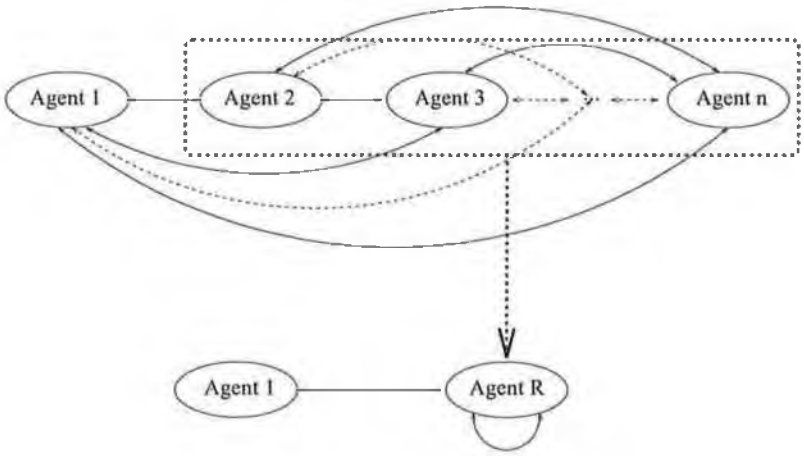


Figure 2.3: Compounding together information

Finally, the approaches introduced in this chapter are mainly "specialized" for authentication and secrecy properties. However, other properties may also be studied as indicated in Chapter 1.

Thus to summarize a good verification technique should:

- automatically prove properties;
- be easily combined with other approaches (to get a better feedback on the protocol safety or to verify a wider range of properties)

2.5 Conclusion

As only few automatic proof approaches are available, we decided to work on an automatic proof approach. The Genet and Klay's technique looks like a nice starting point as it can be automated and it seems to be easily combinable with another approach. As it will be seen in Chapter 5, other people also decided to work on developing automatic proof tools to verify cryptographic protocols. Nevertheless, it was not the only active fields over the last years, two other fields were:

- extending the power of the intruder used during the verification [Mea00, CLT03, CRZ05],
- extending the verification to other protocol properties: freshness [NRV04], fairness [BDD+05, TVV05] and anonymity [GHvRP05].

Chapter 3

Genet and Klay's approach

This chapter starts by introducing some basic notation and definitions required to understand the following overview of rewriting systems and tree automata. Following this, the Genet's concept [Gen98a], later re-used by Genet and Klay [GK00a] to verify cryptographic protocols, is introduced. The final part of this chapter explains how Genet and Klay verify protocols, and highlights the pros and cons of the technique.

3.1 Definitions

An *alphabet* $\mathcal{F}$ is a finite set of elements of the form *f:i* where *f* is a symbol and $i = ar(f)$ with ar an arity function from $\mathcal{F}$ to N. Symbols of arity zero are called constants.

Let $\mathcal{X}$ be a set of constants that are different from the constants of an alphabet $\mathcal{F}$, then the set $\mathcal{T}(\mathcal{F}, \mathcal{X})$ of *terms* is defined as follows:

1. $\mathcal{X} \subseteq \mathcal{T}(\mathcal{F}, \mathcal{X})$ and
2. $\{x | x \in \mathcal{F} and ar(x) = 0\} \subseteq \mathcal{T}(\mathcal{F}, \mathcal{X})$ and
3. $\{f(t_1, \ldots, t_n) | f \in \mathcal{F} and ar(x) > 0 and all\, t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F}, \mathcal{X})\} \subseteq \mathcal{T}(\mathcal{F}, \mathcal{X})$

Constants of $\mathcal{X}$ used in the terms are called variables and $Var(s)$ denotes the set of variables of the term $s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$. Terms without variables are called ground

terms. The set of ground terms is written $\mathcal{T}(\mathcal{F})$. Finally, a term is linear if none of its variables occurs more than once.

Terms can also be seen as trees labeled by elements of $\mathcal{F} \cup \mathcal{X}$. For example, the term $f(t_1, t_2, t_3)$ can be represented by the tree:

It is possible to navigate through terms. We need the set $\mathcal{P}os(s)$ of positions in the term s in $\mathcal{T}(\mathcal{F}, \mathcal{X})$. $\mathcal{P}os(s) = \{\epsilon\} \cup \mathcal{P}os_i(s)$ with ϵ the top-most position in a term[1] and $\mathcal{P}os_i(s)$ inductively defined as follows:

1. $\mathcal{P}os_i(s) = \emptyset$ if $s \in \mathcal{X}$ or $s \in \{x | x \in \mathcal{F} and ar(x) = 0\}$ and
2. $\mathcal{P}os_i(f(t_1, \ldots, t_n)) = \bigcup_{i=1}^{n}(\{i\} \cup \{ip | \mathcal{P}os_i(t_i) \neq \emptyset \text{ and } p \in \mathcal{P}os_i(t_i)\})$ if $f \in \mathcal{F}$, $ar(f) = n$ and $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F}, \mathcal{X})$.

Now that all the positions in s are known, it is easy to navigate trough the subterms of the term s; $s|_p$ denotes the subterm of s at the position $p \in \mathcal{P}os(s)$. It is also possible to replace the subterm $s|_p$ by the term t; the notation is $s[t]_p$.

It is also possible to access the symbol at the ϵ position using $\mathcal{R}oot(s)$ which denotes the symbol at position ϵ in s.

$\mathcal{P}os(s)$ gives all the positions in s, moreover, only the positions of the symbols of $\mathcal{F}$ can be required. Thus this set for the term s, $\mathcal{P}os_{\mathcal{F}}(s)$, is defined as follow:

- $\mathcal{P}os_{\mathcal{F}}(s)$=$\{p \in \mathcal{P}os(s) \mid p \neq \epsilon \wedge \mathcal{R}oot(s|_p) : i \in \mathcal{F}$ *with* $i = ar(\mathcal{R}oot(s|_p))\}$.

Example 1 *Let* $\mathcal{F}$=$\{mesg{:}3,\ A{:}0,\ B{:}0\}$ *where* $ar(mesg) = 3$ *and* $ar(A) = ar(B) = 0$ *and* $\mathcal{X}$=$\{x,\ y,\ z\}$, *then:*

- $\mathcal{T}(\mathcal{F}, \mathcal{X})$=$\{mesg(x,\ y,\ z),\ mesg(x,\ B,\ z),\ ...\}$
- $\mathcal{T}(\mathcal{F})$=$\{mesg(A,\ B,\ A),\ mesg(B,\ A,\ B),\ ...\}$

[1] if $s = f(t_1, \ldots, t_n)$ then the top-most position is the position of f.

- $Var(mesg(x, y, z))$={x, y, z}
- *if* $s = mesg(A, B, x)$ *then*
  - $\mathcal{P}os(s) = \{\epsilon, 1, 2, 3\}$
  - $\mathcal{P}os_{\mathcal{F}}(s) = \{1, 2\}$
  - $\mathcal{R}oot(s) = mesg = \mathcal{R}oot(s|_\epsilon)$
  - $t_1 = A$ *and* $s[B]_1 = mesg(B, B, x)$

To conclude this section, the definition of a substitution is given. A substitution σ is an endomorphism from $\mathcal{T}(\mathcal{F}, \mathcal{X})$ to $\mathcal{T}(\mathcal{F}, \mathcal{X})$. $s\sigma$ is the result of the substitution σ on the term $s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$.

Example 2 *Let* $\mathcal{F}$={*mesg:3, A:0, B:0*}, σ={x=A, y=B, z=A} *and* s=*mesg*(x, y, z);

$s\sigma$=*mesg*(A, B, A).

3.1.1 Term Rewriting Systems

Equational reasoning [AKN89a, AKN89b, Höl89] is an important concept in symbolic algebra, automated theorem proving and program verification. Reasoning with equations can prove the validity of equations and can be used to solve equations.

Dershowitz and Jouannaud [DJ90] give the following definition of rewriting systems. Rewrite systems are directed equations (*rewrite rules*) used to compute new equations by repeatedly replacing subterms of a given formula with equal terms until the simplest form possible is obtained. Since the computational power of rewrite systems is as strong as Turing machines [Tur36], and as they are easy to understand, rewriting theory is a very efficient form of equational reasoning.

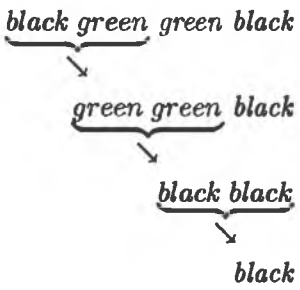
To illustrate the ideas behind this theory, a simple example is considered [DP01] (called by some the Grecian urn problem and by others the coffee can problem). We have *black* and *green* beans in an urn. We remove two beans at a time. If they have the same colour, a *black* bean is added to the urn. If they are different, then a *green*

bean is added. The process is repeated until no more actions can be performed. Now, it is required to know if the colour of the last bean in the urn is predetermined and, if so, what is it?

The urn can be seen as a sequence of *black* and *green* beans (for example: *black greengreen black*). The removal operation is expressed by four rules:

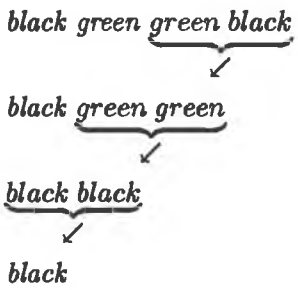
1. *black black* $\longrightarrow$ *black*
2. *green green* $\longrightarrow$ *black*
3. *black green* $\longrightarrow$ *green*
4. *green black* $\longrightarrow$ *green*

For example, a possible sequence of moves is rule 3, rule 2 and rule 1 (the rule is applied to the head of the sequence):



A first remark is that the number of beans in the urn decreases after each operation so the rewrite system terminates.

Taking the same initial urn but changing the sequence of moves: rule 4, rule 2 and rule 1; and applying the rule to the end of the sequence, it yields:



A second remark is that no matter which sequence of rules is used, the same result is achieved.

In general the result is that the colour of the last bean is predetermined. If an even number of *green* beans are in the urn then the last bean will be *black*, else the last bean will be *green*.

Term Rewriting Systems (TRS) are a type of computational model based on *rewriting*. The following definitions, built on each other, introduce the basic information required to understand the approach of Genet ([DP01] is an interesting reference on rewriting systems in general).

Definition 1 *(Term rewriting system [Gen98a]) A* **term rewriting system** $\mathcal{R}$ *is a set of rewrite rules* $l \to r$, *where* $l,\ r \in \mathcal{T}(\mathcal{F},\mathcal{X})$, $l \notin \mathcal{X}$, *and* $Var(r) \subseteq Var(l)$.

The relation $\to_{\mathcal{R}}$ *means that for any* $s,t \in \mathcal{T}(\mathcal{F},\mathcal{X})$ *we have* $s \to_{\mathcal{R}} t$ *if there exists a rule* $l \to r$ *in* $\mathcal{R}$, *a position* $p \in Pos(s)$, *where* $Pos(s)$ *is the set of positions in* s, *and a substitution* σ *such that* $l\sigma = s|_p$ *and* $t = s[r\sigma]_p$.

Example 3 *Let* $\mathcal{R}$=$\{f(a,b) \to g(a,b,a),\ g(a,b,a) \to g(b,a,b),\ g(a,b,a) \to t(a)\}$, $A \in \mathcal{T}(\mathcal{F},\mathcal{X})$, $B \in \mathcal{T}(\mathcal{F},\mathcal{X})$ *and* $\sigma = \{a = A,\ b = B\}$.

We have $f(A,B) \to_{\mathcal{R}} g(A,B,A)$ *as:*

1. *we have the rewrite rule* $f(a,b) \to g(a,b,a) \in \mathcal{R}$,
2. *applying the substitution* σ *to* $f(a,b)$ *gives us* $f(A,B)$; $f(a,b)\sigma = f(A,B)$,
3. *substituting the term* $f(A,B)$ *by the term obtained by applying* σ *to* $g(a,b,a)$ *gives us* $g(A,B,A)$; $f(A,B)[g(a,b,a)\sigma] = g(A,B,A)$.

Some notation and properties can be defined on $\to_{\mathcal{R}}$:

- $\to^+_{\mathcal{R}}$ denotes the transitive closure of $\to_{\mathcal{R}}$.
- $\to^*_{\mathcal{R}}$ denotes the reflexive transitive closure $\to_{\mathcal{R}}$. The set that contains all the terms reachable from a set E of ground terms using $\to^*_{\mathcal{R}}$ is called $\mathcal{R}$-descendants. This set is denoted by $\mathcal{R}^*(E)$.

- Two terms s and t are joinable, written $s \downarrow_{\mathcal{R}} t$, if there exists a term v such that $s \rightarrow_{\mathcal{R}}^{*} v$ and $t \rightarrow_{\mathcal{R}}^{*} v$.
- A term s is *reducible* when there exists a term t such that $s \rightarrow_{\mathcal{R}} t$.
- A term s is irreducible or in normal form when it is impossible to find a term t such that $s \rightarrow_{\mathcal{R}} t$. The set of normal terms that are reachable from E is called $\mathcal{R}$-normal forms and is denoted by $\mathcal{R}^{!}(E)$. At last, $IRR(\mathcal{R})$ is the set of irreducible terms by the TRS $\mathcal{R}$.

Definition 2 *($\mathcal{R}$-descendants and $\mathcal{R}$-normal forms [Gen98a]) Let $\mathcal{R}$ be a TRS and E a set of terms.*

$\mathcal{R}^{*}(E) = \{t \in \mathcal{T}(\mathcal{F}) \mid \exists s \in E \,.\, s \rightarrow_{\mathcal{R}}^{*} t\}$.

$\mathcal{R}^{!}(E) = \{t \in \mathcal{T}(\mathcal{F}) \mid \exists s \in E \,.\, s \rightarrow_{\mathcal{R}}^{*} t \wedge t \in IRR(\mathcal{R})\}$ *where $IRR(\mathcal{R})$ is the set of irreducible terms by the TRS $\mathcal{R}$.*

Example 4 *Let $\mathcal{R}$=$\{f(a,b) \rightarrow g(a,b,a),\ g(a,b,a) \rightarrow g(b,a,b),\ g(a,b,a) \rightarrow t(a)\}$ and E=$\{f(A,B),\ f(B,A)\}$:*

- *$IRR(\mathcal{R})$=$\{t(A),\ t(B),\ t(a)\}$,*
- *the term $g(B,A,B) \in \mathcal{R}^{*}(E)$ as $f(A,B) \rightarrow_{\mathcal{R}}^{*} g(B,A,B)$,*
- *the term $t(A) \in \mathcal{R}^{!}(E)$ as $f(A,B) \rightarrow_{\mathcal{R}}^{*} t(A)$ and $t(A) \in IRR(\mathcal{R})$.*

It is possible to link $IRR(\mathcal{R})$, $\mathcal{R}^{*}(E)$ and $\mathcal{R}^{!}(E)$: Proposition 1.

Proposition 1 $\mathcal{R}^{!}(E) = \mathcal{R}^{*}(E) \cap IRR(\mathcal{R})$

If the left hand side of the rewrite rule $l \rightarrow r$ (resp. right-hand side) contains only one occurrence of each variable, the rule $l \rightarrow r$ is left-linear (resp. right-linear). A rewrite rule is linear if it is right and left linear.

Definition 3 *Let $\mathcal{R}$ be a TRS defined on $\mathcal{T}(\mathcal{F}, \mathcal{X})$. $\mathcal{R}$ is* **left-linear** *(resp. right-linear, linear) if every rewrite rule of $\mathcal{R}$ is left-linear (resp. right-linear, linear).*

Example 5 *Let $\mathcal{R}$={f(a,b) → g(a,b,a), g(a,b,c) → g(b,a,b), g(a,b,c) → t(a)}, this TRS is left linear as all the rules of $\mathcal{R}$ are left linear but it is not linear as the right-hand side of the rewrite rule g(a,b,c) → g(b,a,b) contains 2 occurrences of the variable b. On the other hand, $\mathcal{R}$={f(a,b) → g(b,a), g(a,b) → t(b)} is linear as all the rules of $\mathcal{R}$ are linear.*

Two important properties that a TRS may possess are termination and confluence.

3.1.1.1 Termination

Definition 4 *(Termination [Gen98a]) A TRS $\mathcal{R}$ is terminating if there are no infinite derivations of term $s_i \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ $(i \in \mathbb{N})$ such that $s_0 \to_{\mathcal{R}} s_1 \to_{\mathcal{R}} s_2 \to_{\mathcal{R}} \ldots$.*

The rules in Grecian urn problem decrease the size of the system after each step so it always terminates. Proving or disproving the termination of TRS is an undecidable [HL78]. Lankford [Lan79] explained of the termination of TRS and orders are linked.

Before the concepts of Lankford are introduced, some basic notions on binary relations are given. A binary relation R on a set E is reflexive if and only if for all a in E then aRa. R is antisymmetric if and only if for all a, b in E it holds that if aRb and bRa then $a = b$. R is transitive if, and only if for all a, b and c in E it holds that if aRb and bRc then aRc. R is a partial-ordering and is written $>$ if R is transitive, antisymmetric and reflexive.

Definition 5 *(Well-founded Ordering [Gen98a]) A partial ordering $>$ is well-founded on the set E if there are no infinite decreasing series.*

Example 6 *The partial ordering $>$ on $\mathbb{N}$ is well-founded as the decreasing series are bounded by 0.*

Proposition 2 *([MN70]) A TRS $\mathcal{R}$ defined on $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is terminating if and only if there exists a well-founded partial ordering $>$ on $\mathcal{T}(\mathcal{F}, \mathcal{X})$ such that:*

$$\forall s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X}).s \rightarrow_{\mathcal{R}} t \Rightarrow s > t.$$

The number of pairs s, t that satisfy the condition are usually infinite, so other properties are usually used. These properties prove the termination by looking at orders that can be checked on the rewrite rule instead of all the rewriting steps.

Three approaches to prove the termination can be distinguished:

- semantical methods: the idea is to look for a marker (the size of the term used in the rewrite rules, a symbol used in all the rules, etc.) and to check that the marker decreases after each rewriting step.

  One of the most commonly used techniques is to prove that the order for the set on which the marker is defined is well founded and monotone.

- syntactical methods: the idea is to find well-founded orders that guarantee that each rewriting rule is decreasing with respect to the orders.

  Orders here are defined inductively on the terms. The main two orders here are the recursive path ordering [Der82] and the Knuth-Bendix order [KB70].

- transformational methods: the idea is to transform the current TRS into another TRS on which the termination is easier to prove and which guarantees the termination of the original TRS.

More information about the different methods can be found in [Zan97].

3.1.1.2 Confluence

Definition 6 *(Confluence [DP01]) A TRS is confluent if for all terms s, t, $u \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ such that $s \rightarrow_{\mathcal{R}}^* t$ and $s \rightarrow_{\mathcal{R}}^* u$, then $t \downarrow_{\mathcal{R}} u$.*

Like the case of the termination property, this property is generally undecidable. Confluence becomes decidable, however, if the TRS is terminating, thanks to the "critical pair" (cf. Definition 7) computation and to Newman's lemma (cf. Lemma 1).

Definition 7 *(Critical pair [DP01]) Let $\mathcal{R}$ be a TRS and $l \to r$ and $g \to d$ be two rules of $\mathcal{R}$ with distinct variables. If there exists a position $p \in Pos_{\mathcal{F}}(l)$ and a substitution σ such that $l|_p\sigma = g\sigma$ then $(l[d]_p\sigma, r\sigma)$ is a critical pair of $\mathcal{R}$.*

Example 7 *Let $\mathcal{R} = \{e.x \to x,\ I(i).i \to e,\ (x.y).z \to x.(y.z)\}$ be a TRS.*

If the substitution σ=$\{x = I(i),\ y = i,\ z = z\}$ is applied on $(x.y).z$ the resulting term is $(I(i).i).z$. Two reductions can then be applied on this term $(I(i).i).z \to I(i).(i.z)$ (using $(x.y).z \to x.(y.z)$; the $l \to r$ of Definition 7) and $(I(i).i).z \to e.z$ (using $I(i).i \to e$; the $g \to d$ of Definition 7), thus the critical pair $(e.z,\ I(i).(i.z))$ is deduced.

Lemma 1 *([New42]) Let $\mathcal{R}$ be a terminating TRS. $\mathcal{R}$ is confluent if and only if for all critical pairs (p,q) of $\mathcal{R}$ there exists $w \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ such that $p \to_{\mathcal{R}}^* w$ and that $q \to_{\mathcal{R}}^* w$.*

If a TRS is not confluent, it is possible to make it become confluent using the Knuth-Bendix completion algorithm [KB70]. This algorithm transforms a finite set of identities into a terminating confluent TRS, in Appendix A an example of how it works can be found.

If the algorithm is initialized with a non-confluent TRS $\mathcal{R}$ and a reduction ordering $>$ that guarantees the termination of $\mathcal{R}$, then a confluent TRS $\mathcal{R}'$ is built such that $\to_{\mathcal{R}} \subseteq \to_{\mathcal{R}'}$. For each critical pair (p, q), the completion will add the rule $p \to q$ if $p > q$ and $q \to p$ if $q > p$ to guarantee the confluence of the system. Again more details about the confluence property can be found in [DP01] and [Zan97].

3.1.2 Tree Automata

Automata theory is an essential part of theoretical computer science. It can be used in many fields: natural languages, modeling biological phenomena, programming languages, cryptography, computer graphics, etc.

The main advantage of this theory is that with a finite alphabet and a finite set of grammar rules, an infinite set of words (terms) can be recognized. This set is called a language. The language corresponding to the automaton $\mathcal{A}$ is noted $\mathcal{L}(\mathcal{A})$.

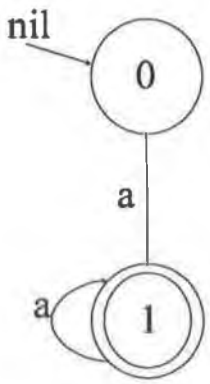


Figure 3.1: Basic automaton

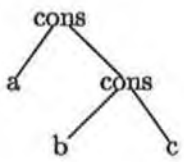
For example given the alphabet $\{a, nil\}$ and the grammar $\{nilf \rightarrow 0, 0a \rightarrow 1, 1a \rightarrow 1\}$ where 0 is the initial state and 1 is the final state, then it is possible to recognize all the words in the $\{nila, nilaa, nilaaa, ...\}$. This concept is illustrated in Figure 3.1.

This diagram gives the graphical form of this automaton. In this figure, the state 0 is reached by the transition labeled by *nil*. Then to go from the state 0 to the state 1 (final state), there is one transition labeled by *a*. With those transitions, the word *nila* is built. Then another transition labeled by *a* is looping on the state 1, with this transition all subsequent words in the language can be built.

By looping once, the word *nilaa* (transition to 0, then transition from 0 to 1 plus transition from 1 to 1) is built, by looping twice the word *nilaaa*, etc.

In this thesis, a special form of automata is used: tree automata. These are called "tree" automata because of the graphical view that can be made of the terms recognized by those automata.

For example, the term *cons(a, cons(b, c))* can be viewed as the tree:



The symbol *cons* at the top is called the root, and *a*, *b* and *c* are called leaves. To

recognize this term, the automaton can either start from the leaves to go to the root or it can start from the root to go to the leaves. It is then possible to distinguish two categories of tree automata depending on how the automata recognize the terms:

- bottom-up automata: the automata start by recognizing the leaves and then move up to the root. Figure 3.2 gives an example of how the term *cons(a, cons(b, c))* would be recognized by a bottom-up automaton. The automaton starts (transition $\rightarrow_1$) by recognizing the leaf *a* (when a subterm is recognized it becomes green), then the leaf *b* (transition $\rightarrow_2$), and so on until it reaches the root.

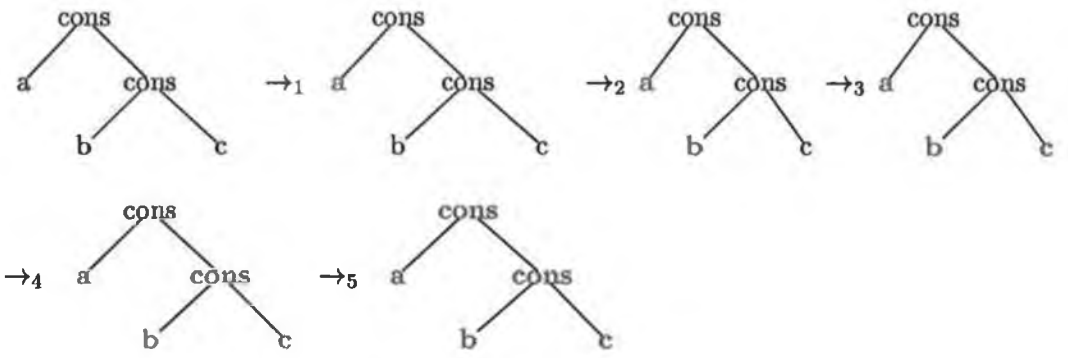


Figure 3.2: Bottom-up process

- top-down automata: the automata start by recognizing the root and then go down to the leaves. Figure 3.3 gives an example of how the term *cons(a, cons(b, c))* would be recognized by a top-down automaton. The automaton starts (transition $\rightarrow_1$) by recognizing the root *cons* (when a subterm is recognized it becomes green), then the leaf *a* (transition $\rightarrow_2$), and so on until it reaches the last leaf *c*.

In this thesis, a subset of the bottom-up tree automata, the bottom-up non-deterministic finite tree automata, is considered. In this section, definitions and properties of the bottom-up non-deterministic finite tree automata are revisited. More information about tree automata can be found in [CDG+98, GS84].

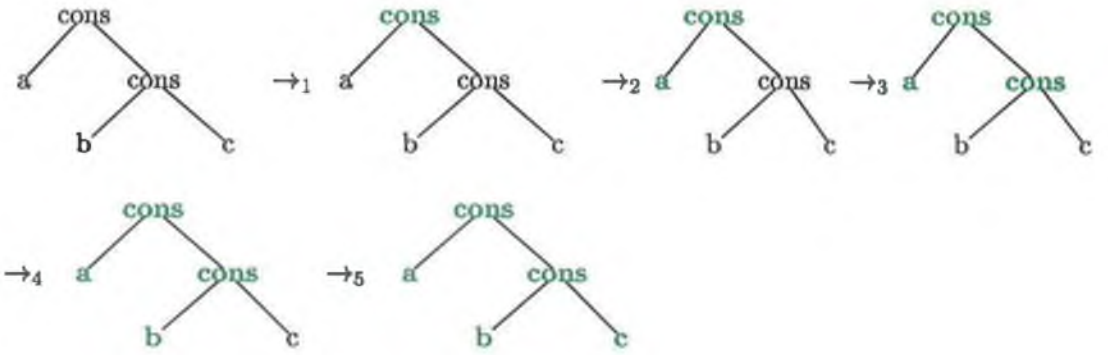


Figure 3.3: Top-down process

Definition 8 *Let $\mathcal{F}$ be a finite set of symbols and $\mathcal{Q}$ be a finite set of symbols of arity zero called states.*

A **transition** *is a rewrite rule $c \to q$, where $c \in \mathcal{T}(\mathcal{F} \bigcup \mathcal{Q})$ and $q \in \mathcal{Q}$.*

A **normalized transition** *is a transition $c \to q$ where $c = q' \in \mathcal{Q}$ or $c = f(q_1, \ldots, q_n) \in \mathcal{T}(\mathcal{F} \bigcup \mathcal{Q})$ with $f \in \mathcal{F}$, $ar(f) = n$ and $q_1, \ldots, q_n \in \mathcal{Q}$.*

Definition 9 *(bottom-up non-deterministic finite tree automaton [Gen98a]) A* **bottom-up non-deterministic finite tree automaton** *is a quadruplet $\mathcal{A} = \{\mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta\}$ where $\mathcal{F}$ is a finite set of symbols, $\mathcal{Q}$ is a finite set of symbols of arity zero called states, $\mathcal{Q}_f$ is the set of terminal states such that $\mathcal{Q}_f \subseteq \mathcal{Q}$, and Δ is a set of normalized transitions.*

A tree automaton $\mathcal{A}$ is deterministic if for any term $t \in \mathcal{T}(\mathcal{F})$ there exists at most one state $q \in \mathcal{Q}$ such that $t \to^*_{\mathcal{A}} q$.

Definition 10 *(Tree language) The* **tree language** *recognized by $\mathcal{A}$ is*

$$\mathcal{L}(\mathcal{A}) = \{t \in \mathcal{T}(\mathcal{F}) \mid \exists q \in \mathcal{Q}_f \,.\, t \to^*_{\mathcal{A}} q\}.$$

The tree language recognized by the state q of the automaton $\mathcal{A}$ is

$$\mathcal{L}(\mathcal{A}, q) = \{t \in \mathcal{T}(\mathcal{F}) \mid t \to^*_{\mathcal{A}} q\}.$$

A tree language (set of terms) E is regular if there exists a bottom-up finite tree automaton $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A})$=E.

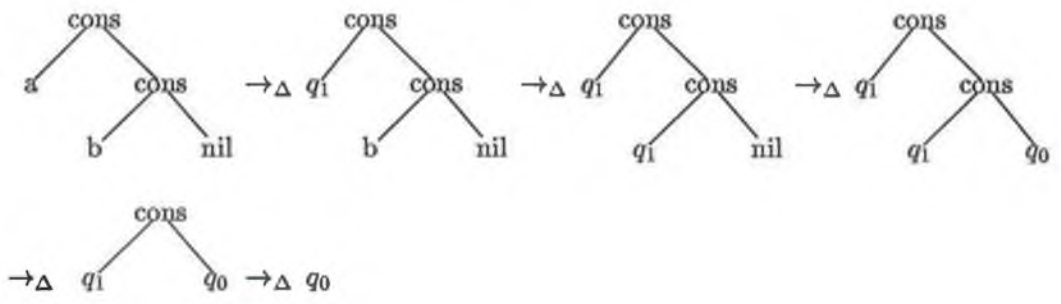


Figure 3.4: Graphical view of reduction process

From now on, only bottom-up non-deterministic finite tree automata are considered and we will refer to these as tree automata for short.

Example 8 *Let $\mathcal{F}$={cons:2, a:0, b:0, nil:0}, Q={q_0, q_1}, Q_f={q_0} and $\Delta = \{con(q_1, q_0) \rightarrow q_0, nil \rightarrow q_0, a \rightarrow q_1, b \rightarrow q_1\}$, then $\mathcal{A} = \{\mathcal{F}, Q, Q_f, \Delta\}$ is a tree automaton.*

This automaton recognizes the set of lists composed of "a" and "b",

$\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}, q_0) = \{nil, cons(a, nil), cons(b, nil), cons(a, cons(a, nil)), \ldots\}$.

This automaton also gives $\mathcal{L}(\mathcal{A}, q_1)$={$a$, b}.

Figure 3.4 shows how the term $cons(a, cons(b, nil))$ can be reduced to the state q_0 by applying transitions from Δ. The sequence applied is the following: $a \rightarrow q_1$, $b \rightarrow q_1$, $nil \rightarrow q_0$, $con(q_1, q_0) \rightarrow q_0$ and $con(q_1, q_0) \rightarrow q_0$. As q_0 is the final state of the automaton, $cons(a, cons(b, nil))$ is recognized by $\mathcal{A}$.

In this thesis, the emphasis is placed on non-deterministic automata, but it is important to know that for each non-deterministic automaton there is an equivalent deterministic automaton.

Theorem 1 *([CDG+98]) Let L be a recognizable set of ground terms. Then there exists a deterministic finite tree automaton that accepts L.*

Algorithm 1 (Determinization [CDG+98]) *The algorithm takes a non-deterministic finite tree automaton $\mathcal{A}$={$\mathcal{F}, Q, Q_f, \Delta$} as input and returns a deterministic finite tree automaton $\mathcal{A}_d$={$\mathcal{F}, Q_d, Q_{df}, \Delta_d$}:*

1. $\mathcal{Q}_d = \emptyset$, $\Delta_d = \emptyset$

2. *repeat*

   if $s=\{q \in \mathcal{Q} | \exists q_1 \in s_1, \ldots, q_n \in s_n, f(q_1, \ldots, q_n) \rightarrow q \in \Delta\}$ *then*

   add s *to* $\mathcal{Q}_d$

   if $(f \in \mathcal{F})$ *and* $(s_1, \ldots, s_n \in \mathcal{Q}_d)$ *then*

   add $f(s_1, \ldots, s_n) \rightarrow s$ *to* Δ_d

   until no rule can be added to Δ_d

3. $\mathcal{Q}_{df} = \{s \in \mathcal{Q}_d | s \cap \mathcal{Q}_f \neq \emptyset\}$

From Algorithm 1, we can deduce that the generation of a deterministic automaton is exponential.

In Algorithm 1 only the accessible states[2] are considered. Example 9 details an example from [CDG+98] to illustrate the use of that particular algorithm.

Example 9 *Let* $\mathcal{F}=\{f:2, g:1, a:0\}$. *Consider the automaton* $\mathcal{A}=\{\mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta\}$ *with* $\mathcal{Q} = \{q, q_g, q_f\}$, $\mathcal{Q}_f = \{q_f\}$ *and* $\Delta = \left\{ \begin{array}{ll} a \rightarrow q & g(q) \rightarrow q \\ g(q) \rightarrow q_g & g(q_g) \rightarrow q_f \\ f(q,q) \rightarrow q & \end{array} \right\}$.

By applying Algorithm 1 on $\mathcal{A}$, *the deterministic automaton* $\mathcal{A}_d=\{\mathcal{F}, \mathcal{Q}_d, \mathcal{Q}_{df}, \Delta_d\}$ *is produced:*

- $\mathcal{Q}_d=\{\{q\}, \{q, q_g\}, \{q, q_g, q_f\}\}$
- $\mathcal{Q}_{df}=\{\{q, q_g, q_f\}\}$
- $\Delta_d = \left\{ \begin{array}{c} a \rightarrow \{q\} \\ g(\{q\}) \rightarrow \{q, q_g\} \\ g(\{q, q_g\}) \rightarrow \{q, q_g, q_f\} \\ g(\{q, q_g, q_f\}) \rightarrow \{q, q_g, q_f\} \end{array} \right\} \cup \{f(s_1, s_2) \rightarrow \{q\} | s_1, s_2 \in \mathcal{Q}_d\}$

To understand how $\mathcal{A}_d$ *is built, a few steps of the computation are presented:*

1. $\{\}_{\Delta_d}$ $\{\}_{\mathcal{Q}_d}$ *(step 1 of Algorithm 1)*

[2] a state q is accessible if there exists a ground term t such that $t \rightarrow_{\mathcal{A}}^{*} q$

2. $\left\{ a \to \{q\} \right\}_{\Delta_d} \left\{ \{q\} \right\}_{Q_d}$ *(step 2 of Algorithm 1)*

3. $\left\{ \begin{array}{c} a \to \{q\} \\ g(\{q\}) \to \{q, q_g\} \end{array} \right\}_{\Delta_d} \left\{ \begin{array}{c} \{q\} \\ \{q, q_g\} \end{array} \right\}_{Q_d}$ *(repeat step 2)*

4. $\left\{ \begin{array}{c} a \to \{q\} \\ g(\{q\}) \to \{q, q_g\} \\ g(\{q, q_g\}) \to \{q, q_g, q_f\} \end{array} \right\}_{\Delta_d} \left\{ \begin{array}{c} \{q\} \\ \{q, q_g\} \\ \{q, q_g, q_f\} \end{array} \right\}_{Q_d}$ *(repeat step 2)*

5. ... *(repeat step 2)*

6. $\left\{ \begin{array}{c} a \to \{q\} \\ g(\{q\}) \to \{q, q_g\} \\ \dots \end{array} \right\}_{\Delta_d} \left\{ \begin{array}{c} \{q\} \\ \{q, q_g\} \\ \{q, q_g, q_f\} \end{array} \right\}_{Q_d} \left\{ \{q, q_g, q_f\} \right\}_{Q_{df}}$ *(step 3)*

$\mathcal{A}$ is a non-deterministic automaton because in Δ there are two transitions with the same left-hand side: $g(q) \to q$ and $g(q) \to q_g$. This non-determinism disappears in $\mathcal{A}_d$:

$$\left\{ \begin{array}{c} g(q) \to q \\ g(q) \to q_g \\ \dots \end{array} \right\}_{\Delta} \Longrightarrow_{algorithm} \left\{ g(\{q\}) \to \{q, q_g\} \dots \right\}_{\Delta_d}$$

To conclude on tree automata, the usual set operations apply on the tree languages (cf. Proposition 3). Let $\mathcal{A}=\{\mathcal{F}, \mathcal{Q}_\mathcal{A}, \mathcal{Q}_\mathcal{A}{}^f, \Delta_\mathcal{A}\}$ and $\mathcal{B}=\{\mathcal{F}, \mathcal{Q}_\mathcal{B}, \mathcal{Q}_\mathcal{B}{}^f, \Delta_\mathcal{B}\}$ be two tree automata:

- union ($\cup$): $\mathcal{L}(\mathcal{A} \cup \mathcal{B})=\mathcal{L}(\mathcal{A}) \cup \mathcal{L}(\mathcal{B})$ and $\mathcal{A} \cup \mathcal{B}=\{\mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta_\mathcal{A} \times \Delta_\mathcal{B}\}$ where $\mathcal{Q} = \mathcal{Q}_\mathcal{A} \times \mathcal{Q}_\mathcal{B}$, $\mathcal{Q}_f = (\mathcal{Q}_\mathcal{A}{}^f \times \mathcal{Q}_\mathcal{B}) \cup (\mathcal{Q}_\mathcal{A} \times \mathcal{Q}_\mathcal{B}{}^f)$ and
  $\Delta_\mathcal{A} \times \Delta_\mathcal{B} = \left\{ \begin{array}{ll} f((q_1, q'_1), \dots, (q_n, q'_n)) \to (q, q') | & f(q_1, \dots, q_n) \to q \in \Delta_\mathcal{A} \text{ and} \\ & f(q'_1, \dots, q'_n) \to q' \in \Delta_\mathcal{B} \end{array} \right\}$.
- intersection ($\cap$): $\mathcal{L}(\mathcal{A} \cap \mathcal{B})=\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{B})$ and $\mathcal{A} \cap \mathcal{B}=\{\mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta_\mathcal{A} \times \Delta_\mathcal{B}\}$ where $\mathcal{Q} = \mathcal{Q}_\mathcal{A} \times \mathcal{Q}_\mathcal{B}$ and $\mathcal{Q}_f = \mathcal{Q}_\mathcal{A}{}^f \times \mathcal{Q}_\mathcal{B}{}^f$.
- complement of a language $\mathcal{L}(\mathcal{A})$, $(\mathcal{L}(\overline{\mathcal{A}}))$: $\mathcal{A}_d=\{\mathcal{F}, \mathcal{Q}_d, \mathcal{Q}_{df}, \Delta_d\}$ is computed with Algorithm 1 and $\overline{\mathcal{A}}=\{\mathcal{F}, \mathcal{Q}_d, \mathcal{Q}_d \setminus \mathcal{Q}_{df}, \Delta_d\}$

- inclusion ($\subseteq$): $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ if and only if $\mathcal{L}(\mathcal{A} \cap \overline{\mathcal{B}}) = \emptyset$.
- difference between two languages $\mathcal{L}(\mathcal{A})$ and $\mathcal{L}(\mathcal{B})$ ($\mathcal{L}(\mathcal{A}) \setminus \mathcal{L}(\mathcal{B})$): $\mathcal{A} \setminus \mathcal{B} = \mathcal{A} \cap \overline{\mathcal{B}}$

Proposition 3 *([CDG+98] The class of regular tree languages is closed under union, intersection and complement.*

The emptiness, inclusion, membership, intersection, intersection non-emptiness, finiteness and equivalence are decidable.

Now that the basic definitions have been presented, the method developed by Genet and Klay is introduced.

3.2 Genet and Klay's idea

Genet and Klay [GK00a] use a term rewriting system to model the protocol and a tree automaton to model the communications. Starting from an initial set of communications, and by using the term rewriting system, an over-approximation of the set of reachable configurations is computed. The secrecy and authentication properties are then checked on the approximation automaton built.

In this section, the theory behind this approach is introduced and the Needham-Schroeder-Lowe protocol [Low95] is used as an example.

3.2.1 Theory

Genet and Klay [GK00a] re-used the PhD work of Genet [Gen98a]. In [Gen98a], for a TRS $\mathcal{R}$ and a set of terms defined by the tree automaton $\mathcal{A}=\{\mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta\}$ $E \subseteq \mathcal{T}(\mathcal{F})$ it is explored how to compute $\mathcal{R}^*(\mathcal{L}(\mathcal{A}))$.

Genet's idea is to get $\mathcal{R}^*(\mathcal{L}(\mathcal{A}))$ starting from $\mathcal{A}$ and $\mathcal{R}$ by extending the set of transitions Δ such that the automaton guarantees:

$$\forall t \,.\, (t \in \mathcal{R}^*(\mathcal{L}(\mathcal{A})) \;\wedge\; t \rightarrow_{\Delta}^* q \;\wedge\; t \rightarrow_{\mathcal{R}}^* u) \Rightarrow u \rightarrow_{\Delta}^* q.$$

As noted by Genet, this property is similar to the confluence property introduced in Section 3.1.1.2 applied on the term rewriting system: ($\mathcal{R} \cup \Delta$). Thus Genet worked on an algorithm similar to the completion algorithm.

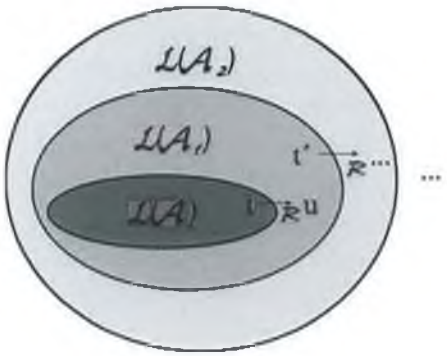


Figure 3.5: Intuition to build $\mathcal{R}^*(\mathcal{L}(\mathcal{A}))$

With that approach, the set of final states and the previous transitions remain unchanged, so after each step the new language contains all the previous languages (cf Figure 3.5).

Moreover for a language $\mathcal{L}(\mathcal{A})$ and a TRS $\mathcal{R}$, $\mathcal{R}^*(\mathcal{L}(\mathcal{A}))$ is not always regular [GT95, Jac96]. [Gen98a] explains how to compute an approximation automaton $\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A})$ ($E \subseteq \mathcal{L}(\mathcal{A})$ and $\mathcal{R}$ left linear) such that $\mathcal{R}^*(\mathcal{L}(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A}))$.

Let $\mathcal{A}=\{\mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta\}$ be a tree automaton. First, an explanation of what is meant by **abstraction** and **normalized transitions** is given. Then his **approximation function** and his algorithm to compute the abstract model are introduced.

For Genet, an abstraction is a function that maps all subterms of a term to states.

Definition 11 *(Abstraction function [Gen98a]) Given a configuration $s \in \mathcal{T}(\mathcal{F} \bigcup \mathcal{Q}) \setminus \mathcal{Q}$.*

An **abstraction** *of s is a mapping α:*

$$\alpha : \{s|_p \mid p \in \mathcal{P}os_{\mathcal{F}}(s)\} \mapsto \mathcal{Q}$$

The mapping α is extended on $\mathcal{T}(\mathcal{F} \bigcup \mathcal{Q})$ by defining α as the identity on $\mathcal{Q}$.

Example 10 *Let $\mathcal{A}=\{\mathcal{F}, \mathcal{Q}, \mathcal{Q}f, \Delta\}$, where $\mathcal{F}=\{f,h,a,b\}$, $\mathcal{Q}=\{q_0, q_1, q_2, q_3\}$, $\mathcal{Q}f=\{q_0\}$ and $\Delta=\{f(q_0,q_0) \rightarrow q_0,\ h(q_1) \rightarrow q_0,\ a \rightarrow q_1,\ b \rightarrow q_0\}$.*

If a term $s=f(h(a), b)$ is given then $\alpha=\{h(a) \to q_1,\ a \to q_2,\ b \to q_3\}$ is an abstraction of s (it is mapping each subterm of s to a state).

In order to keep the set of final states unchanged, Genet needs to add normalized transitions to his current automaton to keep it normalized. The normalization process can be defined inductively with the abstraction in Definition 11.

Definition 12 *(Normalization function [Gen98a]) Let $s \to q$ be a transition such that $s \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$, $q \in \mathcal{Q}$, and α an abstraction of s. The set $Norm_\alpha(s \to q)$ of* **normalized transitions** *is inductively defined by:*

1. *if $s = q$, then $Norm_\alpha(s \to q) = \emptyset$, and*
2. *if $s \in \mathcal{Q}$ and $s \neq q$, then $Norm_\alpha(s \to q) = \{s \to q\}$, and*
3. *if $s = f(t_1, \ldots, t_n)$, then $Norm_\alpha(s \to q) =$*
   $\{f(\alpha(t_1), \ldots, \alpha(t_n)) \to q\} \cup \bigcup_{i=1}^{n} Norm_\alpha(t_i \to \alpha(t_i))$.

Example 11 *Let $\mathcal{A}=\{\mathcal{F}, \mathcal{Q}, \mathcal{Q}f, \Delta\}$, where $\mathcal{F}=\{f,h,a,b\}$, $\mathcal{Q}=\{q_0,q_1,q_2,q_3\}$, $\mathcal{Q}f=\{q_0\}$, $\Delta=\{f(q_0,q_0) \to q_0,\ h(q_1) \to q_0,\ a \to q_1,\ b \to q_0\}$ and $\alpha=\{h(a) \to q_1,\ a \to q_2,\ b \to q_3\}$.*

Following Definition 12:

$$
\begin{aligned}
Norm_\alpha(f(h(a), b) \to q_0) = {} & \{f(q_1, q_3) \to q_0\} \bigcup Norm_\alpha(h(a) \to q_1) \\
& \bigcup Norm_\alpha(b \to q_3) \\
= {} & \{f(q_1, q_3) \to q_0\} \bigcup \{h(q_2) \to q_1\} \\
& \bigcup Norm_\alpha(a \to q_2) \bigcup \{b \to q_3\} \\
= {} & \{f(q_1, q_3) \to q_0,\ h(q_2) \to q_1,\ a \to q_2,\ b \to q_3\}
\end{aligned}
$$

At this stage Genet's approximation function can be introduced. This function links a rewrite rule from the TRS, a state of the current automaton and a substitution of the rewrite rule variables by states of the current automaton to a sequence of states.

Definition 13 *(Approximation Function [Gen98a]) Let $\mathcal{Q}$ be a set of states, $\mathcal{Q}_{new}$ be any set of new states such that $\mathcal{Q} \bigcap \mathcal{Q}_{new} = \emptyset$, and $\mathcal{Q}^*_{new}$ the set of sequences $q_1 \dots q_k$ of states in $\mathcal{Q}_{new}$. Let $\Sigma(\mathcal{Q}, \mathcal{X})$ be the set of substitutions of variables in $\mathcal{X}$ by the states in $\mathcal{Q}$.*

An **approximation function** *is a mapping γ:*

$\mathcal{R} \times (\mathcal{Q} \bigcup \mathcal{Q}_{new}) \times \Sigma((\mathcal{Q} \bigcup \mathcal{Q}_{new}), \mathcal{X}) \mapsto \mathcal{Q}^*_{new}$, such that $\gamma(l \to r, q, \sigma) = q_1 \dots q_k$ where $k = Card(Pos_{\mathcal{F}}(r))$.

This γ function is linked to the abstraction function α. If $\gamma(l \to r, q, \sigma) = q_1 \dots q_k$ and if $\alpha(r\sigma|_{p_i}) = q'_i$ with $p_i \in Pos_{\mathcal{F}}(r) = \{p_1, \dots, p_k\}$ then $q_i = q'_i$ for $i = 1 \dots k$.

Example 12 *Consider the alphabet $\mathcal{F} = \{0 : 0, s : 1\}$, the set of states $\mathcal{Q} = \{q_0, q_1\}$ and the term rewriting system $\mathcal{R} = \{s(x) \to s(s(x))\}$. Possible approximation functions are:*

- $\gamma(s(x) \to s(s(x)),\ q_1,\ \{x = q_0\}) = q_2$;

  we only have one state because $Pos_{\mathcal{F}}(s(s(q_0))) = \{1\}$ as $Root(s(q_0)) \in \mathcal{F}$ and $Root(q_0) \notin \mathcal{F}$,

- $\gamma(s(x) \to s(s(x)),\ q_1,\ \{x = q_2\}) = q_3$,

- $\gamma(s(x) \to s(s(x)),\ q_2,\ \{x = q_0\}) = q_4$,

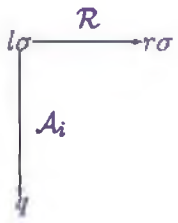
- $\gamma(s(x) \to s(s(x)),\ q_2,\ \{x = q_2\}) = q_5$.

With $\mathcal{Q}_{new} = \{q_2, q_3, q_4, q_5\}$. These approximation functions are not the only possible approximation functions for that system, four were picked to illustrate Definition 13.

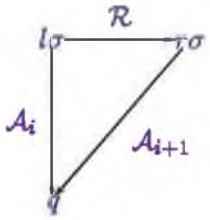
In the rest of the thesis, let $\mathcal{Q}_{new}$ be any set of new states such that $\mathcal{Q} \bigcap \mathcal{Q}_{new} = \emptyset$, and $\mathcal{Q}_u = \mathcal{Q} \bigcup \mathcal{Q}_{new}$.

Algorithm 2 *(Completion [Gen98a]) Starting from a left-linear TRS $\mathcal{R}$, an initial automaton $\mathcal{A}_0 = \mathcal{A}$ and an approximation function γ, Genet and Klay construct $\mathcal{A}_{i+1}$ from $\mathcal{A}_i$ by:*

1. *searching for a critical pair* $(r\sigma, q)$ *with a state* $q \in \mathcal{Q}$, *a rewrite rule* $l \to r$ *and a substitution* $\sigma \in \Sigma(\mathcal{Q}, \mathcal{X})$ *such that* $l\sigma \to^*_{\mathcal{A}_i} q$ *and* $r\sigma \to^*_{\mathcal{A}_i} q$.



2. $\mathcal{A}_{i+1} = \mathcal{A}_i \cup Norm_\gamma(r\sigma \to q)$.



$\mathcal{A}_i \cup Norm_\gamma(r\sigma \to q)$ adds the states created by $Norm_\gamma(r\sigma \to q)$ to the set of states of $\mathcal{A}_i$. It also increases the set of transitions of $\mathcal{A}_i$ with the normalized transitions of $Norm_\gamma(r\sigma \to q)$.

The above process is iterated until it stops on a tree automaton $\mathcal{A}_k$ such that there are no further critical pairs.

Example 13 illustrates the computation process. This example demonstrates one of the drawbacks of the technique; the computation might not stop.

Example 13 *In this example taken from [Gen98b],* $\mathcal{A} = \{\mathcal{F}, \{q_0, q_1, q_2\}, \{q_1\}, \Delta\}$ *is a tree automaton where* $\mathcal{F} = \{app : 2, cons : 2, nil : 0, a : 0\}$, $\Delta = \{app(q_0, q_0) \to q_1, cons(q_2, q_1) \to q_0, nil \to q_0, nil \to q_1, a \to q_2\}$, $\mathcal{R} = \{rl\}$ *with* $rl = app(cons(x, y), z) \to cons(x, app(y, z))$, *and* γ *(Definition 13) the approximation function mapping every tuple* (rl, q, σ) *to one state* $(cons(x, app(y, z))$ *of* rl *contains only one subterm* $app(y, z)$*).*

Genet and Klay process is now used to compute $\mathcal{A}_{i+1}$ *from* $\mathcal{A}_i$*:*

1. *the critical pair* $(cons(q_2, app(q_1, q_0)), q_1)$ *is deduced from* $app(cons(q_2, q_1), q_0) \to^*_{\mathcal{A}} q_1$ *and* $cons(q_2,\ app(q_1,\ q_0)) \to^*_{\mathcal{A}} q_1$;

2. $\mathcal{A}_1 = \mathcal{A} \cup Norm_\gamma(cons(q_2,\ app(q_1,\ q_0)) \rightarrow q_1)$:

- $\gamma\ (rl, q_1, \{x = q_2, y = q_1, z = q_0\})$ *is computed; the state* q_3 *is introduced and* $\gamma\ (rl, q_1, \{x = q_2, y = q_1, z = q_0\}) = q_3$.
- $Norm_\gamma(cons(q_2, app(q_1, q_0)) \rightarrow q_1)$ *is computed with the* γ *given just above;*

$$\begin{aligned} Norm_\gamma(cons(q_2,\ app(q_1,\ q_0)) \rightarrow q_1) \quad &= \{cons(q_2, q_3) \rightarrow q_1\} \cup \\ &\quad Norm_\gamma(app(q_1, q_0) \rightarrow q_3) \\ &= \{cons(q_2, q_3) \rightarrow q_1,\ app(q_1, q_0) \rightarrow q_3\}; \end{aligned}$$

- *the sets of* $\mathcal{A}$ *are updated to produce* $\mathcal{A}_1$;

  The transitions $cons(q_2, q_3) \rightarrow q_1$ *and* $app(q_1, q_0) \rightarrow q_3$ *are added to* Δ *the current automaton set of transitions and* q_3 *is added to the set of states.*

3. *the critical pair* $(cons(q_2, app(q_3, q_0)), q_3)$ *is deduced from*

   $app(cons(q_2, q_3), q_0) \rightarrow^*_{\mathcal{A}_1} q_3$ *and* $cons(q_2,\ app(q_3,\ q_0)) \rightarrow^*_{\mathcal{A}_1} q_3$;

4. $\mathcal{A}_2 = \mathcal{A}_1 \cup Norm_\gamma(cons(q_2,\ app(q_3,\ q_0)) \rightarrow q_3)$:

- $\gamma\ (rl, q_3, \{x = q_2, y = q_3, z = q_0\})$ *is computed; A new state* q_4 *is introduced and* $\gamma\ (rl, q_3, \{x = q_2, y = q_3, z = q_0\}) = q_4$.
- $Norm_\gamma(cons(q_2,\ app(q_3,\ q_0)) \rightarrow q_3)$ *is computed with* γ *above;*

$$\begin{aligned} Norm_\gamma(cons(q_2, app(q_3, q_0)) \rightarrow q_3) \quad &= \{cons(q_2, q_4) \rightarrow q_3\} \cup \\ &\quad Norm_\gamma(app(q_3, q_0) \rightarrow q_4) \\ &= \{cons(q_2, q_4) \rightarrow q_3,\ app(q_3, q_0) \rightarrow q_4\}; \end{aligned}$$

- *the sets of* $\mathcal{A}_1$ *are increased with the new states and transitions to produce* $\mathcal{A}_2$;

  The transitions $cons(q_2, q_4) \rightarrow q_3$ *and* $app(q_3, q_0) \rightarrow q_4$ *are added to* Δ *the current automaton set of transitions and* q_4 *is added to the set of states.*

5. *the critical pair* $(cons(q_2,\ app(q_4,\ q_0)), q_4)$ *is deduced from*

   $app(cons(q_2,\ q_4),\ q_0) \rightarrow^*_{\mathcal{A}} q_4$ *and* $cons(q_2,\ app(q_4,\ q_0)) \rightarrow^*_{\mathcal{A}} q_4$; *etc.*

The computation of the approximation automaton goes forever.

In his thesis, Genet refined the definition of the γ function that guarantees that the completion algorithm terminates: Definition 14. With this refined approximation, the computation in the above example terminates.

Definition 14 *(Ancestor Approximation Function [Gen98a])*
An approximation function γ is an ancestor approximation function if:

1. $\forall l \to r \in \mathcal{R}$, $\forall q \in Q_u$, $\forall \sigma_1, \sigma_2 \in \Sigma(Q_u, \mathcal{X})$,
   $\gamma(l \to r, q, \sigma_1) = \gamma(l \to r, q, \sigma_2)$, *and*
2. $\forall l_1 \to r_1,\ l_2 \to r_2 \in \mathcal{R}$, $\forall q \in Q_u$, $\forall q_1 \dots q_k \in Q_{new}$, $\forall \sigma_1, \sigma_2 \in \Sigma(Q_u, \mathcal{X})$,
   $\gamma(l_1 \to r_1, q, \sigma_1) = q_1 \dots q_k \Rightarrow \forall i = 1 \cdots k, \gamma(l_2 \to r_2, q_i, \sigma_2) = \gamma(l_2 \to r_2, q, \sigma_2)$

Theorem 2 *Every automaton built with an ancestor approximation function is finite.*

The proof of this theorem is given in Appendix B

Theorem 3 [Gen98a] on the other hand gives the main advantage of the technique, which is that the completeness is guaranteed for any approximation function that meets Definition 13. This means that if a property is verified on the abstract model then this property will be also guaranteed by the concrete model.

Theorem 3 *[GK00a](Completeness) Given a tree automaton $\mathcal{A}$ and a left-linear TRS $\mathcal{R}$, for any approximation function γ matching Definition 13:*

$$\mathcal{R}^*(\mathcal{L}(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A}))$$

The above theorem is limited to left-linear rewriting system. From a practical point of view, this is a big restriction; using a left linear term rewriting to model a protocol might change the protocol. For example, we have a protocol where a participant only sends nonces that he created for himself and we have nonces that

are modeled by $N(x,y)$ with x the name of the creator and y the name of the receiver. Then to follow the protocol in the term rewriting system a term $N(x,x)$ should be found but instead if we want to use the approximation approach we have to have $N(x,y)$. This means that the agent will be sending any nonce not only the one he created for himself; thus the verified protocol is different from the real one. Luckily, Theorem 3 can be extended to non left-linear TRS under specific conditions: Theorem 4 [GK00a]. When we will explain how the approach is used to verify cryptographic protocols, we will see that the conditions are verified and thus that the theorem can apply. But first, we introduce the following definition:

Definition 15 *(States matching) Let $\mathcal{A}$ be a tree automaton, Q its set of states, $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$ a non linear term, and $\{p_1,...,p_n\} \subseteq Pos(t)$ the set of positions of a non linear variable x in t.*

t_{lin} denotes the linearized form of the term t, where all occurrences of non linear variables are replaced by disjoint variables (ie. if $t = f(x,y,g(x,x))$ gives $t_{lin} = f(x',y,g(x'',y'')))$.

It is defined that states $q_1,...,q_n \in Q$ are matched by x if and only if there exits $\sigma \in \Sigma(Q,\mathcal{X})$ such that $t_{lin}\sigma \rightarrow^*_{\mathcal{A}} q \in Q$, and $t_{lin}\sigma|_{p_1} = q_1$, ..., $t_{lin}\sigma|_{p_n} = q_n$.

Theorem 4 *(Completeness extended to non left-linear TRS) Given a tree automaton $\mathcal{A}$ and a TRS $\mathcal{R}$, $\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A})$ the corresponding approximation automaton and Q its set of states. For all non left-linear rule $l \rightarrow r \in \mathcal{R}$, for all non linear variables x of l, for all states $q_1,...,q_n \in Q$ matched by x, if either $q_1 = ... = q_n$ or $\mathcal{L}(\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A}), q_1) \bigcap ... \bigcap \mathcal{L}(\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A}), q_n) = \emptyset$ then:*

$$\mathcal{R}^*(\mathcal{L}(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A}))$$

The proof of Theorem 4 done by Genet and Klay [GK00a] is given in Appendix C.

In a recent report [GFT03], conditions on the TRS, the tree automaton and the approximation function are given to get $\mathcal{R}^*(\mathcal{L}(\mathcal{A})) = \mathcal{L}(\mathcal{T}_{\mathcal{R}} \uparrow (\mathcal{A}))$. The main definitions and theorems regarding the equality introduced in the following

paragraphs are taken from [GFT03] (for more details about the proofs, the reader can refer to [GFT03]). However the equality cannot happen with cryptographic protocols as shown in the next section.

Definition 16 *(Right-linearity condition) A tree automaton* $\mathcal{A} = \{\mathcal{F}, Q, Q_f, \Delta\}$ *and a TRS* $\mathcal{R}$ *satisfy the right-linearity condition if*

1. $\mathcal{R}$ *is right-linear, or*
2. $\forall q \in \mathcal{A} : \exists t \in \mathcal{T}(\mathcal{F}) : \mathcal{L}(\mathcal{A}, q) \subseteq \mathcal{R}^*(t)$

The following example illustrates the necessity of these two conditions.

Example 14 *We have:*

- $\mathcal{F} = \{f : 1, g : 2, a : 0, b : 0\}$ *an alphabet,*
- $\mathcal{R} = \{f(x) \to g(x, x)\}$ *a non right-linear TRS and*
- $\mathcal{A} = \{\mathcal{L}, \{q_0, q_1\}, \{q_0\}, \{f(q_1) \to q_0, a \to q_1, b \to q_1\}\}$ *a tree automaton.*

Given $\mathcal{L}(\mathcal{A}) = \{f(a), f(b)\}$. *For any abstraction function* γ, *the completion process adds the transition* $g(q_1, q_1) \to q_0$ *to the current automaton and*

$$\mathcal{L}(\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A})) = \{f(a), f(b), g(a, a), g(b, b), g(a, b), g(b, a)\}$$

which is a superset of

$$\mathcal{R}^*(\mathcal{L}(\mathcal{A})) = \{f(a), f(b), g(a, a), g(b, b)\}.$$

If $\mathcal{R}$ *was right linear or if there was no transition* $b \to q_1$, *then:*

$$\mathcal{L}(\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A})) = \mathcal{R}^*(\mathcal{L}(\mathcal{A})).$$

Similarly, by adding the rule $a \to b$ *into* $\mathcal{R}$, *the exact case would be generated, and the right-linearity condition would be satisfied since for* $q_1 \in \mathcal{A}$ *there would exist the term* a *such that* $\mathcal{L}(\mathcal{A}, q_1) \subseteq \mathcal{R}^*(\{a\}) = \{a, b\}$.

To be coherent with a tree automaton $\mathcal{A}$ and a TRS $\mathcal{R}$, an abstraction function γ must guarantee for each of its states:

- the state is not in the states of $\mathcal{A}$, or
- terms recognized by the state in $\mathcal{A}$ are either a term t' recognized by the subterm linked to the state by γ or $\mathcal{R}$-descendants of t'.

In [GFT03], α (Definition 11) is redefined in Definition 17. The normalization process Definition 12 is also redefined in Definition 19. This is done in order to introduce a coherent abstraction function (Definition 20) for which the automaton completion algorithm is exact.

Definition 17 *(New abstraction function) Let $\mathcal{F}$ be a set of symbols, and $\mathcal{Q}$ a set of states. An abstraction α maps every normalized configuration into a state:*

$$\alpha : \{f(q_1, \ldots, q_n) |\ f \in \mathcal{F},\ ar(f) = n \text{ and } q_1, \ldots, q_n \in \mathcal{Q}\} \mapsto \mathcal{Q}.$$

Definition 18 *(Abstraction state) Let $\mathcal{F}$ be a set of symbols, and $\mathcal{Q}$ a set of states. For a given abstraction function α (Definition 17) and for all configuration $t \in \mathcal{T}(\mathcal{F} \cap \mathcal{Q})$ the abstraction state of t, denoted by $top_\alpha(t)$, is defined by:*

1. *if $t \in \mathcal{Q}$, then $top_\alpha(t) = t$,*
2. *if $t = f(q_1, \ldots, q_n)$ then $top_\alpha(t) = \alpha(f(top_\alpha(t_1), \ldots, top_\alpha(t_n)))$.*

Definition 19 *(New normalization function) Let $\mathcal{F}$ be a set of symbols, and $\mathcal{Q}$ a set of states, $s \to q$ a transition such that $s \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$, $q \in \mathcal{Q}$, and α an abstraction function (Definition 17). The set $Norm_\alpha(s \to q)$ of normalized transitions is inductively defined by:*

1. *if $s = q$, then $Norm_\alpha(s \to q) = \emptyset$, and*
2. *if $s \in \mathcal{Q}$ and $s \neq q$, then $Norm_\alpha(s \to q) = \{s \to q\}$, and*
3. *if $s = f(t_1, \ldots, t_n)$, then $Norm_\alpha(s \to q) =$*
   $\{f(top_\alpha(t_1), \ldots, top_\alpha(t_n)) \to q\} \cup \bigcup_{i=1}^{n} Norm_\alpha(t_i \to top_\alpha(t_i))$.

Definition 20 *(Coherent abstraction function) Let $\mathcal{A} = \{\mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta\}$ be a tree automaton, $\mathcal{R}$ be a TRS and α be an abstraction function. The function α is said to be coherent with $\mathcal{R}$ and $\mathcal{A}$ if for all $t \in Dom(\alpha)$, for all $q \in \mathcal{Q} \cap Ran(\alpha)$ if $\alpha(t) = q$ then $t \to q \in \mathcal{A}$ and there exists a term $t' \in (\mathcal{T}(\mathcal{F}))$ called the representative of q such that $t' \to_{\mathcal{A}}^* t$ and $\mathcal{L}(\mathcal{A}, q) \subseteq \mathcal{R}^*(\{t'\})$.*

We have $\mathcal{A} = \{\mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta\}$ a tree automaton. During the completion, the coherence will stand with the γ function when the normalizations (Definition 12) produce transitions of the form $f(q_1, \ldots, q_n) \to q$ with $f \in \mathcal{F}$, $ar(f) = n$, $q_1, \ldots, q_n \in \mathcal{Q}_u$ *and* $q \in \mathcal{Q} \cap \mathcal{Q}_u$ that satisfy:

- $f(q_1, \ldots, q_n) \to q \in \mathcal{A}$, and
- there exists a term $t' \in (\mathcal{T}(\mathcal{F}))$ such that $t' \to_{\mathcal{A}}^* f(q_1, \ldots, q_n)$ and $\mathcal{L}(\mathcal{A}, q) \subseteq \mathcal{R}^*(\{t'\})$.

Theorem 5 *Let $\mathcal{A}$ be a tree automaton, $\mathcal{R}$ be a TRS and α (Definition 17) be an injective abstraction function coherent with $\mathcal{A}$ and $\mathcal{R}$. If $\mathcal{A}$ and $\mathcal{R}$ satisfy the right-linearity condition, and if $\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A})$ is the automaton produced by completion with α then:*

$$\mathcal{L}(\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A})) \subseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A})).$$

Theorem 6 *Let $\mathcal{A}$ be a tree automaton, $\mathcal{R}$ be a TRS and α (Definition 17) be an injective abstraction function coherent with $\mathcal{A}$ and $\mathcal{R}$. Let $\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A})$ the automaton produced by completion with α. If $\mathcal{A}$ and $\mathcal{R}$ satisfy the right-linearity condition and if $\mathcal{R}$ and $\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A})$ fulfill the condition of Theorem 4, then:*

$$\mathcal{L}(\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A})) = \mathcal{R}^*(\mathcal{L}(\mathcal{A})).$$

The proofs of these theorems can be found in [GFT03].

The next subsection shows how the theory introduced in this subsection (the γ Definition 13 and the completion algorithm) is used to verify cryptographic protocols.

3.2.2 Cryptographic protocol verification

In the introduction, the Needham-Schroeder-Lowe protocol [Low95] was briefly mentioned. This protocol is used in this section to illustrate Genet and Klay's approach.

In the Needham-Schroeder-Lowe protocol [Low95], two agents, Alice and Bob, want to establish a secure communication using a public key infrastructure. Before they exchange any vital information, they use the Needham-Schroeder-Lowe protocol (cf. Figure 3.6) to exchange nonces. The role of these nonces in later messages is to identify the sender.

	Alice initiates a protocol run, sending a nonce Na and her name A to Bob encrypted by Bob's public key.
Message 1:	$A \Longrightarrow B : \{Na, A\}_{Kb}$
	Bob responds to Alice's message with a further nonce Nb encrypted by Alice's public key.
Message 2:	$B \Longrightarrow A : \{Na, Nb, B\}_{Ka}$
	Alice proves her existence by sending Nb back to Bob encrypted by Bob's public key.
Message 3:	$A \Longrightarrow B : \{Nb\}_{Kb}$

Figure 3.6: Needham-Schroeder-Lowe protocol

The goal of Genet and Klay's technique is to compute an automaton that recognizes an over-approximation of the reachable configurations of the network from an initial automaton that models the initial configurations and a TRS that models the protocol.

Their initial automaton models the initial configuration of the network and the intruder's initial knowledge and abilities. Their TRS models the protocol steps and also some intruder's abilities. Their approximation function has been introduced in Section 3.2.1.

The syntax and the semantics used in [GK00a] are summarized in Table 3.1.

agt(x)	x is an agent
c_init(x, y, z)	x thinks he has established a communication with y but he really communicates with z
c_resp(x, y, z)	x thinks he responds to a request of communication from y but he really communicates with z
cons(x, y)	concatenation of the information x and y
encr(x, y, z)	z is encrypted by participant y with the key x
goal(x, y)	x wants to communicate with y
mesg(x, y, z)	z is a message sent by x to y
N(x, y)	nonce created by x to communicate with y
null	end of list
pubkey(x)	public key of x

Table 3.1: Description of the terms used

3.2.2.1 Initial automaton for the Needham-Schroeder-Lowe protocol

Figure 3.7 gives the **initial automaton** for the Needham-Schroeder-Lowe protocol (later in this document, it will be shown that this initial automaton is also valid for other protocols). This automaton has seven states $\mathcal{Q}$={q_0, q_1, q_2, q_3, q_4, q_5, q_{13}} and one final state $\mathcal{Q}_f$={q_{13}}. The automaton uses the alphabet $\mathcal{F}$={A:0, B:0, 0:0, s:1, agt:1, U:2, goal:2, mesg:3, cons:2, pubkey:1, encr:3, N:2, null:0}[3]. All the automaton transitions are normalized transitions[4].

Transitions of **Part 1** in Figure 3.7 model the initial configuration of the network.

In this network, A (Alice) and B (Bob) are two trusted agents. They follow the protocol rules and the intruder does not have access to their private information, unless he catches it by spying on their exchanges. We also have an unbounded number of untrusted agents: 0, s(0), s(s(0)), etc. . They are untrustable as the intruder has access to their private information and can then usurp their identity. A, B, 0, s(0),

[3]format of the alphabet is "term : arity of the term"

[4]transitions have the format $f(q_1, ..., q_k) \rightarrow q$ where $f \in \mathcal{F}$, $q \in \mathcal{Q}$ and $q_i \in \mathcal{Q}$ (k is the arity of f and $i \in [1...k]$)

Automaton	nspk		
States	$q_0\ q_1\ q_2\ q_3\ q_4\ q_5\ q_{13}$		
Final States	q_{13}		
Transitions			
Part 1	$0 \rightarrow q_0$	$s(q_0) \rightarrow q_0$	
	$A \rightarrow q_1$	$B \rightarrow q_2$	
	$agt(q_0) \rightarrow q_3$	$agt(q_1) \rightarrow q_4$	$agt(q_2) \rightarrow q_5$
	$U(q_{13}, q_{13}) \rightarrow q_{13}$		
	$goal(q_3, q_3) \rightarrow q_{13}$	$goal(q_3, q_4) \rightarrow q_{13}$	$goal(q_3, q_5) \rightarrow q_{13}$
	$goal(q_4, q_4) \rightarrow q_{13}$	$goal(q_4, q_5) \rightarrow q_{13}$	$goal(q_5, q_3) \rightarrow q_{13}$
	$goal(q_5, q_5) \rightarrow q_{13}$	$goal(q_5, q_4) \rightarrow q_{13}$	$goal(q_5, q_3) \rightarrow q_{13}$
Part 2	$agt(q_0) \rightarrow q_{13}$	$agt(q_1) \rightarrow q_{13}$	$agt(q_2) \rightarrow q_{13}$
	$mesg(q_{13}, q_{13}, q_{13}) \rightarrow q_{13}$	$null \rightarrow q_{13}$	
	$cons(q_{13}, q_{13}) \rightarrow q_{13}$		
	$pubkey(q_3) \rightarrow q_{13}$	$pubkey(q_4) \rightarrow q_{13}$	$pubkey(q_5) \rightarrow q_{13}$
	$encr(q_{13}, q_3, q_{13}) \rightarrow q_{13}$		
	$N(q_3, q_3) \rightarrow q_{13}$	$N(q_3, q_4) \rightarrow q_{13}$	$N(q_3, q_5) \rightarrow q_{13}$

Figure 3.7: Initial automaton of the Needham-Schroeder-Lowe protocol

s(s(0)), ... are just names, they are linked to their role by the transitions $agt(q_0) \rightarrow q_3$, $agt(q_1) \rightarrow q_4$ and $agt(q_2) \rightarrow q_5$. Those transitions allow us to distinguish 3 agents; two trusted ones recognized by q_4 and q_5, and one untrusted one recognized by q_3. The untrusted agents are compounded to one agent (like Agent R in Figure 2.3 in Section 2.4). The communications between the three agents are studied.

Initially, all the agents want to communicate with each other. To express this, a transition of the form $goal(q_i, q_j) \rightarrow q_{13}$ (the agent recognized by the state q_i wants to communicate with the one recognize by the state q_j) is used. By doing this a special meaning is attached to the state q_{13}: q_{13} *is the state that corresponds to the network.* There are nine transitions of this form to cover all the possible initial configurations involving our three agents:

- q_3 wants to communicate with to q_3 or q_4 or q_5

- q_4 wants to communicate with to q_3 or q_4 or q_5
- q_5 wants to communicate with to q_3 or q_4 or q_5

The transition U(q_{13}, q_{13}) $\rightarrow$ q_{13} represents sets of configurations. Without this transition, the automaton would only be able to recognize one configuration of the network at a time; for example goal(agt(A),agt(B)) or goal(agt(B),agt(A)). With this transition, the concatenation of two or more configurations can be modeled. The automaton is then able to recognize sets of configurations such as U(goal(agt(A),agt(B)), goal(agt(B),agt(A))), the set containing the configurations goal(agt(A),agt(B)) and goal(agt(B),agt(A)).

Transitions of **Part 2** in Figure 3.7 model some intruder initial knowledge and abilities. It means that q_{13} ***corresponds to the network but also to the intruder knowledge.***

In [GK00a], the intruder is the network and he has the capabilities of the Dolev-Yao intruder [DY83]. The intruder can intercept all messages exchanged on the network and can decrypt messages if he has captured the appropriate decryption keys. In addition he can build and send fraudulent messages if he has the appropriate encryption keys.

The intruder initially knows:

- all the agents in the network, which is why we have the transitions of the form $agt(q_i) \rightarrow q_{13}$,
- all the agents' public keys; the transitions of the form $pubkey(q_i) \rightarrow q_{13}$,
- the nonces created by the untrusted agents; the transitions of the form $N(q_3,q_i) \rightarrow q_{13}$.

He can create and send fraudulent messages using his knowledge. In Figure 3.7 those abilities are modeled by three transitions:

- cons(q_{13}, q_{13}) $\rightarrow$ q_{13}: he can associate two known pieces of information to create new information,

- encr(q_{13}, q_3, q_{13}) $\rightarrow$ q_{13}: he can encrypt information with what he knows[5],
- mesg(q_{13}, q_{13}, q_{13}) $\rightarrow$ q_{13}: he can send fraudulent messages using what he knows.

3.2.2.2 TRS for the Needham-Schroeder-Lowe protocol

A **term rewriting system (TRS)** is used to model the protocol steps (cf. Figure 3.8).

1. ***x* initiates a communication with *y*:**
   goal(x,y) $\longrightarrow$ U(LHS, mesg(x, y, encr(pubkey(y), x, cons(N(x, y), cons(x, null)))))

2. ***agt*(*u*) answers to NS1 by NS2:**
   mesg(x, agt(u), encr(pubkey(agt(u)), z, cons(v, cons(agt(x2), null)))) $\longrightarrow$ U(LHS, mesg(agt(u), agt(x2), encr(pubkey(agt(x2)), agt(u), cons(v, cons(N(agt(u), agt(x2)), cons(agt(u), null))))))

3. ***agt*(*y*) answers to NS2 by NS3:**
   mesg(x, agt(y), encr(pubkey(agt(y)), z2, cons(N(agt(y), agt(z)), cons(u, cons(agt(z), null))))) $\longrightarrow$ U(LHS, mesg(agt(y), agt(z), encr(pubkey(agt(z)), agt(y), cons(u, null))))

4. **after he received NS2, *agt*(*y*) believes that he has initiated a communication with *agt*(*z*) but he talks with *z*2:**
   mesg(x, agt(y), encr(pubkey(agt(y)), z2, cons(N(agt(y), agt(z)), cons(u, cons(agt(z),null))))) $\longrightarrow$ U(LHS, c_init(agt(y), agt(z), z2))

5. **after he received NS3, *agt*(*y*) believes that he talks to *agt*(*z*) but he talks with *z*2:**
   mesg(x, agt(y), encr(pubkey(agt(y)), z2, cons(N(agt(y), z), null))) $\longrightarrow$
   U(LHS, c_resp(agt(y), z, z2))

Figure 3.8: Rules for the Needham-Schroeder-Lowe protocol

[5]the second element of "encr" always refers to the person that encrypts; the intruder cannot usurp honest agent identities to encrypt.

Cumulative rules of the form $l \longrightarrow U(l, new\_information)$ are used, which means that when the term l is found, the term *new_information* is added to l. The first term of the rules, the part before $\longrightarrow$, is a pre-condition (message received or initial configuration (*goal(...)*) that must be satisfied for a message to be sent. The second term is the concatenation of the current configuration (the pre-condition) with the message that will be sent (or established communication at the end of a run; *c_init(...)*, *c_resp(...)*).

Thus in Figure 3.8, the first rule means that when the state *goal(x,y)*, "x wants to communicate with y" is found, the message *mesg(x, y, encr(pubkey(y), x, cons(N(x, y), cons(x, null))))*, "x sends y a nonce, and his name encrypted with the public key of y", is added to the current trace LHS. LHS stands for Left-Hand-Side is equal to the left hand side terms of the rule, it is used to simplify the notation. It means that

$$goal(x, y) \longrightarrow U(LHS, mesg(x, y, encr(pubkey(y), x, cons(N(x, y), cons(x, null)))))$$

is equivalent to:

$$goal(x, y) \longrightarrow U(goal(x, y), mesg(x, y, encr(pubkey(y), x, cons(N(x, y), cons(x, null))))).$$

In the initial automaton, we only express a part of the intruder's abilities. We still need to model that he can: decompose complex information, decrypt information encrypted with the public keys of the untrusted agents and access to all the information exchanged. Those operations are expressed with rewrite rules and are added to the **TRS** (cf. Figure 3.9):

In Figure 3.9, it is apparent that instead of having for example

$$U(cons(x, y), z) \longrightarrow U(LHS, x),$$

we have

$$U(cons(x, y), z) \longrightarrow U(LHS, add(x)).$$

Using *add(x)* instead of x allows us to keep one final state q_{13} for the automaton.

Each time a rule is used, the variables are replaced by states. For example, if *U(cons(x, y), z)* $\longrightarrow$ *U(LHS, x)* and during a computation $x = q_{45}$ then the term

U(LHS, q_{45}) has to be normalized to be added to the current automaton. q_{45} is already a state and so cannot be altered, but the fact that the intruder knows the information recognized by q_{45} still needs to be expressed. For that q_{45} would have to be added to the set of final states as the final states model the network and the knowledge of the intruder. With *add(x)* the problem disappears, the transition $add(q_{45}) \rightarrow q_{13}$ is used during the normalization and is added to the current automaton. The intruder will use the last rewrite rule $add(x) \longrightarrow x$ to access the information and to link the term to the final state q_{13} when needed.

1. **decomposition of complex information - add(x) means that the intruder caught the information x by analyzing the network**
   U(cons(x, y), z) $\longrightarrow$ U(LHS, add(x))
   U(cons(x, y), z) $\longrightarrow$ U(LHS, add(y))

2. **decryption of information encrypted with untrusted agents' public key**
   U(encr (pubkey(agt(0)), y, z), u) $\longrightarrow$ U(LHS, add(z))
   U(encr (pubkey(agt(s(x))), y, z), u) $\longrightarrow$ U(LHS, add(z))

3. **access to all the information exchanged**
   U(mesg(x, y, z), u) $\longrightarrow$ U(LHS, add(z))

4. **access to caught data**
   add(x) $\longrightarrow$ x

Figure 3.9: Rules for the intruder's abilities

Finally in the **TRS**, there are also rules, Figure 3.10, to express the associativity and commutativity (for short AC) of the *U* symbol used in the previous rules.

U(x, U(y, z)) $\longrightarrow$ U(U(x, y), z)
U(U(x, y), z) $\longrightarrow$ U(x, U(y, z))
U(x, y) $\longrightarrow$ U(y, x)

Figure 3.10: AC rules

3.2.2.3 Approximation for the Needham-Schroeder-Lowe protocol

As already said, Genet and Klay decided to approximate the system by studying the exchanges between two trusted agents and another one that is obtained by compounding all other agents in the network (the state q_3 in the automaton of Figure 3.7).

When the approximation function γ is built, it means that for each rewrite rule, there are only nine possible substitutions for the variables corresponding to the sender and the receiver.

For example, for the first rule of the TRS: goal(x,y) $\longrightarrow$ U(LHS, mesg(x,...)).

Let R = goal(x,y) $\longrightarrow$ U(LHS, mesg(x, y, encr(pubkey(y), x, cons(N(x, y), cons(x,null))))), then using Definition 13:

$\gamma(R, q_{13}, \{x = q_3,\ y = q_3\})$ $\gamma(R, q_{13}, \{x = q_4,\ y = q_3\})$ $\gamma(R, q_{13}, \{x = q_5,\ y = q_3\})$
$\gamma(R, q_{13}, \{x = q_3,\ y = q_4\})$ $\gamma(R, q_{13}, \{x = q_4,\ y = q_4\})$ $\gamma(R, q_{13}, \{x = q_5,\ y = q_4\})$
$\gamma(R, q_{13}, \{x = q_3,\ y = q_5\})$ $\gamma(R, q_{13}, \{x = q_4,\ y = q_5\})$ $\gamma(R, q_{13}, \{x = q_5,\ y = q_5\})$

In practice, users do not see those γ (but they can deduce them). They only see the $Norm_\gamma(r\sigma \rightarrow q)$ of the completion algorithm (Algorithm 2). In [GK00a], the $Norm_\gamma(r\sigma \rightarrow q)$ seen by the users are referred to as "approximation" rules.

Figure 3.11 is an example of an "approximation" rule that the user is going to deal with. That rule gives the normalization process applied on the left hand side of the rule R, with the substitution σ = {x=q_4, y=q_5} and the state q_{13}.

[term that must be normalized before it could be added to the current automaton] $\longrightarrow$ [transitions that will be added to the current automaton]

[U(LHS, mesg(q_4,q_5,encr(pubkey(q_5),q_4,cons(N(q_4,q_5), cons(q_4, null))))) $\rightarrow$ q_{13}] $\longrightarrow$
[LHS $\rightarrow$ q_{13} null$\rightarrow$ q_{18} cons(q_4,q_{18}) $\rightarrow$ q_{17} N(q_4,q_5) $\rightarrow$ q_{16} cons(q_{16},q_{17}) $\rightarrow$ q_{15}
pubkey(q_5) $\rightarrow$ q_{14} encr(q_{14},q_4,q_{15}) $\rightarrow$ q_{13} mesg(q_4,q_5,q_{13}) $\rightarrow$ q_{13}]

Figure 3.11: Example of "approximation" rule

When a rewrite rule, R, a substitution, $\sigma = \{x=q_4, y=q_5\}$, and a normalization state, q_{13}, have been identified then γ is computed (Definition 13):

$$\gamma(R, q_{13}, \sigma) = q_{13}q_{13}q_{13}q_{14}q_{15}q_{16}q_{17}q_{18}.$$

A state is linked to each functional position of *U(LHS, mesg(x, y, encr(...)))*, here for example q_{18} to null, q_{17} to cons(x,null), ..., q_{13} to LHS.

Using the normalization process (Definition 12) and using $\gamma(R, q_{13}, \sigma)$, with R1=$mesg(q_4, q_5, encr(pubkey(q_5), q_4, cons(N(q_4, q_5), cons(q_4, null))))$ the transitions to be added to the current automaton are produced as follow:

$$\begin{aligned} Norm_\gamma(U(LHS, R1) \rightarrow q_{13}) \quad &= \{U(q_{13}, q_{13}) \rightarrow q_{13}\} \bigcup Norm_\gamma(LHS \rightarrow q_{13}) \bigcup Norm_\gamma(R1 \rightarrow q_{13}) \\ &\vdots \\ &= \{U(q_{13}, q_{13}) \rightarrow q_{13}, \quad LHS \rightarrow q_{13}, \quad null \rightarrow q_{18}, \\ &\quad cons(q_4,q_{18}) \rightarrow q_{17}, \quad N(q_4,q_5) \rightarrow q_{16}, \\ &\quad cons(q_{16},q_{17}) \rightarrow q_{15}, \quad pubkey(q_5) \rightarrow q_{14}, \\ &\quad encr(q_{14},q_4,q_{15}) \rightarrow q_{13}, \quad mesg(q_4,q_5,q_{13}) \rightarrow q_{13}\} \end{aligned}$$

In this way, the "approximation" rules are generated to normalize each term.

When the completion algorithm identifies a term that must be normalized, an "approximation" rule will be used to normalize the term and add it to the current automaton.

The TRS, used here, is not left-linear. In section 3.2.1 we explained that the completeness (Theorem 4) was guaranteed for non left-linear TRS by keeping deterministic states matched by the non-linear variables.

Here, the non-linear variables match the terms A, B, 0, s(0), s(s(0)), etc. which are recognized by the states q_1, q_2 and q_0. Initially, these states are deterministic and they will stay deterministic if the user does not do any error in his approximation function (for example by linking those states to other terms than the initial ones). Assuming no error from user, we have $\mathcal{R}^*(\mathcal{L}(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{T}_\mathcal{R}\uparrow(\mathcal{A}))$, where $\mathcal{T}_\mathcal{R}\uparrow(\mathcal{A})$ is the approximation automaton computed. One interesting point that goes with our idea of automating the process is that making sure those conditions are satisfied is easy if the "approximation" rules are automatically generated but it requires more attention when it is done manually. If the user makes an error, then the whole

process is flawed.

The TRS is not right linear and it will often be the case for cryptographic protocols within our syntax[6]. In the initial automaton, the state q_{13} will always not satisfy the second condition of the right linearity condition (Definition 16). Moreover the abstraction function γ is not injective. So there will never be:

$$\mathcal{L}(\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A})) \;=\; \mathcal{R}^*(\mathcal{L}(\mathcal{A})).$$

3.2.2.4 Verification

The approximation automaton models the intruder knowledge; thus it is possible to check the confidentiality of a piece of information by making sure that the information is not recognized by the automaton.

For the Needham-Schroeder-Lowe protocol, the confidentiality of the nonces between A and B is required to be confirmed. The secrecy of these nonces will be guaranteed, if they are not valid terms recognized by the current automaton. To do so, the intersection of the approximation automaton with an automaton that recognizes these nonces (cf Figure 3.12) is checked. If the intersection is empty then the property is verified on the approximated model. As "$\mathcal{R}^*(\mathcal{L}(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A}))$", the property is also verified on the concrete system. For this protocol we have an empty intersection. Nevertheless, no conclusion could have been drawn from a non-empty intersection.

The approximation automaton also contains the belief of the agent when a communication is established with the terms *c_init* and *c_resp*. So it is possible to check the authentication property: "If A or B thinks that he communicates with B or A then he really speaks with B or A".

This property will be verified if the approximation automaton does not recognize terms of the from $c\_resp(x, y, z)$ and $c\_init(t, u, v)$, where $y \neq z$ and $u \neq v$ for $x \in \{agt(A),\ agt(B)\}$, $y \in \{agt(A),\ agt(B)\}$, $t \in \{agt(A),\ agt(B)\}$ and $u \in \{agt(A),\ agt(B)\}$. Again the intersection of the approximation automaton

[6] often we will have at least one message of the protocol that gives a rewrite rule of the form: $\ldots \longrightarrow U(LHS, mesg(x, y, \ldots x \ldots))$

Automaton	Not_Secret	
States	q_1 q_2 q_4 q_5 q_{13}	
Final States	q_{13}	
Transitions		
	$A \rightarrow q_1$	$agt(q_1) \rightarrow q_4$
	$B \rightarrow q_2$	$agt(q_2) \rightarrow q_5$
	$U(q_{13}, q_{13}) \rightarrow q_{13}$	
	$N(q_4, q_5) \rightarrow q_{13}$	$N(q_5, q_4) \rightarrow q_{13}$
	$N(q_4, q_4) \rightarrow q_{13}$	$N(q_5, q_5) \rightarrow q_{13}$

Figure 3.12: Nonces between Alice and Bob

with an automaton recognizing the faulty terms (cf Figure 3.13) is checked. The intersection is empty so the property is verified.

Automaton	Wrong_Belief	
States	q_0 q_1 q_2 q_3 q_4 q_5 q_6 q_{13}	
Final States	q_{13}	
Transitions		
	$0 \rightarrow q_0$	
	$s(q_0) \rightarrow q_0$	$agt(q_0) \rightarrow q_3$
	$A \rightarrow q_1$	$agt(q_1) \rightarrow q_4$
	$B \rightarrow q_2$	$agt(q_2) \rightarrow q_5$
	$U(q_{13}, q_{13}) \rightarrow q_{13}$	
	$c\_init(q_4, q_5, q_3) \rightarrow q_{13}$	$c\_init(q_4, q_5, q_4) \rightarrow q_{13}$
	$c\_resp(q_5, q_4, q_3) \rightarrow q_{13}$	$c\_resp(q_5, q_4, q_5) \rightarrow q_{13}$
	$c\_init(q_5, q_4, q_3) \rightarrow q_{13}$	$c\_init(q_5, q_4, q_5) \rightarrow q_{13}$
	$c\_resp(q_4, q_5, q_3) \rightarrow q_{13}$	$c\_resp(q_4, q_5, q_4) \rightarrow q_{13}$

Figure 3.13: Alice and Bob do not really communicate with each other

The same verification done on an approximation automaton computed with ancestor approximation Figure 3.14 give different results. Only the authentication seems to be verified even though the secrecy is been proven for this protocol. This

failure is not a surprise as the approximation applied on the secret information is too large; all the nonces are gathered together so it is impossible to verify if any of them are unknown of the intruder.

$[\, U(LHS, mesg(x, y, encr(pubkey(y), x, cons(N(x, y), cons(x, null))))) \rightarrow q_{13}] \longrightarrow$
$[\, \text{LHS} \rightarrow q_{17} \quad \text{null} \rightarrow q_{18} \quad cons(x, null) \rightarrow q_{19} \quad N(x, y) \rightarrow q_{20} \quad cons(q_{20}, q_{19}) \rightarrow q_{21}$
$pubkey(y) \rightarrow q_{22} \quad encr(q_{22}, x, q_{21}) \rightarrow q_{23} \quad mesg(x, y, q_{23}) \rightarrow q_{24}]$

$[\, U(LHS, mesg(agt(u), agt(x2), encr(pubkey(agt(x2)), agt(u), cons(v, cons(N(agt(u), agt(x2)),$
$cons(agt(u), null)))))) \rightarrow q_{13}] \longrightarrow [\, \text{LHS} \rightarrow q_{25} \quad \text{null} \rightarrow q_{26} \quad agt(u) \rightarrow q_{27} \quad cons(q_{27}, q_{26}) \rightarrow q_{28}$
$agt(x2) \rightarrow q_{29} \quad N(q_{27}, q_{29}) \rightarrow q_{30} \quad cons(q_{30}, q_{28}) \rightarrow q_{31} \quad cons(v, q_{31}) \rightarrow q_{32}$
$pubkey(q_{29}) \rightarrow q_{33} \quad encr(q_{33}, q_{27}, q_{32}) \rightarrow q_{34} \quad mesg(q_{27}, q_{29}, q_{34}) \rightarrow q_{35}\,]$

$[\, U(LHS, mesg(agt(y), agt(z), encr(pubkey(agt(z)), agt(y), u))) \; rightarrow \; q_{13}] \longrightarrow$
$[\, \text{LHS} \rightarrow q_{36} \quad agt(y) \rightarrow q_{37} \quad agt(z) \rightarrow q_{38} \quad pubkey(q_{38}) \rightarrow q_{39} \quad encr(q_{39}, q_{37}, u) \rightarrow q_{40}$
$mesg(q_{37}, q_{38}, q_{40}) \rightarrow q_{41}]$

Figure 3.14: Ancestor approximation for Needham-Schroeder-Lowe protocol

3.3 Conclusion

Genet and Klay's approach is an effective, quick and simple approach for the verification of secrecy and authentication properties, however:

- the approximation function used must be given by hand;
- the user must choose carefully his approximation function in order to guarantee the termination of the computation of $\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A})$. The ancestor approximation cannot be used as it is inefficient to verify secrecy properties.
- if the intersection is not empty, another method must be used to verify the property.

The next chapter will present the improvements made in the course of this research to solve these problems.

Chapter 4

Improvements

In the previous chapter, two approximations of Genet and the method by which they can be used to verify cryptographic protocols were introduced. The method has some drawbacks that make it difficult to use in an industrial context. Two improvements to the approximation approach are introduced in this chapter. The first one is directly linked to the approximation function; a new approximation function is defined. The challenge is to find an approximation function that:

- guarantees the termination of the computation,
- does not require user interactions and
- is suitable for secrecy and authentication verification.

The second improvement is more general, it is an investigation on the combination of two cryptographic protocol verification techniques. The approximation technique will be one of them. The challenge is to find another verification approach such as both approaches are complementary and the passage from one to the other is easy.

4.1 New approximation function

Table 4.1 summarizes the pros and cons of the approximations introduced in the previous chapter.

Approximation	Pros	Cons
Basic approximation (Definition 13)	can verify protocols' properties	computation may not terminate unless users intervene
Ancestor approximation (Definition 14)	computation terminates	too abstract to verify protocols

Table 4.1: Pros and Cons of Genet's approximations

In this section, a new approximation function γ_f (Definition 25), that takes the pros of the basic and the ancestor approximation, is introduced. This approximation guarantees the termination of the computation of the approximation automaton by restricting the set of new states created by the approximation function to a finite set (like the ancestor approximation); Section 4.1.3 shows that the information lost as a result of our approximation does not affect the verification of the secrecy and authentication properties. This approximation also introduces rules to optimize the use of the states. The idea is to use the same state for identical terms. Three particular cases can be identified:

- If the current term t is already recognized by the state q in the current automaton then γ_f links it to a state q.

  For example, assume $N(q_5, q_5)$ is recognized by the state q_{14} in the current automaton. If this term is met again during the completion it will be linked to q_{14}.

- If the current term t loops on itself, then the same state is used every time it loops during the completion.

  For example, assume $encr(\ldots, \ldots, encr(\ldots, \ldots, \ldots))) \rightarrow q_{22}$ has been normalized to $encr(q_7, q_5, q_{16}) \rightarrow q_{22}$. If $encr(\ldots, \ldots, encr(q_7, q_5, q_{16})) \rightarrow q_{22}$ has to be normalized, then $encr(q_7, q_5, q_{22}) \rightarrow q_{22}$ is produced, $encr(q_7, q_5, q_{16})$ is linked to the state q_{22}.

- If two identical terms are found, the new normalization links them to the same state.

For example, if $cons(N(q_5, q_5), N(q_5, q_5)) \rightarrow q_{22}$ has to be normalized, then $cons(q_{16}, q_{16}) \rightarrow q_{22}$ is produced; both $N(q_5, q_5)$ are linked to q_{16}.

4.1.1 Approximation function γ_f

In order to limit the number of new states, an intermediate function β (Definition 21) that introduces a finite number of new states is used. We make sure that the approximation function γ_f (Definition 25) only uses the states of the initial automaton and those of β (Proposition 8).

The way it works is described below:

1. a set of γ_f that must be computed: $\gamma_f(x \rightarrow s(s(x)), q_0, \{x = q_1\})$, $\gamma_f(x \rightarrow s(s(x)), q_0, \{x = q_2\})$, $\gamma_f(x \rightarrow s(s(x)), q_0, \{x = q_3\})$, etc.

2. we have $\beta(x \rightarrow s(s(x)), q_0, \{x = x\}) = q$. It gives the state that can be used in γ_f for $x \rightarrow s(s(x))$, q_0 and any substitutions of x

3. instead of introducing a new state for every γ_f, and if there is no reason for

$$\begin{aligned} \gamma_f(x \rightarrow s(s(x)), q_0, \{x = q_1\}) \quad & \neq \gamma_f(x \rightarrow s(s(x)), q_0, \{x = q_2\}) \\ & \neq \gamma_f(x \rightarrow s(s(x)), q_0, \{x = q_3\}) \\ & \neq \ etc, \end{aligned}$$

as for the user those terms normalized using γ_f are used in the same way in the rest of the system, then

$$\begin{aligned} \gamma_f(x \rightarrow s(s(x)), q_0, \{x = q_1\}) \quad & = \gamma_f(x \rightarrow s(s(x)), q_0, \{x = q_2\}) \\ & = \gamma_f(x \rightarrow s(s(x)), q_0, \{x = q_3\}) \\ & = \ldots \\ & = q, \text{ using } \beta. \end{aligned}$$

In the above example the γ_f is an ancestor approximation, as no matter what the substitution is, the same state is used. But the β function used to define the γ_f's states in general is more complex so the γ_f will not always be an ancestor approximation.

It is important for an agent to distinguish information created by him from that created by other participants. The TRS allows this by typing the messages. For

example, if an agent recognizes his nonce, then the notation N(agt(x), agt(y)) is found in the TRS rule. When this is a nonce created by someone else, the notation N(w, z) is used. Moreover there is no need for an agent to distinguish the unknown information of different exchanges of the same message. The agent replies to messages, not to unknown information. Those messages contain information about the presumed sender (i.e. a name or a nonce created by the receiver to communicate with a particular agent). Thus whatever is the value of the unknown information received; the same message will be sent.

Thus it is possible to identify two categories of states, those linked to a specific term (this is the case when the term contains variables that can be substituted by terms of arity zero) and those linked to a set of terms. The function β is going to define states according to this last concept.

Initially the number of rewrite rules, the number of terms of arity zero and the number of states linked to those terms are finite. For each rewrite rule, β will be computed for all possible substitutions of variables by states linked to terms of arity zero, plus the identity (thus a variable x can be substituted by itself). For example, if we have the term $N(x, y)$ and the state q_1 that is linked to a term of arity zero then β is computed for: $N(q_1, q_1)$, $N(q_1, y)$, $N(x, q_1)$ and $N(x, y)$. As substitutions do not allow the mapping from variables to variables, we call our mapping $\beta-substitution$.

Definition 21 *(Intermediate function) Let $\mathcal{Q}$ be the set of automaton's initial states and $\mathcal{Q}_{new}$ be the set of states introduced by the computation. Let $\mathcal{Q}_u^*$ be the set of sequences $q_1 \dots q_k$ of states in $\mathcal{Q}_u$ where $\mathcal{Q}_u = \mathcal{Q} \bigcup \mathcal{Q}_{new}$. Let $\mathcal{R}$ be a term rewriting system containing a finite set of rewrite rules. Let $\mathcal{A} = \{\mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta\}$ be a tree automaton. Let $\mathcal{Q}_{a0} \subset \mathcal{Q}$ be the set of states corresponding to the terms of arity zero and $\mathcal{X}$ be the set of variables used in $\mathcal{R}$. Let $\Theta = \{x = y | (y = x \wedge x \in \mathcal{X}) \vee y \in \mathcal{Q}_{a0}\}$ be a set of $\beta-$substitutions. Let $Pos_{\mathcal{F}}(r) = \{p_1, \dots, p_k\}$ be the set of functional positions in r, $k = Card(Pos_{\mathcal{F}}(r))$ and $p_0 = \epsilon$ where ϵ denotes the root position in the tree when the term is viewed as a tree.*

The function β is a mapping: $\mathcal{R} \times \mathcal{Q}_u \times \Theta \mapsto \mathcal{Q}_u^*$, such that

$$\beta(l \to r, q_0, \theta) = q_1 \dots q_k$$

with $q_i \in \mathcal{Q}_{new}$ for $i \in [1,k]$ by default.

Then for all $i \in [1,k]$:

1. *if $\exists q' \in \mathcal{Q}$ such that $(r\theta|_{p_i}) \to_{\mathcal{A}}^* q'$ then $q_i = q'$;*
2. *if $\exists l' \to r' \in \mathcal{R}$, $\exists z = Card(Pos_{\mathcal{F}}(r'))$, $\exists j \in [0,z]$, $\exists \theta' \in \Theta$, $\exists q'_0 \in \mathcal{Q}_u$, $\exists\, q'_1, \dots, q'_z \in \mathcal{Q}_u$ such that $\beta(l' \to r', q'_0, \theta') = q'_1 \dots q'_z \wedge (r\theta|_{p_i}) = (r'\theta'|_{p_j})$ then $q_i = q'_j$.*

The two rules used in the definition of β optimize the use of the states available, following the idea that the same state should be linked to identical terms:

- the first rule says that if a subterm of a term is already recognized by a state q' in the current automaton, then β also links this subterm to q'.
- the second rule says that two identical subterms are linked to the same state by β.

Proposition 4 *Identical subterms in two β computations (Definition 21) using the same automaton $\mathcal{A}$, the same TRS $\mathcal{R}$, and the same set of states $\mathcal{Q}_u$, are linked to the same state.*

Proof [Proof by contradiction] Let $l \to r$ and $l' \to r'$ be two rules of $\mathcal{R}$. Let θ and θ' be two elements of Θ. Let $q_0, \dots, q_n, q'_0, q'_1, \dots, q'_z$ be states of $\mathcal{Q}_u$ with $n = Card(Pos_{\mathcal{F}}(r))$ and $z = Card(Pos_{\mathcal{F}}(r'))$.

This proposition is proven by showing that for $\beta(l \to r, q_0, \theta) = q_1 \dots q_n$ and $\beta(l' \to r', q'_0, \theta') = q'_1 \dots q'_z$ if two subterms $r\theta|_{p_i}$ and $r'\theta'|_{p_j}$ are such that $r\theta|_{p_i} = r'\theta'|_{p_j}$, then it is impossible to have $q_i \neq q'_j$.

Let us assume that for $\beta(l \to r, q_0, \theta) = q_1 \dots q_n$ and $\beta(l' \to r', q'_0, \theta') = q'_1 \dots q'_z$, $\exists i \in [0,n], j \in [0,k]$ such that $r\theta|_{p_i} = r'\theta'|_{p_j}$ and $q_i \neq q'_j$.

1. If $\exists q' \in \mathcal{A}$ such that $(r\theta|_{p_i}) \rightarrow_{\mathcal{A}}^{*} q'$ then $q_i{=}q'$ with the first optimization rule of Definition 21. Thus as $r\theta|_{p_i}{=}\, r'\theta'|_{p_j}$, $r'\theta'|_{p_j}{=}\, q'$. $\Longrightarrow$ **Contradiction**

2. For i by assumption $\exists l' \rightarrow r' \in \mathcal{R}$, $\exists \theta' \in \Theta$, $\exists q'_1, \ldots, q'_z \in \mathcal{Q}_u$ such that $r\theta|_{p_i}{=}\, r'\theta'|_{p_j}$ and $\beta(l' \rightarrow r', q'_0, \theta') = q'_1 \ldots q'_z$. Thus with the second optimization rule of Definition 21, $q_i \neq q'_j$. (the same result is obtained starting from j). $\Longrightarrow$ **Contradiction**

Following Definition 21 it is impossible to have $r\theta|_{p_i}{=}\, r'\theta'|_{p_j}$ and $q_i \neq q'_j$. So identical subterms in two β computations are linked to the same state. $\diamond$

A direct consequence of Proposition 4 is Proposition 5.

Proposition 5 *Starting from the same automaton $\mathcal{A}$, the same TRS $\mathcal{R}$, and the same set of states $\mathcal{Q}_u$, identical subterms in two β computations (Definition 21) on the same rule using the same substitution are linked to the same state.*

The following algorithm explains how to compute the set containing all the possible approximation functions β. This particular set, Ψ (Definition 22) will be useful later.

Definition 22 *(Set of β) Ψ is the set containing one occurrence of each possible function β defined by Definition 21 and computed as explained by Algorithm 3.*

If $\mathcal{X}$ is the set of variables used in $\mathcal{R}$ and $\mathcal{Q}_{a0}$ is the set of states corresponding to the terms of arity zero, then $\Theta{=}\{x = y |\ (y = x \wedge x \in \mathcal{X}) \vee y \in \mathcal{Q}_{a0}\}$ is a finite set as both $\mathcal{X}$ and $\mathcal{Q}_{a0}$ are finite sets.

By definition, the set of $\beta-$substitutions Θ and the set of rules $\mathcal{R}$ stay the same during the computation of Ψ (no rewrite rule or new $\beta-$substitution is added during the computation). Thus the computation of Ψ can be broken up into two "for" loops that explore all the triplets (r, q, θ) for $r \in \mathcal{R}$, $q \in \mathcal{Q}_u$ and $\theta \in \Theta$.

Algorithm 3 *Starting with two empty sets, $\mathcal{Q}_{new}$ and $\mathcal{S}$, Ψ is computed as follow:*

1. for all ($r \in \mathcal{R}$, $q \in \mathcal{Q}$, $\theta \in \Theta$) do

if $((\beta(r, q, \theta) = q_1 ... q_n) \notin \mathcal{S})$ then

$\mathcal{S} = \{\beta(r, q, \theta) = q_1 ... q_n\} \cup \mathcal{S}$

for all $1 \leq i \leq n$ *do*

if $((q_i \notin \mathcal{Q}) \wedge (q_i \notin \mathcal{Q}_{new}))$ *then*

$\mathcal{Q}_{new} = \{q_i\} \cup \mathcal{Q}_{new}$

fi

done

fi

done

2. *for all* $(r \in \mathcal{R},\ q \in \mathcal{Q}_{new},\ \theta \in \Theta)$ *do*

if $((\beta(r, q, \theta) = q_1 ... q_n) \notin \mathcal{S})$ *then*

$\mathcal{S} = \{\beta(r, q, \theta) = q_1 ... q_n\} \cup \mathcal{S}$

fi

done

3. $\Psi = \mathcal{S}$

Basically, the first loop adds new states to $\mathcal{Q}_{new}$, the set of new states, and β functions to $\mathcal{S}$, the set of β functions. The second loop, *for all* $(r \in \mathcal{R},\ q \in \mathcal{Q}_{new},\ \theta \in \Theta)$, only adds β functions to $\mathcal{S}$ as no new states are created because of Proposition 5 (the set Θ is the same for both loops).

Proposition 6 *The computation of the set* Ψ *terminates.*

Proof[The computation of Ψ terminates]

By definition the sets $\mathcal{R}$, $\mathcal{Q}$ and Θ are finite. Thus the computation of the loop *for all* $(r \in \mathcal{R},\ q \in \mathcal{Q},\ \theta \in \Theta)$ terminates and introduces a finite set of new states $\mathcal{Q}_{new}$.

The computation of the loop *for all* $(r \in \mathcal{R},\ q \in \mathcal{Q}_{new},\ \theta \in \Theta)$ also terminates as $\mathcal{R}$, $\mathcal{Q}_{new}$ and Θ are finite sets.

Both loops terminate, so the computation of Ψ terminates. ◇

Let n be the number of rewrite rules, s be the number of states, s_1 be the number of states linked to a term of arity zero, s_{new} be the number of new states introduced by the computation, v be the number of variables and nb_Ψ the number of β in Ψ computed. The complexity C of Algorithm 3 is the complexity C1 of the first loop plus the complexity C2 of the second loop:

- C1=$s * n * (\sum_{i=0}^{v-1}(s_1^{v-1} * (i+1)) + 1) \ * \ \sum_{j=0}^{nb_\Psi - 1} j \ * \ s * \sum_{l=0}^{s_{new}-1} l$
- C2=$s_{new} * n * (\sum_{i=0}^{v-1}(s_1^{v-1} * (i+1)) + 1) \ * \ \sum_{j=0}^{nb_\Psi - 1} j$
- C=C1+C2

It is possible to rewrite this expression using only n by saying s=n+a, s_1=n+b, s_{new}=n+c, v=n+d and nb_Ψ=n+e where a,b,c,d,e are real numbers. A simple expression can then be deduced C=$O(n^n)$. Thus the complexity of Algorithm 3 is exponential.

The introduction of this section states that β were used to reduce the number of states used in the new approximation function γ_f. For example, by assuming $\beta(x \to s(s(x)), q_0, \{x = x\}) = q$, it was said that

$$\begin{aligned}\gamma_f(x \to s(s(x)), q_0, \{x = q_1\}) &= \gamma_f(x \to s(s(x)), q_0, \{x = q_2\}) \\ &= \gamma_f(x \to s(s(x)), q_0, \{x = q_3\}) \\ &= \ldots \\ &= \beta(x \to s(s(x)), q_0, \{x = x\}).\end{aligned}$$

To give the formal definition of γ_f, the relation $\ll_{max}$ that links γ_f and β must be introduced. $\ll_{max}$ and $\ll$ are relations between the elements of Θ and of $\Sigma(\mathcal{Q}_u, \mathcal{X})$. $\Sigma(\mathcal{Q}_u, \mathcal{X})$ is a set of substitutions for which all the variables are substituted by states and Θ is a set of $\beta-$substitutions for which the variables can either stay the same or be substituted by states linked to terms of arity zero, thus $\ll_{max}$ and $\ll$ will be linking one or more substitutions from $\Sigma(\mathcal{Q}_u, \mathcal{X})$ to one $\beta-$substitution from Θ.

Definition 23 *(Substitution inclusion) Let $\mathcal{R}$ be a term rewriting system. Let $\mathcal{X}$ be the set of variables in $\mathcal{R}$. Let $\mathcal{A} = \{\mathcal{F}, \mathcal{Q}_u, \mathcal{Q}_f, \Delta\}$ be a tree automata.*

Let $\theta \in \Theta$ and $\sigma \in \Sigma(\mathcal{Q}_u, \mathcal{X})$ be two substitutions of the same variables. There is the relation $\ll$ between θ and σ, $\theta \ll \sigma$, if for all $(x = q) \in \sigma$ with $x \in \mathcal{X}$ and $q \in \mathcal{Q}_u$ then:

- $(x = q) \in \theta$ *or*
- $(x = x) \in \theta$.

Example 15 *Let $\sigma = \{x = q_1, y = q_2, z = q_3\}$, $\theta_1 = \{x = q_1, y = q_2, z = q_3\}$ and $\theta_2 = \{x = x, y = q_2, z = q_3\}$, assuming that q_1, q_2 and q_3 are states linked to some terms of arity zero (this is to satisfy the definition of Θ):*

- $\theta_1 \ll \sigma$ *as:*
  - *for $(x = q_1) \in \sigma$ we have $(x = q_1) \in \theta_1$, and*
  - *for $(y = q_2) \in \sigma$ we have $(y = q_2) \in \theta_1$, and*
  - *for $(z = q_3) \in \sigma$ we have $(z = q_3) \in \theta_1$.*
- $\theta_2 \ll \sigma$ *as:*
  - *for $(x = q_1) \in \sigma$ we have $(x = x) \in \theta_2$, and*
  - *for $(y = q_2) \in \sigma$ we have $(y = q_2) \in \theta_2$, and*
  - *for $(z = q_3) \in \sigma$ we have $(z = q_3) \in \theta_1$.*

Proposition 7 *Let $\mathcal{R}$ be a term rewriting system. Let $\mathcal{X}$ be the set of variables in $\mathcal{R}$. Let $\mathcal{A} = \{\mathcal{F}, \mathcal{Q}_u, \mathcal{Q}_f, \Delta\}$ be a tree automata.*

For all $\sigma \in \Sigma(\mathcal{Q}_u, \mathcal{X})$ there exists at least one $\theta \in \Theta$ such that $\theta \ll \sigma$.

Proof Let $\mathcal{R}$ be a term rewriting system. Let $\mathcal{X}$ be the set of variables in $\mathcal{R}$. Let $\mathcal{A} = \{\mathcal{F}, \mathcal{Q}_u, \mathcal{Q}_f, \Delta\}$ be a tree automata. Let $\mathcal{Q}_{a0} \subseteq \mathcal{Q}_u$ be the set of states corresponding to the terms of arity zero.

For all $\sigma \in \Sigma(\mathcal{Q}_u, \mathcal{X})$ and for all $(x = q) \in \sigma$ then $q \in \mathcal{Q}_{a0}$ or $q \notin \mathcal{Q}_{a0}$. For any $\sigma \in \Sigma(\mathcal{Q}_u, \mathcal{X})$, it is possible to generate a $\beta-$substitution θ for all $(x = q) \in \sigma$ if:

- $q \in \mathcal{Q}_{a0}$ then $(x = q) \in \theta$ and

- $q \notin \mathcal{Q}_{a0}$ then $(x = x) \in \theta$

θ only contains identity substitutions, $(x = x)$, or substitutions of variables by states linked to terms of arity zero, as $(x = q)$ only if $q \in \mathcal{Q}_{a0}$. So $\theta \in \Theta$.

Therefore for all $(x = q) \in \sigma$ with $x \in \mathcal{X}$ and $q \in \mathcal{Q}_u$:

- $(x = q) \in \theta$ or
- $(x = x) \in \theta$.

This means that $\theta \ll \sigma$.

So for all $\sigma \in \Sigma(\mathcal{Q}_u, \mathcal{X})$ there exists at least one $\theta \in \Theta$ such that $\theta \ll \sigma$. $\diamond$

Definition 24 *(Substitution maximum inclusion) Let $\mathcal{R}$ be a term rewriting system containing a finite set of rules. Let $\mathcal{X}$ be the set of variables in $\mathcal{R}$. Let $\mathcal{A} = \{\mathcal{F}, \mathcal{Q}_u, \mathcal{Q}_f, \Delta\}$ be a tree automata. Let $\mathcal{Q}_{a0} \subseteq \mathcal{Q}_u$ be the set of states corresponding to the terms of arity zero. Let $\sigma \in \Sigma(\mathcal{Q}_u, \mathcal{X})$ be a substitution.*

For all $\theta \in \Theta$ such that $\theta \ll \sigma$ then $\theta \ll_{max} \sigma$ if and only if for all $(x = q) \in \sigma$:

- *if $q \in \mathcal{Q}_{a0}$ then $(x = q) \in \theta$ and*
- *if $q \notin \mathcal{Q}_{a0}$ then $(x = x) \in \theta$.*

The relation $\ll_{max}$ is used to guarantee that amongst all the possible θ, such that $\theta \ll \sigma$, the one that satisfies this relation is the one that has the maximum number of identical substitutions of variables by states with σ.

Example 16 *Let $\sigma_1 = \{x = q_1, y = q_2, z = q_3\}$, $\sigma_2 = \{x = q_2, y = q_2, z = q_3\}$, $\theta_1 = \{x = q_1, y = q_2, z = q_3\}$, $\theta_2 = \{x = x, y = q_2, z = q_3\}$ and $\theta_3 = \{x = x, y = q_2, z = z\}$, assuming that q_1, q_2 and q_3 are states linked to some terms of arity zero (this is to satisfy the definition of Θ):*

- *$\theta_1 \ll \sigma_1$, $\theta_2 \ll \sigma_1$ and $\theta_3 \ll \sigma_1$ but θ_1 has the maximum of identical elements with σ_1 so $\theta_1 \ll_{max} \sigma_1$.*
- *$\theta_2 \ll \sigma_2$ and $\theta_3 \ll \sigma_2$ but θ_2 has more elements in common with σ_2 than θ_3 so $\theta_2 \ll_{max} \sigma_2$.*

Now that β, Ψ and $\ll_{max}$ have been introduced, the approximation function γ_f can be defined.

Definition 25 *(New approximation function) Let $\mathcal{R}$ be a term rewriting system. Let $\mathcal{A} = \{\mathcal{F}, \mathcal{Q}_u, \mathcal{Q}_f, \Delta\}$ be a tree automata. Let $Pos_{\mathcal{F}}(r) = \{p_1, \ldots, p_k\}$ be the set of functional positions in r, $k = Card(Pos_{\mathcal{F}}(r))$ and $p_0 = \epsilon$ where ϵ denotes the root position in the tree when the term is viewed as a tree. Let $\Sigma(\mathcal{Q}_u, \mathcal{X})$ be the set of substitutions of variables in $\mathcal{X}$ by the states in $\mathcal{Q}_u$. Let β be a function corresponding to Definition 21. Let Ψ be the set defined by Definition 22.*

An approximation function γ_f is a mapping γ_f: $\mathcal{R} \times \mathcal{Q}_u \times \Sigma(\mathcal{Q}_u, \mathcal{X}) \mapsto \mathcal{Q}_u^*$ with

$$\gamma_f(l \to r, q, \sigma) = q_1 \ldots q_k.$$

For the $\theta \in \Theta$ such that $\theta \ll_{max} \sigma$ (possible because of Proposition 7 and Definition 24) there exists

$$\beta(l \to r, q, \theta) = q'_1 \ldots q'_k \in \Psi;$$

thus by default $q_i = q'_i$ for all $i \in [1, k]$.

Then for all $i \in [1, k]$:

1. *if $\exists j \in [0, k]$, $\exists z = Card(Pos_{\mathcal{F}}(r|_{p_j}))$, $\exists f \in \mathcal{F}$, $\exists q'_0 \in \mathcal{Q}_u$, $\exists q''_1, \ldots, q''_z \in \mathcal{Q}_u$ such that:*
   - *$q'_0 = q$ and*
   - *$r|_{p_j} = f(t_1, \ldots, t_z)$ and*
   - *$r\sigma|_{p_i} = f(q''_1, \ldots, q''_z)$ and*
   - *for all $h \in [1, z]$ there exists $m \in [1, k]$ such that $q''_h = q'_m \wedge t_h = r|_{p_m}$*

   then $q_i = q'_j$;
2. *if $\exists j \in [1, k]$ such that $(i \neq j \wedge (r\sigma|_{p_i}) = (r\sigma|_{p_j}))$ then $q_j = q_i$.*

The default generation of the sequence of states makes sure that γ_f is produced using β. Then as in Definition 21, two rules are employed to modeling the use of the states available following the principal that during the normalization, two identical terms will be linked to the same state.

The first rule is used so that, after applying the substitution, a term to be normalized and its normalization are linked to the same state.

For example, assume that $agt(q_1) \to q_3$ and $agt(q_2) \to q_4$ are transitions of the current automaton $\mathcal{A}$.

> If $\beta(... \to cons(N(agt(a),\ agt(b)),\ N(x,\ y)),\ q,\ \{a = q_1, b = q_2, x = x, y = y\}) = q_5q_3q_4q_6$
>
> If $\{a = q_1, b = q_2, x = x, y = y\} \ll_{max} \{a = q_1, b = q_2, x = q_3, y = q_4\}$
>
> Then by default $\gamma_f(... \to cons(N(agt(a),\ agt(b)),\ N(x,\ y)),\ q,\ \{a = q_1,\ b = q_2,\ x = q_3,\ y = q_4\}) = q_5q_3q_4q_6$.

But the substitutions of x by q_3 and y by q_4 reveal the term $N(q_3, q_4)$ which is the normalized form of $N(agt(q_1), agt(agt(q_2))$ as $agt(q_1) \to q_3$ and $agt(q_2) \to q_4$. The first rule makes sure that the same state is linked to both of them, that is why $\gamma_f(...,\ q,\ \{a = q_1,\ b = q_2,\ x = q_3,\ y = q_4\}) = q_5q_3q_4q_5$ after optimization.

The second rule is used to ensure that two terms, which become identical after substitutions, are linked to the same state.

For example, assume that $agt(q_1) \to q_3$ and $agt(q_2) \to q_4$ are transitions of the current automaton $\mathcal{A}$.

> If A function β with substitutions $\{a = a,\ b = b,\ x = x,\ y = y\} \ll_{max} \{a = q_3,\ b = q_4,\ x = q_3,\ y = q_4\}$ had been computed and yields $\beta(... \to cons(N(a,\ b),\ N(x,\ y)),\ q,\ \{a = a,\ b = b,\ x = x,\ y = y\}) = q_5q_6$.
>
> Then by default $\gamma_f(... \to cons(N(a,\ b),\ N(x,\ y)),\ q,\ \{a = q_3,\ b = q_4,\ x = q_3,\ y = q_4\}) = q_5q_6$.

But the substitutions of a by q_3, b by q_4, x by q_3 and y by q_4 reveal the two identical terms $N(q_3, q_4)$ which had been linked to q_5 and q_6 . The second simplification rule

makes sure that those identical terms are linked to the same state, so:

$$\gamma_f(..., q, \{a = q_3, b = q_4, x = q_3, y = q_4\}) = q_5 q_5.$$

After having introduced the new approximation function, its properties can be studied.

Proposition 8 *The set of states used by γ_f is bounded by the set of states of Ψ.*

This proposition is proven by showing that it is impossible to have a rule $l \to r$, a state q and a substitution σ that map to a sequence of states in which one of the states in the sequence is new (not used in Ψ and not in the initial automaton).

Proof[Proof by contradiction] The set $\mathcal{Q}_u$ contains the states of the initial automaton and those introduced by Ψ.

Let us assume $\exists l \to r \in \mathcal{R}, \exists q \in \mathcal{Q}_u, \exists \sigma \in \Sigma(\mathcal{Q}_u, \mathcal{X})$ such that $\gamma_f(l \to r, q, \sigma) = q_1 ... q_k$, and there is $q_i \in \{q_1, ..., q_k\}$ such that $q_i \notin \mathcal{Q}_u$ and $k = Card(Pos_{\mathcal{F}}(r))$.

As a consequence of Proposition 7 and Definition 24 there exists a $\theta \in \Theta$ such that $\theta \ll_{max} \sigma$. By default, we have in Definition 25:

$$\gamma_f(l \to r, q, \sigma) = q'_1 \dots q'_k = \beta(l \to r, q, \theta)$$

with $(\beta(l \to r, q, \theta) = q'_1 ... q'_k) \in \Psi$ and $q_j = q'_j$ for $j \in [1, k]$. Moreover with the definition of Ψ (Definition 22), we have $q_1, ..., q_k$ **are use in** $\Psi \Longrightarrow$ **Contradiction**

The default sequence of states contradicts the assumption.

But may be the optimization rules introduce the new state and then validate the assumption. Both optimization rules substitute a state of the default sequence by another one:

- in the first rule, the state q_i is substituted by q'_j for $j \in [1, k]$ or by q'_0 with $q'_0 = q$. By definition $q'_1, \dots, q'_k \in \mathcal{Q}_u$ and $q \in \mathcal{Q}_u$. $\Longrightarrow$ **Contradiction**
- in the second rule, the state q_j is substituted by the state q_i.

If q_i has not been modified by the first rule, $q_i = q_i' \in \mathcal{Q}_u$. $\Longrightarrow$ **Contradiction**

If q_i has been modified by the first rule then

$$q_i \in \{q, q_1', \ldots, q_k'\} \text{ and } \{q, q_1', \ldots, q_k'\} \subseteq \mathcal{Q}_u. \Longrightarrow \textbf{Contradiction}$$

So it is impossible to create a $\gamma_f(l \to r, q, \sigma) = q_1 ... q_n$ where there is $q_i \in \{q_1, ..., q_n\}$ such that $q_i \notin \mathcal{Q}_u$. ◇

Proposition 9 *Let $\mathcal{R}$ be a TRS. Let γ_f be an approximation function such that γ_f is defined by Definition 25. Let $\mathcal{A}_0$=$\{\mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta\}$ and $\mathcal{A}_{fk}$ be two tree automata such that $\mathcal{A}_{fk}$ is computed using the completion algorithm (Definition 2) with $\mathcal{R}$, γ_f and the initial automaton $\mathcal{A}_0$. The completion algorithm first looks for a critical pair, term T not recognized by the current automaton and a state S that recognizes the term that gave T by rewriting. Then the normalized form of $T \to S$ and all the intermediate transitions required by the normalization are added to the current automaton by the completion to produce the new automaton.*

For all non left-linear rules $l \to r \in \mathcal{R}$, for all non linear variables x of l, for all states $q_1, ..., q_n \in \mathcal{Q}$ that substitute x, if either $q_1 = ... = q_n$ or $\mathcal{L}(\mathcal{A}_{fk}, q_1) \bigcap ... \bigcap \mathcal{L}(\mathcal{A}_{fk}, q_n) = \emptyset$ then: $\mathcal{R}^*(\mathcal{L}(\mathcal{A}_0)) \subseteq \mathcal{L}(\mathcal{A}_{fk})$.

Proposition 9 is proven the same way Theorem 4 was (cf. Appendix C). But it could have also been proven using γ Definition 13 and Theorem 4.

Proof[Idea of the proof using γ] Let $\mathcal{A}_k = \{\mathcal{F}, \mathcal{Q}, \mathcal{Q}_{final}, \Delta\}$ be the automaton produced with γ and let $\mathcal{A}_{fk} = \{\mathcal{F}, \mathcal{Q}_f, \mathcal{Q}_{final}, \Delta_f\}$ be the automaton produced with γ_f. The computation of both automata starts from the same initial automaton $\mathcal{A}_0$ and from the same TRS $\mathcal{R}$.

Let $t \in \mathcal{L}(\mathcal{A}_\gamma)$:

- If t is recognized by the initial automaton then $t \in \mathcal{L}(\mathcal{A}_{\gamma_f})$
- Every term not recognized by the initial automaton will be recognized by a unique sequence of transitions of Δ as γ creates new states for every completion step. Moreover the completion process does not modify the set of final states

so every new term recognized by the language must have been added by the completion. Thus t was added to $\mathcal{L}(\mathcal{A}_\gamma)$ by a completion step.

Assuming that $\gamma(l \to r, q, \sigma) = q_1...q_k$ was used to add t into the language, it is possible to find a θ such that $\theta \ll \sigma$ (Definition 24 and Proposition 7). With θ it is possible to look into γ_f to find a σ' such that $\theta \ll \sigma'$ and $\gamma_f(l \to r, q, \sigma') = q'_1...q'_k$. This means t was also added to $\mathcal{L}(\mathcal{A}_{\gamma_f})$ by the completion.

Therefore, for any term t of $\mathcal{L}(\mathcal{A}_\gamma)$ such that $t \to *_{\mathcal{A}_\gamma} q$, it is possible to find an equivalent sequence of transitions of Δ_f such that $t \to_{\mathcal{A}_{\gamma_f}} q$. So $\mathcal{L}(\mathcal{A}_\gamma) \subseteq \mathcal{L}(\mathcal{A}_{\gamma_f})$ is guaranteed. As Theorem 4 is guaranteed for $\mathcal{L}(\mathcal{A}_\gamma)$, the completeness is also guaranteed for $\mathcal{L}(\mathcal{A}_{\gamma_f})$ under the same assumptions. ◇

Theorem 7 *Let $\mathcal{R}$ be a TRS containing a finite number of rules. Let $\mathcal{A}_{f0} = \{\mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta\}$ be a tree automaton. Let γ_f be an approximation function such that γ_f is defined by Definition 25. If the number of rules in $\mathcal{R}$ and the number of states in $\mathcal{Q}$ are finite then the computation of $\mathcal{A}_{fk}$ will stop.*

The completion algorithm terminates when no more new transitions can be added to the current automaton. So, like for the ancestor approximation (Definition 14), this theorem that states that the approximation automaton is finite is proven by showing that a finite number of states are introduced by the completion.

Proof Starting from a TRS $\mathcal{R}$ containing a finite number of rules, an initial automaton $\mathcal{A}_{f0}$= {$\mathcal{F}$, $\mathcal{Q}$, $\mathcal{Q}_f$, Δ} and an approximation function γ_f, $\mathcal{A}_{f(i+1)}$ is built from $\mathcal{A}_{fi}$[1] by:

1. searching for a critical pair $(r\sigma, q)$ with a state $q \in \mathcal{Q}_u$, a rewrite rule $l \to r$ and a substitution $\sigma \in \Sigma(\mathcal{Q}_u, \mathcal{X})$ such that $l\sigma \to^*_{\mathcal{A}_{fi}} q$ and $r\sigma \to^*_{\mathcal{A}_{fi}} q$
2. $\mathcal{A}_{f(i+1)} = \mathcal{A}_{fi} \cup Norm_{\gamma_f}(r\sigma \to q)$.

The sets $\mathcal{R}$, $\mathcal{X}$, $\mathcal{Q}_u$ and $\Sigma(\mathcal{Q}_u, \mathcal{X})$ are used by the completion.

[1] it is assumed $\mathcal{A}_{fi}$= {$\mathcal{F}$, $\mathcal{Q}_u$, $\mathcal{Q}_f$, Δ'} where $\mathcal{Q}_u = \mathcal{Q} \bigcup \mathcal{Q}_{new}$ and $\mathcal{Q}_{new}$ the set of states introduced by γ_f

By assumption, the set of rules, $\mathcal{R}$, and the set of variables used in $\mathcal{R}$, $\mathcal{X}$, are finite.

By looking at the above definition $q \in \mathcal{Q}_u$ and $\mathcal{Q}_u = \mathcal{Q} \bigcup \mathcal{Q}_{new}$. $\mathcal{Q}$ is the set of initial states and is finite by assumption. $\mathcal{Q}_{new}$ is the set of states introduced during the completion by γ_f. Proposition 8 says that the states used by γ_f are bounded by those of Ψ and Ψ introduces a finite number of new states (cf. proof of termination of Algorithm 3), so $\mathcal{Q}_{new}$ is a finite set. This means that $\mathcal{Q}_u$ is a finite set. $\diamond$

Figure 4.1 summarizes the way $\mathcal{A}_{fk}$ is computed. First, the set Ψ is computed according to the Algorithm 22. Then the completion process (Algorithm 2) is applied. It uses the γ_f (Definition 25), the normalization process (Definition 12), the finite set of states $\mathcal{Q}_u$ (states of the initial automaton and those introduced by Ψ), the finite set of rewriting rules $\mathcal{R}$ and the finite set of substitution $\Sigma(\mathcal{Q}_u, \mathcal{X})$.

Figure 4.2 is a more detailed version of Figure 4.1 with the algorithms and definitions detailed. First the sets Ψ and $\mathcal{Q}_u$ are computed, for all the substitutions θ. The first loop using the set $\mathcal{Q}$ introduces a finite set of new states and a finite number of β. The second loop using $\mathcal{Q}_{new}$, the set of new states introduced by the first loop, only adds β functions to the set of β functions. When the sets Ψ and $\mathcal{Q}_u$ are available, $\mathcal{A}_{fk}$ can be computed. Critical pairs are searched:

- when a pair is found, a γ_f for the term to normalize is computed. With that γ_f a normalization of the term is deduced. Finally the current automaton is updated to recognize both terms of the critical pair.
- when no more critical pairs are found (in the worst case, when all the combinations of rewrite rules, states and substitutions have been explored), the computation stops.

It can be seen on Figure 4.2, the computation of $\mathcal{A}_{fk}$ relies on the critical pairs search which apply all the possible substitutions $\sigma \in \Sigma(\mathcal{Q}_u, \mathcal{X})$ for every automaton. Thus the complexity of the computation of $\mathcal{A}_{fk}$ is exponential.

When we will be verifying protocol, we will have between two and four states linked to terms of arity zero. the computation time of Ψ and $\mathcal{Q}_u$ is then only depending of size of the term rewriting system and the number of variables. The

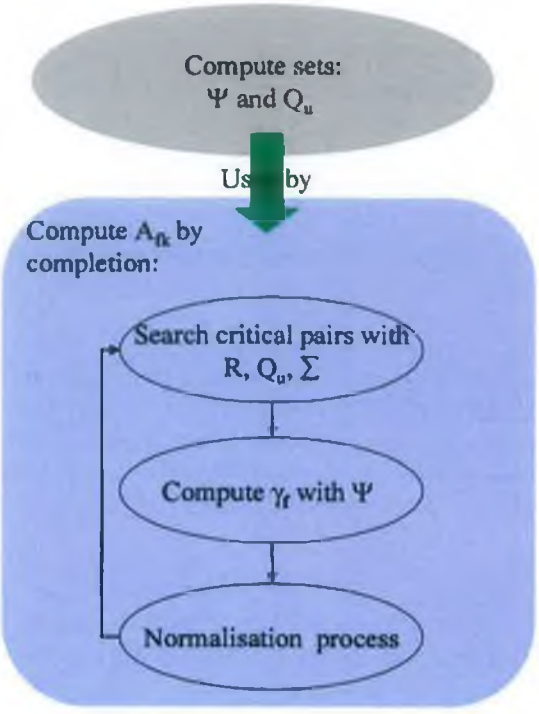


Figure 4.1: Abstract computation of the approximation automaton $\mathcal{A}_{fk}$ with the TRS $\mathcal{R}$, the initial automaton $\mathcal{A}_0 = \{\mathcal{F}, \mathcal{Q}_u, \mathcal{Q}_f, \Delta\}$, the set of variables $\mathcal{X}$ and initially $\mathcal{Q}_{new} = S = \emptyset$

protocols that we will be using in the next chapter, will be small or medium size regarding the number of steps and the size of the messages, thus the computation of Ψ and $\mathcal{Q}_u$ is done quickly and their sizes are small. So the sizes of the sets used in the completion are reasonable so the computation of the approximation automaton is done quickly.

4.1.2 Example

This example was taken from [Gen98b] and presented in Chapter 3 to show that the computation of the approximation with γ could run forever. Here, it is used to show that with γ_f the same computation terminates.

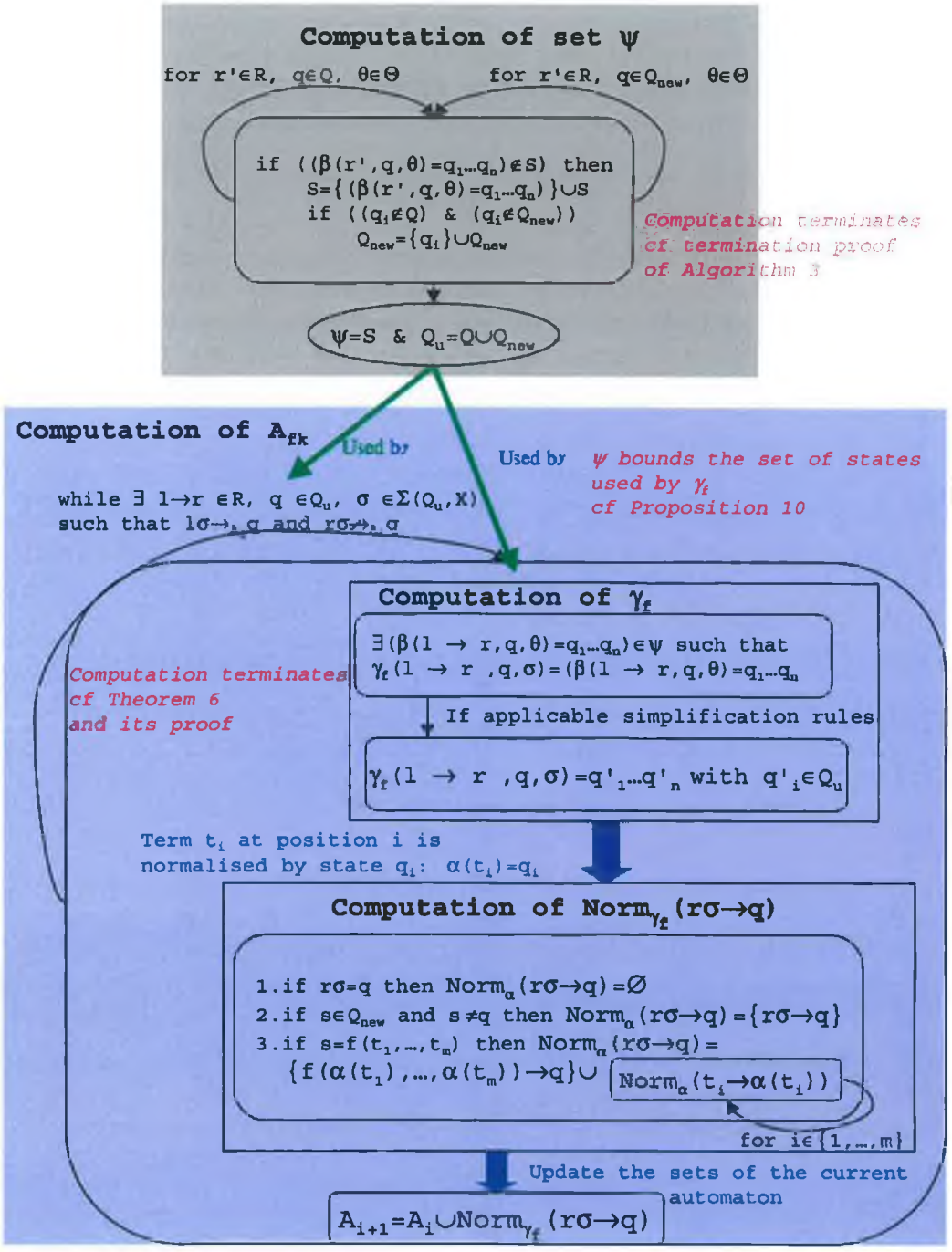


Figure 4.2: Detailed computation of the approximation automaton $\mathcal{A}_{fk}$ with the TRS $\mathcal{R}$, the initial automaton $\mathcal{A}_0 = \{\mathcal{F}, \mathcal{Q}_u, \mathcal{Q}_f, \Delta\}$, the set of variables $\mathcal{X}$ and initially $Q_{new} = S = \emptyset$

Example 17 *Let* $\mathcal{A} = \{\mathcal{F}, \{q_0, q_1, q_2\}, \{q_1\}, \Delta\}$ *be a tree automaton where :*

- $\mathcal{F} = \{app : 2, cons : 2, nil : 0, a : 0\}$,
- Δ=$\{app(q_0, q_0) \rightarrow q_1,\ cons(q_2, q_1) \rightarrow q_0,\ nil \rightarrow q_0,\ nil \rightarrow q_1,\ a \rightarrow q_2\}$, $rl = app(cons(x, y), z) \rightarrow cons(x, app(y, z))$,
- $\mathcal{R} = \{rl\}$, *and*
- γ_f *(Definition 25) the approximation function mapping every tuple* (rl, q, σ) *to one state* $(cons(x, app(y, z))$ *of* rl *contains only one subterm* $app(y, z))$.

After having computed all the possible β, *the Genet and Klay process is used to compute* $\mathcal{A}_{i+1}$ *from* $\mathcal{A}_i$*:*

1. *we have* $app(cons(q_2,\ q_1),\ q_0) \rightarrow^*_{\mathcal{A}} q_1$ *and* $cons(q_2,\ app(q_1,\ q_0)) \rightarrow^*_{\mathcal{A}} q_1$ *so we have the critical pair* $(cons(q_2,\ app(q_1,\ q_0)), q_1)$;
2. $\mathcal{A}_1 = \mathcal{A} \cup Norm_{\gamma_f}(cons(q_2,\ app(q_1,\ q_0)) \rightarrow q_1)$ *is built:*

   (a) *we have* $\Psi = \begin{cases} \beta\ (rl, q_1, \{x = q_2, y = q_1, z = q_0\}) = q_3 \\ \vdots \\ \beta\ (rl, q_1, \{x = q_2, y = y, z = q_0\}) = q_4 \\ \vdots \\ \beta\ (rl, q_1, \{x = x, y = y, z = z\}) = q_5 \end{cases}$

   (b) $\gamma_f\ (rl, q_1, \{x = q_2, y = q_1, z = q_0\})$ *is computed:*

   i. *we have* $\{x = q_2, y = q_1, z = q_0\} \ll_{max} \{x = q_2, y = q_1, z = q_0\}$

   ii. *and* $\beta(rl, q_1, \{x = q_2, y = q_1, z = q_0\}) = q_3$

   iii. *so* $\gamma_f\ (rl, q_1, \{x = q_2, y = q_1, z = q_0\}) = \beta(rl, q_1, \{x = q_2, y = q_1, z = q_0\}) = q_3$.

   (c) $Norm_{\gamma_f}(cons(q_2,\ app(q_1,\ q_0)) \rightarrow q_1)$ *is computed with* $\gamma_f\ (rl, q_1, \{x = q_2, y = q_1, z = q_0\})$ *as follow:*

$$\begin{aligned} Norm_{\gamma_f}(cons(q_2, app(q_1, q_0)) \rightarrow q_1) &= \{cons(q_2, q_3) \rightarrow q_1\} \cup \\ & \quad Norm_{\gamma_f}(app(q_1, q_0) \rightarrow q_3) \\ &= \{cons(q_2, q_3) \rightarrow q_1,\ app(q_1, q_0) \rightarrow q_3\} \end{aligned}$$

(d) *the sets of* $\mathcal{A}$ *are updated to produce* $\mathcal{A}_1$*;*

The transitions $cons(q_2, q_3) \rightarrow q_1$ *and* $app(q_1, q_0) \rightarrow q_3$ *are added to* Δ*, the current automaton set of transitions and* q_3 *is added to the set of states.*

3. *as* $app(cons(q_2, q_3), q_0) \rightarrow^*_{\mathcal{A}_1} q_3$ *and* $cons(q_2, app(q_3, q_0)) \rightarrow^*_{\mathcal{A}_1} q_3$*, the critical pair* $(cons(q_2, app(q_3, q_0)), q_3)$ *is deduced;*

4. $\mathcal{A}_2 = \mathcal{A}_1 \cup Norm_{\gamma_f}(cons(q_2, app(q_3, q_0)) \rightarrow q_3)$*:*

   - *we have* $\Psi = \begin{cases} \beta(rl, q_3, \{x = q_2, y = q_1, z = q_0\}) = q_3 \\ \vdots \\ \beta(rl, q_3, \{x = q_2, y = y, z = q_0\}) = q_4 \\ \vdots \\ \beta(rl, q_3, \{x = x, y = y, z = z\}) = q_5 \end{cases}$
   - $\gamma_f(rl, q_3, \{x = q_2, y = q_3, z = q_0\})$ *is computed:*

     (a) *we have* $\beta(rl, q_1, \{x = q_2, y = y, z = q_0\}) = q_4$

     (b) *and* $\{x = q_2, y = y, z = q_0\} \ll_{max} \{x = q_2, y = q_3, z = q_0\}$

     (c) *so* $\gamma_f(rl, q_3, \{x = q_2, y = q_3, z = q_0\}) = \beta(rl, q_3, \{x = q_2, y = y, z = q_0\}) = q_4$
   - $Norm_{\gamma_f}(cons(q_2, app(q_3, q_0)) \rightarrow q_3)$ *is computed with* $\gamma_f(rl, q_3, \{x = q_2, y = q_3, z = q_0\})$ *as follow:*

     $$\begin{aligned} Norm_{\gamma_f}(cons(q_2, app(q_3, q_0)) \rightarrow q_3) &= \{cons(q_2, q_4) \rightarrow q_3\} \cup \\ & \quad Norm_{\gamma_f}(app(q_3, q_0) \rightarrow q_4) \\ &= \{cons(q_2, q_4) \rightarrow q_3,\ app(q_3, q_0) \rightarrow q_4\}; \end{aligned}$$
   - *the sets of* $\mathcal{A}_1$ *are updated to produce* $\mathcal{A}_2$*;*

     The transitions $cons(q_2, q_4) \rightarrow q_3$ *and* $app(q_3, q_0) \rightarrow q_4$ *are added to* Δ*, the current automaton set of transitions and* q_4 *is added to the set of states.*

5. *the critical pair* $(cons(q_2, app(q_4, q_0)), q_4)$ *is deduced from* $app(cons(q_2, q_4), q_0) \rightarrow^*_{\mathcal{A}} q_4$ *and* $cons(q_2, app(q_4, q_0)) \rightarrow^*_{\mathcal{A}} q_4$*;*

6. $\mathcal{A}_3 = \mathcal{A}_2 \cup Norm_{\gamma_f}(cons(q_2,\ app(q_4,\ q_0)) \rightarrow q_4)$:

- *we have* $\Psi = \begin{cases} \beta\,(rl, q_4, \{x = q_2, y = q_1, z = q_0\}) = q_3 \\ \vdots \\ \beta\,(rl, q_4, \{x = q_2, y = y, z = q_0\}) = q_4 \\ \vdots \\ \beta\,(rl, q_4, \{x = x, y = y, z = z\}) = q_5 \end{cases}$
- $\gamma_f\ (rl, q_4, \{x = q_2, y = q_4, z = q_0\})$ *is computed:*
  - *(a) we have* $\beta(rl, q_1, \{x = q_2, y = y, z = q_0\}) = q_4$
  - *(b) and* $\{x = q_2, y = y, z = q_0\} \ll_{max} \{x = q_2, y = q_3, z = q_0\}$
  - *(c)* $\gamma_f\ (rl, q_4, \{x = q_2, y = q_4, z = q_0\}) = \beta(rl, q_3, \{x = q_2, y = y, z = q_0\}) = q_4$
- $Norm_{\gamma_f}(cons(q_2,\ app(q_4,\ q_0)) \rightarrow q_4)$ *is computed with* $\gamma_f\ (rl, q_4, \{x = q_2, y = q_4, z = q_0\})$ *as follow :*

$$\begin{aligned} Norm_{\gamma_f}(cons(q_2, app(q_4, q_0)) \rightarrow q_4) &= \{cons(q_2, q_4) \rightarrow q_4\} \cup \\ & \quad Norm_{\gamma_f}(app(q_4, q_0) \rightarrow q_4) \\ &= \{cons(q_2, q_4) \rightarrow q_4,\ app(q_4, q_0) \rightarrow q_4\}; \end{aligned}$$

- *the sets of* $\mathcal{A}_2$ *are updated to produce* $\mathcal{A}_3$*;*

  The transitions $cons(q_2, q_4) \rightarrow q_4$ *and* $app(q_4, q_0) \rightarrow q_4$ *are added to* Δ*, the current automaton set of transitions.*

7. *No more critical pairs are found. The computation stops unlike the computations with the* γ *function of [Gen98b, GK00a] given in Definition 13.*

4.1.3 Why is it ok for protocols?

The syntax and semantics introduced in [GK00a] had been extended to deal with shared keys, session/complex keys, private keys, servers and hashed information (cf. Figure 4.2). In order to reduce the number of states used in the approximation function, the **null** term used by Genet and Klay to end a list of information had been removed. Thus a list ends with its last element, for example *con(a, cons(b, null))* becomes *cons(a, b)*.

agt(x)	x is an agent
c_init(x, y, z)	x thinks he has established a communication with y but he really communicates with z
c_resp(x, y, z)	x thinks he responds to a request for communication from y but he really communicates with z
cons(x, y)	concatenation of the information x and y
encr(x, y, z)	z is encrypted by y with the key x
goal(x, y)	x wants to communicate with y
hash1(x, y)	y is hashed by x
hash2(x, y, z)	z is hashed by y with the key x
key(x, y, z)	key created by z for y and x or key created by x to communicate with y with the information z
mesg(x, y, z)	z is a message sent by x to y
N(x, y, oc)	nonce created by x to communicate with y and where oc is a nonce number. When there is only one nonce between x and y, oc is equal to t0 (by default t0 for the nonce number 0). But when in a protocol run you have 2 different nonces created by x for y each nonce will have a different oc for example t0 and t1. To simplify the notations and as in the examples used only one nonce is created by an agent for another, **N(x, y)** is used except in Chapter 5 where a prototype is introduced.
pubkey(x)	public key of x
prikey(x)	private key of x
serv(x)	x is a server
sharekey(x, y, oc)	key shared by x and y with oc a key number. When in a protocol run you have 2 keys shared by x and y each key will have a different oc for example t0 and t1.

Table 4.2: Description of the terms used

The messages exchanged during the protocol runs are composed of basic pieces of information (i.e. agent name, shared key, etc.) or of concatenations of basic pieces of information (i.e. agent name and shared key encrypted, etc.). To reduce the number of messages that can be sent, the format of the messages is fixed by typing them. So in the term rewriting system (TRS) for example $pubkey(agt(x))$ is used to indicate that x can only be an agent name instead of having $pubkey(x)$. It makes analysis incapable of determining "type attacks". Nevertheless, [HLS03] justifies the assumption that all agents can identify the type of the information sent.

In a message, two types of information can be distinguished, one type that can be understood by the agent (i.e. agent names, etc.) and the other that cannot be understood by the agent (i.e. an agent cannot access a piece of information that has been encrypted if he does not have the right decryption key, etc.).

In the TRS, this distinction is visible. For example, an agent can identify a nonce if he has created the nonce. In the TRS when $agt(x)$ has created a nonce to communicate with $agt(y)$, N(agt(x),agt(y)) is found in the TRS and when it is a nonce created by someone else N(w,z) is used instead.

The new approximation also makes this distinction. In one case the state corresponding to the precise nonce is used and in the other, a state (because of the approximation β) that gathers together all the possible cases is used. The approximation β gives precise states for known information (as known information contains variables that can be substituted by terms of arity zero) and abstract states for unknown ones.

To avoid unknown information from different messages being gathered together by γ_f during the normalization process, each unknown piece of information in the TRS's rules has different variables. Thus it is impossible to find two rules within the TRS containing two unknown pieces of information with the same variables. For example, in the TRS it is impossible to have $N(x, y)$ in two rules; instead $N(x, y)$ and $N(x_1, y_1)$ are found. There is an exception; it is possible to find unknown pieces of information with the same variables when the protocol states these are identical in a message. For example, if the agent forwards an unknown nonce and writes it

twice in the message, then the term $N(x, y)$ is found twice in the rule.

The goal is to verify that information is kept secret during protocol runs (secrecy properties) and that actors can identify senders of messages (authentication properties). These properties are checked only for communications between trusted agents. The computation of the approximation automaton with γ [GK00a] guarantees that communications between trusted agents are not gathered together with other communications. The introduction of states for known and unknown information with γ_f does not modify this. The "known" states are linked to one agent, who created the information. The "unknown" states are linked to a particular message of a specific communication as γ_f creates them for one message involving particular agents. The approximation function and the normalization process that use these states distinguish communications between Alice and Bob from those between Bob and someone else, etc. Thus, the verification of the secrecy and authentication properties is not affected by the introduction of particular state for known and unknown information.

Moreover the distinction introduced between known and unknown information is very helpful, when the intersection of the approximation automaton and the negation property automaton is not empty (i.e. when the property being verified is not guaranteed to be valid, the property may be verified). By looking at the approximation automaton with the approximation function, information that can help the user to verify whether the property is satisfied with other methods [Mea96, Pau98, JRV00] or otherwise, can be deduced. In particular, by studying the states of the automaton, the user can find the particular step which may lead to an attack and thus have an idea as to how to direct the verification using other verification techniques.

4.2 Combining approach

When the intersection of the approximation automaton and the negation automaton is not empty[2], another verification technique must be used to check if the property

[2]From the experiments we carried out with our approximation only when the protocol was flawed it happened. But for some protocols, it could happen when the approximation is too abstract. For

is verified or not. The inductive approach of Paulson [Pau98] has succeeded in verifying a large range of protocols, but requires experts to carry out the proofs. The inductive technique, on the other hand, offers a very powerful framework to backup the approximation when it fails while the approximation approach simplifies the work of the experts. It seems a good idea to combine the approximation approach with the inductive technique as both approaches deal with traces of events, their intruder has the same properties, and the protocol steps can easily extract from the Isabelle specification to be transform into a term rewriting system.

In this section, first the inductive approach is detailed, followed by an introduction to the combination of the techniques.

4.2.1 Inductive approach

This technique was briefly introduced in Chapter 2. In his approach, Paulson reasons on the set of all the possible traces reachable with a particular protocol.

As initial assumptions, he has an infinite number of agents in the network and an intruder that matches Dolev and Yao's intruder model [DY83]. He also considers that traces are lists of events and an event contains information about agents and messages.

In his method, there are three types of agents:

- The server who can always be trusted.
- The user who can be safe or not (depending on whether the spy knows his secret key, for example).
- The spy who is an attacker and is accepted as a valid user.

These agents can extend the trace in any way permitted by the protocol and can forward messages that they cannot read. Messages may include:

agent names, nonces, timestamps, keys, compound messages, hashed messages, encrypted messages. There are two kinds of events:

example with the approximation of Genet and Klay, no distinction is made for two nonces sent by the same agent, so the no emptiness is observed even if one nonce is secured like for ISO611 protocol.

- *Says A B X*: the agent A sends the message X to the agent B.
- *Notes A X* : the agent A stores the message X internally.

To manipulate the event lists, he defines three operations: *parts*, *analz* and *synth*. These operations are needed to express assertions and describe the possible actions of the attacker. They are defined by induction on possibly infinite sets of messages.

If the set H contains an agent's initial knowledge and the history of all messages sent in a trace, then we have:

- *parts H*, the components of messages in H that could be obtained by decomposing complex messages and breaking every encryption.
- *analz H*, the components of messages in H that could be decrypted using only the information contained in H.
- *synth H*, the set of all messages that could be built up using messages in H as components.

So if the set H shows all the traffic in the network, then the attacker can send fraudulent messages drawn from the set *synth*(*analz H*).

The evolution of a trace is defined by 4 types of messages that model precise events:

1. initial trace that is an empty list (Nil on Figure 4.3);
2. protocol step that adds events to the current trace if some pre-conditions are satisfied. NS1 in Figure 4.3 models the first step of the Needham-Schroeder-Lowe protocol [Low95] (cf. Figure 3.6);
3. fake message that defines how the message is created (pre-condition) and sent by the intruder (Fake on Figure 4.3);
4. accidental message that models the accidental loss of information by an agent (Oops on Figure 4.3 models the loss of the session key in the symmetric key Needham-Schroeder protocol [NS78]).

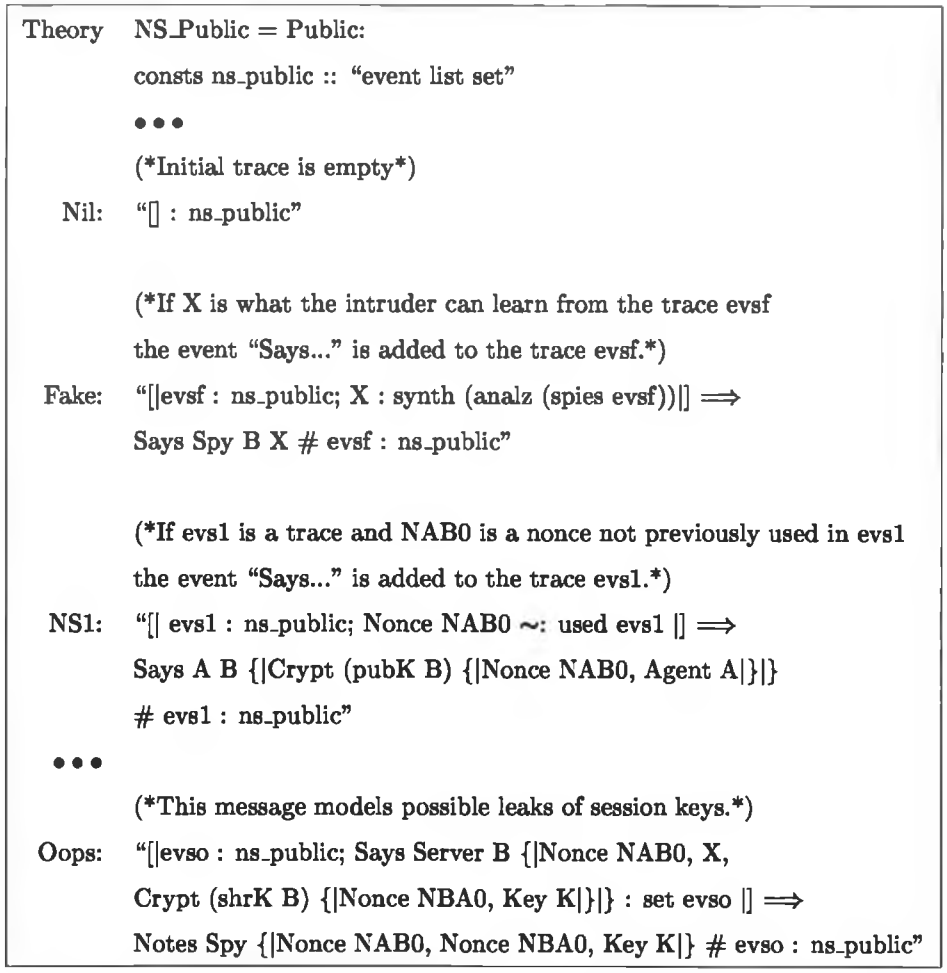


Figure 4.3: Example of Isabelle specification

Several protocol properties can be verified by induction on the trace. This means that the method checks that each type of message introduced above preserves the properties. If a rule does not satisfy the property, then a flaw has been discovered. The properties that can be verified with this technique are:

- possibility properties, which assure that message formats agree from one step to the next.
- forwarding lemmas, which assure that the spy will not learn anything new by seeing a message that an agent forwards and can not decrypt.
- regularity lemmas, which assure that whatever the spy does he can never

get hold of particular information (excluding information known by the bad agents). For example the intruder never acquires of a secret key initially shared by two trusted agents.

- unicity theorems, which assure that session keys and timestamps uniquely identify their message.
- secrecy theorems, which assure that secret information can not be caught by the intruder.
- authenticity guarantees, which assure the authenticity of information.

Using the Isabelle theorem prover [Pau94], Paulson verified a large range of protocols: the Internet protocol TLS [Gro96b, Pau99], the Kerberos protocol [BP98a, BP97, BP98b], the SET protocol [BMPT00, Pau01, BMP02, BMP03] and some other protocols[Pau98]. The proofs of these protocols are available on the Isabelle website[3].

The inductive approach has successfully verified many protocols, so what is the advantage of combining it with the approximation technique and how are these techniques combined?

4.2.2 Why and How?

By combining the inductive approach [Pau98] and the approximation technique, the advantages of each method are exploited. Both approaches reason on traces and are easy to understand.

The inductive approach is a good method of verification which can verify several properties. But in this approach, the secrecy and authenticity properties/theorems are very difficult to prove (to have an idea, look at the proof of the Needham-Schroeder protocol on the Isabelle website). The proofs require an experienced user to introduce the right lemma at the right time to make the proofs stop.

On the other hand, with the approximation technique, a quick and semi-automatic (the user only enters the TRS, the approximation function, the initial automaton)

[3]http://www.cl.cam.ac.uk/Research/HVG/Isabelle/library/HOL/Auth/

verification of these two properties can be done. The user does not need to be an expert to write the TRS, the initial automaton and the approximation function. Chapter 5 will explain how the TRS, the approximation function, the initial automaton can be automatically generated from an Isabelle specification. The automaton for the negation of the secrecy is automatically generated and the automaton for the negation of the authentication based on "c_resp" and "c_init" can be re-used for different protocols.

In both techniques, the secrecy of the information is proven for honest agents and assumes that no accident reveals the information to the intruder. In the inductive approach, the authenticity is proven for honest agents. This is established if honest agents received their last message with the information they expected, given that they sent the correct messages previously. In the approximation approach, authenticity is proven for honest agents. This is established if honest agents received their last message with the information they expected, if the message was created by the right agent, and there was no accidental loss. The approximation computes an over-approximation of the set of all the messages that can be sent, thus the second condition to establish the authenticity in the inductive approach is guaranteed.

Hence, by using the approximation in the inductive proof, the user's work can be simplified and the time spent in the verification of protocols can be reduced. When the properties cannot be verified by the approximation, the results of the unsuccessful verification will give information that will help the user to carry the inductive approach.

The techniques will be combined as follow:

1. The approximation technique is used to verify the secrecy and authentication properties.

2. If the properties are satisfied, these results are used as axioms in the inductive method. If the properties are not satisfied, the inductive approach is used to check if a flaw really exists.

   By looking at the approximation function and at the approximation automa-

ton, the user can find the protocol step which might lead to a flaw and use that information to do his inductive proof. Each state used in the approximation function is linked to a specific term (known information like $agt(q_2)$) or a specific rewrite rule (unknown information like $encr(a\_1, b\_1, c\_1)$). Thus by looking at states that are used in the transition $add(...) \rightarrow q_{13}$ in the approximation automaton, it is possible to trace back messages that might lead to a flaw in the protocol.

3. The inductive approach is used to prove the remaining properties (the proofs of those properties are mainly the same for all the protocols [Pau98]).

Chapter 5.3 will illustrate using real examples how the idea works. The approximation technique can simplify the work of an Isabelle user.

4.3 Conclusion

In this chapter, we introduced a substantial and significant improvement to Genet and Klay's approach by:

1. defining an automatically generated approximation function that ensures that:
   - the computation of the automaton terminates,
   - and the secrecy and authentication can be verified on the resulting automaton;
2. combining this improved approach with the inductive approach of Paulson to take advantage of the strengths of both techniques.

The next chapter will introduce the tool developed to test these improvements and will detail the results of experiments carried out.

Chapter 5

Prototype

In [GK00a], a prototype based on a tree automata library[1] developed by Thomas Genet [Gen98b] for the ELAN[2] prototyping environment had been used to carry out experiments on the Needham-Schroeder-Lowe protocol. The prototype was not optimized for the completion algorithm (cf. Algorithm 2). Thomas Genet and Valérie Viet Triem Tong [GT01] implemented Timbuk[3]. Timbuk is a library for OCAML[4] [RV98, LDG+01] that is optimized for the computation of reachable configurations of a system using the completion algorithm.

To be able to validate the improvements described in Chapter 4, in particular the approximation function, a prototype was implemented around Timbuk. The prototype extracts the protocol steps from an Isabelle specification and generates the input file for Timbuk (alphabet + variables + TRS + initial automaton + approximation function). The prototype also generates the automaton to verify secrecy properties. Before the prototype is presented, the Timbuk library must be introduced.

[1] http://www.loria.fr/equipes/protheo/SOFTWARES/ELAN/examples/elan-automata.html
[2] http://www.loria.fr/equipes/protheo/SOFTWARES/ELAN/
[3] http://www.irisa.fr/lande/genet/timbuk/index.html
[4] http://caml.inria.fr/ocaml/index.html

5.1 Timbuk library

This library offers basic functions on non-deterministic finite tree automata, such as:

- boolean operations: intersection, union, inversion;
- normalization of transitions;
- reading and writing automata to disk.

This tool was chosen as it offers the possibility to compute the approximation of a set of descendants of an initial automaton from a TRS with an approximation function according to the completion algorithm (cf. Algorithm 2).

The format of the input file to compute this approximation is given by Figure 5.1.

Ops ... *alphabet used*
Vars ... *list of variables used*

TRS R
... *list of rules: first term* $\longrightarrow$ *second term*

Automaton automat
States ... *list of states*
Final States ... *list of states*
Transitions
... *list of transitions: first term* $\rightarrow$ *second term*

Approximation R1
States ... *list of states*
Rules
... *list of normalization rules: [term to normalize]* $\longrightarrow$ *[normalization process]*

Figure 5.1: Timbuk input file

The approximation *R1* in 5.1 is optional as Timbuk offers 4 modes:

- automatic: no approximation *R1* has been given and the algorithm uses the approximation function γ (Definition 13). It means that new states are created for each step of the completion and the computation may not terminate.

- step-by-step: no approximation *R1* has been given and the user normalizes the terms at each step of the completion.

- semi-automatic: an approximation *R1* is given by the user. In that mode the user can either enter approximation rules where all the variables are substituted or where some variables are kept.

  For example, the user can enter $[N(q_2, q_3)] \longrightarrow [N(q_2, q_3) \rightarrow q_5]$ and $[N(q_2, q_4)] \longrightarrow [N(q_2, q_4) \rightarrow q_5]$ or he can enter $[N(x, y)] \longrightarrow [N(x, y) \rightarrow q_5]$. During the computation Timbuk will substitute the variables, and $N(q_2, q_3)$ and $N(q_2, q_4)$ will be linked to q_5. Thus at the end, in the approximation automaton in both cases, we have $N(q_2, q_3) \rightarrow q_5$ and $N(q_2, q_4) \rightarrow q_5$.

- combination of the previous modes.

It was decided to use Timbuk in the semi-automatic mode by developing a tool that automatically generates the input file for Timbuk. The user would only have to give the protocol specification and then would get the input file for Timbuk.

5.2 IS2TiF (Isabelle Specification to Timbuk File)

This section explains how our prototype IS2TiF (Isabelle Specification to Timbuk File) works and also how to use it. The tool simplifies the user's work by generating the correct input file for Timbuk from a protocol specification and also the negation automata for the secrecy properties.

As the previous section shows, the Timbuk file can roughly be split into 3 parts: the TRS, the initial automaton and the approximation function. The prototype translates the protocol specification into rewrite rules using compiler translation techniques. The TRS is then used to generate the approximation function. Since the initial automaton is the same for all protocols, it is saved into a file.

The negation automaton is produced using compiler translation techniques from a specification of the information that must be secret.

5.2.1 TRS + Initial automaton + Approximation function

The user is presented with two options, depending on whether he wishes to use the combining approach (cf. Section 4.2) or just the approximation technique. The protocol specification can be saved in the file *inputIT.txt*[5] or an ISABELLE specification can be written and the prototype used to extract the protocol's steps into *inputIT.txt*

In *inputIT.txt*, messages follow the ISABELLE format **Says A B M** where A and B are agent names and M is the message sent by A to B. Messages can contain (cf. Table 5.1):

- agent names;
- nonces;
- keys (public keys, shared keys, complex keys);
- session identifier;
- pre-master secret;
- hashed messages (using a key or not);
- encrypted messages.

In Table 5.1, one notices that the nonces and shared keys end with a number, for example *Nonce NAB0*. This is because agents can share more than one key and nonce in a protocol run.

This is well illustrated in the following example: in a protocol an agent can create two nonces, one known by everybody and one that should be secret. In [GK00a], those nonces would have been modeled by the same term, for example *N(agt(a),agt(b))* and the verification of the secrecy of the second nonce would have

[5]The file where the prototype is going to look for the protocol's steps

Agent A	name of agent A
Nonce NAB0	nonce number 0 created by A to communicate with B If in the same protocol you have two nonces between A and B, you have NAB0 and NAB1
PubK A	public key of A
PriK A	private key of A
ShrK AB0	key number 0 shared by A and B If in the same protocol you have two keys between A and B, you have AB0 and AB1
Key (I_1) (I_2) I_3	complex key built with information I_1, I_2 and I_3
Sid AB	session identifier created by A to communicate with B
PMS AB	pre-master secret created by A to communicate with B
Hash M	M is hashed information
Hash (K) M	M is hashed information with the key K
Crypt (K) M	M is encrypted with the key K
{|I_1,I_2|}	concatenation of the information I_1 and I_2

Table 5.1: Syntax and semantics used in *inputIT.txt*

failed. By adding a number at the end of nonces, those nonces can now been distinguished by having for example *N(agt(a),agt(b),0)* and *N(agt(a),agt(b),1).*

When the file *inputIT.txt* is available, the program does a lexical and syntax analysis of this file. It builds an abstract syntax tree of the protocol steps. The lexical analysis reduces the text of each line to independent lexical units. Then the syntax analysis uses those units to build the abstract syntax tree.

To illustrate this process, in Table 5.2, we have the information {|*Agent A, Nonce NAB0*|}. We first apply a lexical analysis to this information and get the following sequence of lexical units: *Lsymbol* "{|", *LAgent*, *LString* "*A*", *Lsymbol* ",", *LNonce*, *LString* "*NAB0*" and *Lsymbol* "|}". Then by applying a syntax analysis, we get an abstract syntax tree of the initial information.

Isabelle line	{|Agent A, Nonce NAB0|}
⇓ Syntax analysis	⇓ Lexical analysis
Sequence of lexical units	Lsymbol "{|", LAgent, LString "A", Lsymbol ",", LNonce, LString "NAB0", Lsymbol "|}"
⇓ Syntax analysis	⇓ Syntax analysis
Syntax tree	*Tlinf* Agent "A" Nonce "NAB0"

Table 5.2: Lexical and syntax analysis example

5.2.1.1 TRS

Using the syntax tree in memory, our prototype generates the TRS.

A message sent will be used as a pre-condition to send another one. Each protocol step is used to generate the first term, *pre-condition*, of one rewrite rule and the second term, *response*, of another one.

For each of the protocol steps, the prototype creates and saves all the first terms in the file *TRS1.txt* and all the second terms in the file *TRS2.txt*. To generate the terms, the following rules are applied to each step:

- for the first term of a rewrite rule:

  1. only information known by the receiver appears clearly.

     For example, if an agent receives a nonce that he had created, we have N(agt(a),agt(b),t0). Otherwise we have $N(a_1, b_1, t_1)$.

  2. if the receiver has the right decryption key then he has access to the encrypted information.

     So for example, if an agent receives information encrypted with his public key then he has access to that information and we have encr(pubkey(agt(b)),a_1,...). Otherwise we have $encr(a_1, b_2, c_3)$.

- for the second term of a rewrite rule:

  1. only information known by the sender appears clearly.
  2. sender can encrypt information if he has the right encryption key.

In order to reduce the computation time of the approximation automaton, the messages are typed. That means that agents know the format of messages and will only reply to messages following the right format.

Table 5.3 shows the TRS rules generated for the first step of the Needham-Schroeder-Lowe protocol.

Isabelle specification	Says A B {|Crypt (pubK B) {|Nonce NAB0, Agent A|}|}
TRS	$\cdots \longrightarrow$ U(LHS, mesg(agt(a), agt(b), encr(pubkey(agt(b)), agt(a), cons(N(agt(a), agt(b), t0), agt(a))))) mesg(a_4, agt(b), encr(pubkey(agt(b)), a_3, cons(N(a_1, b_1, t_1), agt(a)))) $\longrightarrow$ U(**LHS**, $\cdots$)

Table 5.3: Example of transformation

In the row "TRS", the first rule is easy to understand, the agent knows all the information, which is why everything appears clearly.

The second rewrite rule is a bit more complex, here the message is the first term of the rewrite rule. Therefore when the agent receives the message, he does not know who sent and encrypted it, which is why we have a_4 and a_3 (and not agt(a_4) and agt(a_3) as it could be from a server). However, the agent has access to the information encrypted as the message has been encrypted with his public key, pubkey(agt(b)), and he knows his private key, needed to decrypt the message. As messages are typed, he knows that the first information is a nonce, but as he did not create it, we have N(a_1, b_1, t_1). He is also expecting an agent's name, which explains the agt(a) subterm.

The tool does not generate the rewrite rules for the authentication (rules with c_init and c_resp). As the authentication might require a more complex condition than just the reception of the last message sent, the user has to enter these rules by hand if he wants to verify such properties. These rules will be saved in *Sinit.txt* and *Sresp.txt*. The program then generates *TRS.txt* by linking all the first terms

of *TRS1.txt* to their match in *TRS2.txt* and then by adding the rules from *Sinit.txt* and *Sresp.txt*.

5.2.1.2 Initial automaton

The intruder's abilities are the same for all protocols (Dolev-Yao model [DY83]). The initial configuration of the network is also the same for all protocols: everybody wants to communicate with everyone. That is why the initial automaton can be saved in a file, which will be re-used for every verification.

The initial automaton defines the number of participants and the communications that will be established. [CLC03] proved that two agents are sufficient for the analysis of security properties of cryptographic protocols when the protocols allow an agent to talk to himself ("self talking" protocol). If the protocol does not allow "agents to talk to themselves" ("not self talking" protocol) and there is an attack involving n agents, then there is an attack involving at most $k + 1$ agents (k is the number of roles that an agent can play). So in order to use these results to optimize the run-time of the computation of the approximation automaton, four initial automata are identified:

- if the protocol allows an agent to talk to himself:
  - one automaton with an honest agent A and a set of untrusted agents and where the communications between A and A, A and the set, the set and the set are going to be established;
  - one automaton with a server S, an honest agent A and a set of untrusted agents and where the communications between A and A, A and the set, the set and the set are going to be established. This is when a trusted server is used by the agents to establish their communications.
- if the protocol does not allow an agent to talk to himself:
  - one automaton with two honest agents, A and B, and a set of untrusted agents and where the communications between A and B, A and the set, B and the set, the set and the set are going to be established;

– one automaton with a server S, two honest agents, A and B, and a set of untrusted agents and where the communications between A and B, A and the set, B and the set, the set and the set are going to be established.

The rewrite rules criticize the intruder's capacities, the AC rewrite rules and the initial automaton for protocols without a server are saved in the file *automatonwos1.txt* if protocols allow "self talking" and in the file *automatonwos2.txt* if protocols do not allow "self talking". Those files do not include rewrite rules and transitions involving a server. The rewrite rules and initial automaton for protocols with a server are in the files *automatonws1.txt* for "self talking" protocols (same file as *automatonwos1.txt* + rewrite rules and transitions involving a server) and files *automatonws2.txt* for "none self talking" protocols. The user is also free to add rewrite rules or transitions into those files if he wants to change the initial assumptions (for example if he decides that initially the intruder knows a shared secret between trusted agents).

5.2.1.3 Approximation function

In Timbuk, an approximation rule is composed of two terms. The first term corresponds to the term that will be normalized and the second to the normalization process that will be used during the computation of the approximation automaton by Timbuk.

The process to generate the approximation function is identical to the process for generating the TRS:

1. generation of the first terms, file *approx1.txt* (terms to normalize),
2. generation of the second terms, file *approx2.txt* (normalization processes),
3. creation of a file *approx.txt* (cf. Figure 5.2) with *approx1.txt*, *approx2.txt* and *finapprox.txt* that contains approximation rules for c_init rules, c_resp rules and AC rules.

The approximation rules are generated from the second terms of rewrite rules of the protocol steps. For each rule, the sender and receiver variables are replaced by

the states linked to the agents. There are nine substitutions corresponding to the nine possible exchanges between Alice, Bob and the rest (cf. Section 3.2.2). The reader may wonder why there are nine cases whereas in the initial automaton, the nine possible exchanges are not considered anymore. This is to make sure that the user can alter the initial automaton without caring about the approximation, since the approximation is generated for the nine possible exchanges.

Figure 5.2 gives an idea of the format of the approximation produced by the tool. On this figure, two approximation rules, one when an agent creates and sends a nonce (first rule) and one when an agent forwards a nonce (second rule), are given.

```
[U(LHS, mesg(agt(q1), agt(q2), encr(pubkey(agt(q2)), agt(q1), cons(N(agt(q1), agt(q2), qt0),
agt(q1))))) → q13] ⟶ [LHS → q13 agt(q1) → q4 agt(q2) → q5 N(q4, q5, qt0) → q16
cons(q16, q4) → q15 pubkey(q5) → q14 encr(q14, q4, q15) → q13
mesg(q4, q5, q13) → q13]

[U(LHS, mesg(agt(q0), agt(q0), encr(pubkey(agt(q0)), agt(q0), cons(N(x, y, t), agt(q0)))))
→ q13] ⟶ [LHS → q13 agt(q0) → q3 N(x, y, t) → q70 cons(q70, q3) → q69
pubkey(q3) → q66 encr(q66, q3, q69) → q13 mesg(q3, q3, q13) → q13]
```

Figure 5.2: Example of approximations

We said that the computation of the set of β functions is exponential (cf. previous chapter) but to deal with protocols there is no need to compute the whole set. Only the β with substitutions in which only the variables of known terms are substituted by states are interesting. This is because only known terms contain variables that are substituted by states linked to terms of arity zero during the completion. Only agents' names, indices of nonces, indices of keys and freshness levels have an arity zero. Thus the approximation rules produced only care about terms in which agents' names, indices of nonces, indices of keys and freshness levels are states and not variables like on Figure 5.2. The normalization process of a rule is then:

- defined using a β function when the term to normalize still contains some free variables (i.e. x, y and t in the second rule in Figure 5.2). During the

completion by Timbuk, the free variables are replaced only by relevant states (those really required by the computation); the result of this substitution will give the normalization process using γ_f because of the relation between β and γ_f (cf. Definition 25).

- defined using a γ_f function when the term to normalize has all his variable substituted (i.e. the first rule in Figure 5.2).

Moreover, in the set of approximation rules, rules are added for particular approximations covered by the two simplification rules of Definition 25. For example if a message contains at least two terms of identical information (for example one nonce known and one unknown or two unknown nonces), approximation rules are added. Those extra rules cover *equality cases*:

- the message contains one known nonce, N(agt(q2), agt(q2), qt0) and one unknown nonce N(a_1, b_1, t_1), 2 rules must be found in the approximation function.

  One with N(agt(q2), agt(q2), qt0) and N(q5, q5, qt0) (assuming that agt(q2) is linked to the state q5) for the case where after substitution and normalization, the two nonces are identical. Another with N(agt(q2), agt(q2), qt0) and N(a_1, b_1, t_1) to cover the other cases.

- the message contains two unknown nonces, N(a_1, b_1, t_1) and N(a_2, b_2, t_2), 2 rules must be found

  in the approximation function.

  One with only N(a_1, b_1, t_1) or only N(a_2, b_2, t_2) for the case where the two nonces are identical. And one with N(a_1, b_1, t_1) and N(a_2, b_2, t_2) to cover the other cases.

  The rule with only N(a_1, b_1, t_1) or N(a_2, b_2, t_2) has to be added to cover the case in which after substitution, the two nonces are equal (second simplification rule of Definition 25). Between N(a_1, b_1, t_1) and N(a_2, b_2, t_2), the nonce (in general the information) that is the more accessible to the intruder will be picked. The following rules apply in that order:

1. if one of the nonces, $N(a_2, b_2, t_2)$, is not encapsulated in a cryptographic primitive then $N(a_2, b_2, t_2)$ is picked.

   The intruder already has access to $N(a_2, b_2, t_2)$, so the state linked to this nonce is already (or will be) in his possession. Using $N(a_1, b_1, t_1)$ would give him access to $N(a_1, b_1, t_1)$ but also to all the nonces linked to the state q, which we do not want to do as this may give him access to other nonces that might perfectly be inaccessible for the intruder.

2. if one of the nonces, $N(a_2, b_2, t_2)$, is encapsulated in a cryptographic primitive but the intruder has access to it (for example if $N(a_2, b_2, t_2)$ is encrypted with the key of an untrusted agent) then $N(a_2, b_2, t_2)$ is picked.

3. if both nonces are encrypted but one of the them, say $N(a_2, b_2, t_2)$, has been encrypted more than the other one, then $N(a_2, b_2, t_2)$ is picked.

4. if both nonces have the same "encapsulation level" then $N(a_1, b_1, t_1)$ is picked.

The goal of these rules is to ensure that the approximation reveals as little information as possible to the intruder.

The computation of the rules only takes a couple of seconds as in the protocol context :

- all the terms to normalize will point only to the terminal state q_{13} ;
- only the term with valid substitutions of variables by states linked to terms of arity zero are considered (terms with only the subterms of the form agt(x) where x is substituted).

The normalization for c_init, c_resp and AC terms is always the same so the approximation rules for those terms are saved in *finapprox.txt*. When *approx1.txt* and *approx2.txt* are available, the program generates *approx.txt* that contains the approximation function (protocol steps -approx1 → approx2- + *finapprox.txt*).

By concatenating *TRS.txt*, *automatonwos1.txt* or *automatonwos2.txt* or *automatonws1.txt* or *automatonws2.txt*, and *approx.txt*, the prototype creates the input file for Timbuk. In Appendix D, the Timbuk file used to verify the Needham-Schroeder protocol where an agent cannot talk to himself, is presented.

5.2.2 Negation automaton

The user enters the information that he wants to check in a file, for example *secrecy.txt*. In that file, the information must have the form **Inf A B M** where A is the sender and B the receiver of the message where the information M first appears. M must follow the syntax presented in Table 5.1.

The program does a lexical and syntax analysis of the file, then from the syntax tree it generates the automaton criticize that the information M known by the honest agents (Alice and Bob) is also known by the intruder. If the user wants to check if a nonce between Alice and Bob is secret, then he has to enter *Inf A B* $\{|NAB0|\}$ and the prototype will produce the automaton in Figure 5.3.

Automaton	Not_Secret	
States	q1 q2 q4 q5 q13 qt0	
Final States	q13	
Transitions		
	A → q1	agt(q1) → q4
	B → q2	agt(q2) → q5
	U(q13, q13) → q13	t0 → qt0
	N(q4, q5) → q13	N(q5, q4, qt0) → q13
	N(q4, q4, qt0) → q13	N(q5, q5, qt0) → q13

Figure 5.3: Nonces between Alice and Bob

5.2.3 User guidelines to use the IS2TiF

In Ocaml, the user has to load the file *is2tif.ml* and then enter **go();;** to launch IS2Tif. The user can enter his commands (cf. Table 5.4) after the prompt. When the user has generated his Timbuk input file and all the negation automata for the

secrecy properties[6], he can leave IS2TiF. He can then use his files in Timbuk to compute the approximation automaton of his protocol and check the properties.

For the user who is not familiar with Timbuk, we have a file *approx.ml* that the user can load in Ocaml and enter:

let aut_comp=MyComp.complet r protot r1[];;

This file can be used to compute the approximation if the user has saved his input file under *approx.txt*. When the user has done that, he has access to a menu and can launch the computation of the approximation.

Figure 5.4 summarizes how a user verifies protocols with IS2TiF and Timbuk. He has to give the protocol and the properties to IS2TiF (except for the authentication, the negation automaton is the same for all the protocol so this automaton is provided to the user). Then he uses the files produced by IS2TiF to launch the computation of the approximation before he can check the automata intersection with Timbuk.

5.3 Experiments

To validate the improvements and the prototype some tests were conducted on simple protocols taken from [CJ97] and on the Transport Layer Security protocol [Gro96b]. The tests have been carried out on a Pentium III (733 MHz) with 128Mb of RAM + 500Mb of virtual memory managed by Windows NT.

The objectives of those experiments are multiple. The theory behind the approximation approach is watertight if used under the right conditions (cf. Proposition 9). The first objective of those experiments is to test that our prototype is producing the correct input file for Timbuk and to check that the produced approximations are correct (i.e. no bug in Timbuk). The second objective is to check that the approach can be used on concrete protocol verification and more importantly that the verification of those protocols does not miss any of the known flaws. The third

[6]the negation automaton for the authentication property will be the same for all the protocols so it is saved in file *authen.txt*; this automaton was presented in Chapter 3 Figure 3.13

AddSinit r	replaces the current TRS rule in *Sinit.txt* by r (ie. mesg(agt(a_6), agt(a), encr (pubkey(agt(a)), agt(a_5), cons(N(agt(a), agt(b)), a_2))) → U(LHS, c_init(agt(a), agt(b),agt(a_5)))); if the user does not want any rule it just does **AddSinit**
AddSrep r	replaces the current TRS rule in *Sresp.txt* by r; if the user does not want any rule it just does **AddSrep**
End	closes IS2TiF
Extract filename	extracts the protocol steps from the Isabelle file filename and saves them in *inputIT.txt* (ie. Extract ns.txt)
Generate1 filename	generates the input file, filename, for Timbuk for protocols allowing an agent to talk to himself (ie. Generate1 toto.txt)
Generate2 filename	generates the input file, filename, for Timbuk for protocols allowing an agent to talk to himself (ie. Generate2 toto.txt)
Generate_automaton filename	generates the file, filename, which contains the automaton to check a secrecy property (ie. Generate_automaton Nonce.txt)
Load filename	loads the file in memory (ie. Load input.txt)
List	lists the TRS rules

Table 5.4: IS2TiF commands

objective is to test the effectiveness of the implemented solution for the combination approach. The prototype was not define with the help of a model (Z,B, UML) and the test campaign was not built around the protocol functionalities but around a set of protocols. Thus the prototype is reliable only for the protocols we checked as we manually checked the results. The prototype should have been unit tested to gain in reliability.

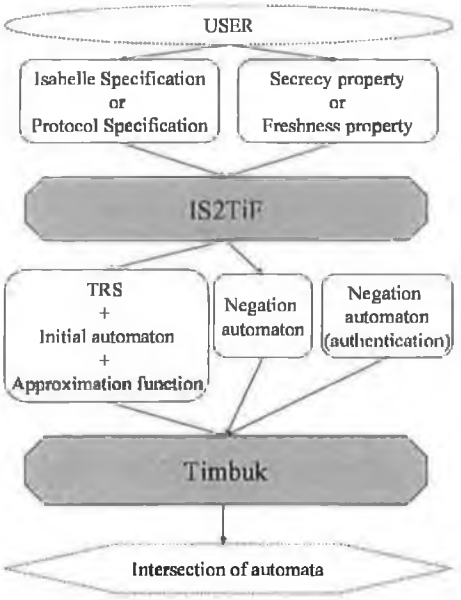


Figure 5.4: IS2TiF + Timbuk

5.3.1 Standard protocols

The protocols studied here are well known, as are their flaws, and are usually used to validate new verification techniques:

- Needham-Schroeder-Lowe [Low95];
- Needham-Schroeder [NS78];
- Otway-Rees simplified [AN96];
- Otway-Rees modified by us;
- Woo-Lam Pi3 [WL94];
- Andrew Secure RPC [Sat89].

For each protocol, the secrecy and authentication properties were checked. Table 5.5 summarizes the results[7] obtained with our approximation. Where it indicates "may

[7]the first time is the time to compute the approximation automaton when the protocol allows an agent to talk to himself. The second time is the time to compute the approximation automaton

be a flaw", then the flaw was found using the inductive approach. Table 5.5 also illustrates the main drawback of the approach, the computation time is exponential; just by adding one agent the execution times increase drastically. But the expensive time approach can cover an extra constraint that stands that an agent cannot speak to himself. Thus the few exra minutes are not important if that constraint is strong for the protocol verified.

Protocols (computational time)	**Properties**	
	Secrecy	**Authentication**
Needham-Schroeder symmetric key (41s and 6min 10s)	verified	verified
Needham-Schroeder (18s and 3min 50s)	may be a flaw	may be a flaw
Needham-Schroeder-Lowe (16s and 1min 59s)	verified	verified
Otway Rees simplified (1min 03s and 17min 12s)	verified	verified
Otway Rees (ours) (1min 22s and 20min)	verified	may be a flaw
Woo Lam Pi3 (31s and 14min)	none	verified
Andrew Secure RPC (48s and 10min 50s)	verified	verified

Table 5.5: Test results

No unknown flaws were discovered. The work (with the new approximation function and the combining approach) done on each protocol is the same. The verification of the Needham-Schroeder-Lowe is detailed in the following section. Then the properties for each protocol are described.

5.3.1.1 New function + combining approach

The Needham-Schroeder-Lowe protocol was introduced in Section 3.2.2. Alice and Bob want to establish a secure communication using a public key infrastructure. Before they send any vital information, they use the Needham-Schroeder-Lowe protocol (cf. Figure 5.5) to exchange nonces that later should permit the identification of senders of messages.

To verify the protocol, the first step is to write the Isabelle specification. Figure 5.6 shows the protocol specification using Isabelle syntax. Then the prototype is

when the protocol does not allow an agent to talk to himself

	Alice initiates a protocol run, sending a nonce Na and her name A to Bob.
Message 1:	$A \Longrightarrow B : \{Na, A\}_{Kb}$
	Bob responds to Alice's message with a further nonce Nb.
Message 2:	$B \Longrightarrow A : \{Na, Nb, B\}_{Ka}$
	Alice proves her existence by sending Nb back to Bob.
Message 3:	$A \Longrightarrow B : \{Nb\}_{Kb}$

Figure 5.5: Needham-Schroeder-Lowe protocol

used to generate the Timbuk input file. When this file is created, the Timbuk library is used to compute the approximation automaton.

When the computation of the approximation automaton is over, the secrecy and the authentication properties can be verified.

The secrecy property The secrecy property that must be guaranteed by the protocol is: *"The intruder can never access the nonces created by Alice (to communicate with Bob) or Bob (to communicate with Alice)".*

To verify this property with the approximation technique, first an automaton of the negation of this property is automatically generated with IS2TiF. This automaton is identical to the one in [GK00a] and given in Figure 3.12 in Chapter 3.2.2.

To refresh the reader's memory the automaton is given in Figure 5.7. "Not_Secret" recognizes the nonces created by Alice and Bob to communicate with each other. The intersection of this automaton with the approximation one is empty, so the protocol satisfies the property.

The authentication property The authentication property is: *"If Alice thinks that she communicates with Bob, then she really speaks with Bob. And if Bob thinks that he communicates with Alice then he really talks with Alice".*

This automaton is identical to the one in [GK00a] and given in Figure 3.13 in

```
Theory  NS_Public = Public:
        • • •
        (*Alice initiates a protocol run, sending a nonce to Bob*)
NS1:    "[| evs1 : ns_public; Nonce NAB0 ~: used evs1 |] ⟹
        Says A B {|Crypt (pubK B) {|Nonce NAB0, Agent A|}|}
        # evs1 : ns_public"

        (*Bob responds to Alice's message with a further nonce*)
NS2:    "[| evs2 : ns_public; Nonce NBA0 ~: used evs2;
        Says A' B {|Crypt (pubK B) {|Nonce NAB0, Agent A|}|} : set evs2 |] ⟹
        Says B A {|Crypt (pubK A) {|Nonce NAB0, Nonce NBA0, Agent B|}|}
        # evs2 : ns_public"

        (*Alice proves her existence by sending NBA0 back to Bob.*)
NS3:    "[| evs3 : ns_public;
        Says A B {|Crypt (pubK B) {|Nonce NAB0, Agent A|}|} : set evs3;
        Says B' A {|Crypt (pubK A) {|Nonce NAB0, Nonce NBA0, Agent B|}|}:
        set evs3|] ⟹ Says A B {|Crypt (pubK B) {|Nonce NBA0|}|}
        # evs3 : ns_public"
 end
```

Figure 5.6: Inductive specification of the Needham-Schroeder-Lowe protocol

Automaton	Not_Secret	
States	q_1 q_2 q_4 q_5 q_{13}	
Final States	q_{13}	
Transitions		
	$A \rightarrow q_1$	$agt(q_1) \rightarrow q_4$
	$B \rightarrow q_2$	$agt(q_2) \rightarrow q_5$
	$U(q_{13}, q_{13}) \rightarrow q_{13}$	
	$N(q_4, q_5) \rightarrow q_{13}$	$N(q_5, q_4) \rightarrow q_{13}$
	$N(q_4, q_4) \rightarrow q_{13}$	$N(q_5, q_5) \rightarrow q_{13}$

Figure 5.7: Nonces between Alice and Bob

Chapter 3.2.2.

Automaton	Wrong_Belief	
States	q_0 q_1 q_2 q_3 q_4 q_5 q_6 q_{13}	
Final States	q_{13}	
Transitions		
	$0 \rightarrow q_0$	
	$s(q_0) \rightarrow q_0$	$agt(q_0) \rightarrow q_3$
	$A \rightarrow q_1$	$agt(q_1) \rightarrow q_4$
	$B \rightarrow q_2$	$agt(q_2) \rightarrow q_5$
	$U(q_{13}, q_{13}) \rightarrow q_{13}$	
	$c\_init(q_4, q_5, q_3) \rightarrow q_{13}$	$c\_init(q_4, q_5, q_4) \rightarrow q_{13}$
	$c\_resp(q_5, q_4, q_3) \rightarrow q_{13}$	$c\_resp(q_5, q_4, q_5) \rightarrow q_{13}$
	$c\_init(q_5, q_4, q_3) \rightarrow q_{13}$	$c\_init(q_5, q_4, q_5) \rightarrow q_{13}$
	$c\_resp(q_4, q_5, q_3) \rightarrow q_{13}$	$c\_resp(q_4, q_5, q_4) \rightarrow q_{13}$

Figure 5.8: Alice and Bob do not really communicate with each other

In Figure 5.8, "Wrong_Belief" recognizes all the possible wrong beliefs for communications between Alice and Bob. Again by checking the intersection of "Wrong_Belief" with the approximation automaton, it is checked that the approximation automaton can recognize wrong beliefs. In other words, it is checked if the computation of the reachable states can lead Alice or Bob to wrong beliefs.

For this version of the Needham-Schroeder protocol, the intersection of this automaton with the approximation one is empty, so the protocol satisfies the property.

How are approximation results used? The secrecy and the authentication properties are verified by the protocol. These results can be used in our inductive proof.

In the inductive proof for this protocol[8], the theorems corresponding to those properties are:

- for the secrecy:

[8]http://www.cl.cam.ac.uk/Research/HVG/Isabelle/library/HOL/Auth/

- Spy_not_see_NAB: the Spy does not see the nonce sent in the message *NS1* if Alice and Bob are secure;
- Spy_not_see_NBA: the Spy does not see the nonce sent in the message *NS2* if Alice and Bob are secure.

With the automata technique, it is proven that the intruder never catches the nonces exchanged between Alice and Bob. If the intruder cannot see the nonces of Alice and Bob, then he does not see the one sent in *NS1* and the one sent in *NS2*.

The first time the protocol was proven with Isabelle, these two theorems could have had been added as axioms in the Isabelle specification using the result of the approximation instead of having been proven by the user.

- for the authentication:
  - A_trusts_NS2: if Alice receives message *NS2* and has used NAB to start a run, then Bob has sent message NS2;
  - B_trusts_NS3: if Bob receives message *NS3* and has used NBA in NS2, then Alice has sent message NS3.

The Genet and Klay approach checks that when Alice wants to establish a communication with Bob, after *NS2* she really speaks with him. We also verified that when Bob thinks that he is responding to Alice, he really speaks with Alice after *NS3*. So the theorems "A_trusts_NS2" and "B_trusts_NS3" are also true. Once again instead of having to prove these theorems with Isabelle, the result of the approximation would have allowed the user to insert them in the Isabelle specification as axioms (cf. Figure 5.9).

For the Needham-Schroeder-Lowe protocol, the secrecy and authentication properties by approximation are verified. The next section presents the work done on other basic protocols.

Theory NS_Public = Public:

• • •

axioms

Spy_not_see_NAB "[|Says A B {|Crypt(pubK B) {|Nonce NAB0, Agent A|}|} : set evs;
A ~: bad; B ~: bad; evs : ns_public|] ⟹
Nonce NAB0 ~: analz (spies evs)"

Spy_not_see_NBA "[|Says B A {|Crypt (pubK A) {|Nonce NAB0, Nonce NBA0,
Agent B|}|} : set evs;
A ~: bad; B ~: bad; evs : ns_public|] ⟹
Nonce NBA0 ~: analz (spies evs)"

A_trusts_NS2 "[|Says A B {|Crypt(pubK B) {|Nonce NAB0, Agent A|}|} : set evs;
Says B' A {|Crypt(pubK A) {|Nonce NAB0, Nonce NBA0,
Agent B|}|} : set evs;
A ~: bad; B ~: bad; evs : ns_public|] ⟹
Says B A {|Crypt(pubK A) {|Nonce NAB0, Nonce NBA0,
Agent B|}|} : set evs"

B_trusts_NS3 "[|Says B A {|Crypt (pubK A) {|Nonce NAB0, Nonce NBA0,
Agent B|}|} : set evs;
Says A' B {|Crypt (pubK B) {|Nonce NBA0|}|} : set evs;
A ~: bad; B ~: bad; evs : ns_public|] ⟹
Says A B {|Crypt (pubK B) {|Nonce NBA0|}|} : set evs"

end

Figure 5.9: New inductive specification of the Needham-Schroeder-Lowe protocol

5.3.1.2 Properties verified on the other protocols

Since the secrecy properties are different from one protocol to another, this section details the verification done on the other simple protocols.

Needham-Schroeder symmetric key In this version of the protocol, Alice and Bob trust a server (S in Figure 5.10) to create a session key Kab. Then they use this key to authenticate each other with a nonce.

Message 1:	$A \Longrightarrow S : A, B, Na$
Message 2:	$S \Longrightarrow A : \{Na, B, Kab, \{Kab, A\}_{Kbs}\}_{Kas}$
Message 3:	$A \Longrightarrow B : \{Kab, A\}_{Kbs}$
Message 4:	$B \Longrightarrow A : \{Nb\}_{Kab}$
Message 5:	$A \Longrightarrow B : \{Nb - 1\}_{Kab}$

Figure 5.10: Needham-Schroeder symmetric key protocol

For this protocol it is proven that Kab and Nb can not be discovered by the intruder, and so at the end of the protocol, Alice and Bob communicate with each other.

Needham-Schroeder This protocol has already been introduced in Chapter 1, as well as its flaw (cf. Figure 1.4). The proof of the same properties also failed on the approximation automaton computed with the new approximation function.

In Section 4.2, it was indicated that for a non-empty intersection, the approximation could be useful to validate the existence of a flaw with Isabelle. The Timbuk file and the approximation automaton of that version of Needham-Schroeder are available in Appendix D and Appendix E. For this protocol, the secrecy is not verified; this means that there is at least one transition from the information captured, $add(q \cdots)$, to the state q_{13} in the automaton. In the approximation automaton, 19

Message 1:	$A \Longrightarrow B : \{Na, A\}_{Kb}$
Message 2:	$B \Longrightarrow A : \{Na, Nb\}_{Ka}$
Message 3:	$A \Longrightarrow B : \{Nb\}_{Kb}$

Figure 5.11: Needham-Schroeder protocol

transitions of the form $add(q \cdots) \rightarrow q_{13}$ can be found:

$add(q_{53}) \rightarrow q_{13}$	$add(q_{62}) \rightarrow q_{13}$	$add(q_{76}) \rightarrow q_{13}$
$add(q_{82}) \rightarrow q_{13}$	$add(q_3) \rightarrow q_{13}$	$add(q_4) \rightarrow q_{13}$
$add(q_5) \rightarrow q_{13}$	$add(q_{42}) \rightarrow q_{13}$	$add(q_{33}) \rightarrow q_{13}$
$add(q_{23}) \rightarrow q_{13}$	$add(q_{60}) \rightarrow q_{13}$	$add(q_{51}) \rightarrow q_{13}$
$add(q_{88}) \rightarrow q_{13}$	$add(q_{43}) \rightarrow q_{13}$	$add(q_{34}) \rightarrow q_{13}$
$add(q_{24}) \rightarrow q_{13}$	$add(q_{31}) \rightarrow q_{13}$	$add(q_{52}) \rightarrow q_{13}$
$add(q_{13}) \rightarrow q_{13}$		

Only the rules that involve the honest agents' nonces (for example such that $add(q \cdots) \rightarrow q_{13}$ and $N(q_4, q_5, qt0) \rightarrow q \cdots$ are in the automaton) are kept. Thus only 4 transitions remain:

$add(q_{76}) \rightarrow q_{13}$	$add(q_{82}) \rightarrow q_{13}$
$add(q_{60}) \rightarrow q_{13}$	$add(q_{51}) \rightarrow q_{13}$

By looking at the states at the right side of the transitions and at the approximation function, messages leading to the non-empty intersection are found:

[U(LHS, mesg(agt(q_2), agt(q_0), encr(pubkey(agt(q_0)), agt(q_2), cons(N(a_3, b_3, t_3),
N(agt(q_2), agt(q_0), qt0))))) → q_{13}]
⟶ [LHS → q_{13} agt(q_2) → q_5 serv(q_7) → q_6 agt(q_0) → q_3 N(q_5, q_3, qt0) → q_{23}
N(a_3, b_3, t_3) → q_{51} cons(q_{51}, q_{23}) → q_{52} pubkey(q_3) → q_{25} encr(q_{25}, q_5, q_{52}) → q_{13}
mesg(q_5, q_3, q_{13}) → q_{13}]

[U(LHS, mesg(agt(q_1), agt(q_0), encr(pubkey(agt(q_0)), agt(q_1), cons(N(a_6, b_6, t_6),
N(agt(q_1), agt(q_0), qt0))))) → q_{13}]
⟶ [LHS → q_{13} agt(q_1) → q_4 serv(q_7) → q_6 agt(q_0) → q_3 N(q_4, q_3, qt0) → q_{33}
N(a_6, b_6, t_6) → q_{60} cons(q_{60}, q_{33}) → q_{61} pubkey(q_3) → q_{25} encr(q_{25}, q_4, q_{61}) → q_{13}
mesg(q_4, q_3, q_{13}) → q_{13}]

[U(LHS, mesg(agt(q_2), agt(q_0), encr(pubkey(agt(q_0)), agt(q_2), N(a_12, b_12, t_12)))) → q_{13}]
⟶ [LHS → q_{13} agt(q_2) → q_5 serv(q_7) → q_6 agt(q_0) → q_3 N(a_12, b_12, t_12) → q_{76}
pubkey(q_3) → q_{25} encr(q_{25}, q_5, q_{76}) → q_{13} mesg(q_5, q_3, q_{13}) → q_{13}]

[U(LHS, mesg(agt(q_1), agt(q_0), encr(pubkey(agt(q_0)), agt(q_1), N(a_15, b_15, t_15)))) → q_{13}]
⟶ [LHS → q_{13} agt(q_1) → q_4 serv(q_7) → q_6 agt(q_0) → q_3 N(a_15, b_15, t_15) → q_{82}
pubkey(q_3) → q_{25} encr(q_{25}, q_4, q_{82}) → q_{13} mesg(q_4, q_3, q_{13}) → q_{13}]

The user knows now that in the inductive proof he must study carefully the secrecy on the last two messages.

After having proven regularity and unicity lemmas (cf. Section 4.2.1), the user can go straight to the proof of the secrecy of nonce NBA0 in the second message:

"[|Says B A {|Crypt (pubK A) {|Nonce NAB0, Nonce NBA0|}|} : set evs;
A ∼: bad; B ∼: bad; evs : ns_public|] ⟹ Nonce NBA0 ∼: analz (spies evs)"

But the proof of this theorem terminates by "false", so the secrecy of NBA0 is not guaranteed by the protocol.

Otway Rees simplified In this version of the protocol [AN96], Alice and Bob trust a server (S in Figure 5.12) to create a session key, Kab. It was verified that the session key Kab could not be caught by the intruder and that Alice and Bob really communicate with the person they believe they are communicating at the end.

Message 1: $A \Longrightarrow B : A, B, Na$

Message 2: $B \Longrightarrow S : A, B, Na, Nb$

Message 3: $S \Longrightarrow B : \{Na, A, B, Kab\}_{Kas}, \{Nb, A, B, Kab\}_{Kbs}$

Message 4: $B \Longrightarrow A : \{Na, A, B, Kab\}_{Kas}$

Figure 5.12: Otway Rees simplified protocol

Otway Rees modified by us In this version of the protocol, we modified the previous protocol by removing the names of the agents from the information encrypted. The secrecy property of the session key Kab is still verified on this version of the protocol. Nevertheless, the verification of the authenticity property (Alice and Bob really communicate with the person they think they are) failed.

Message 1: $A \Longrightarrow B : A, B, Na$

Message 2: $B \Longrightarrow S : A, B, Na, Nb$

Message 3: $S \Longrightarrow B : Na, \{Na, Kab\}_{Kas}, \{Nb, Kab\}_{Kbs}$

Message 4: $B \Longrightarrow A : Na, \{Na, Kab\}_{Kas}$

Figure 5.13: Otway Rees protocol modified by us

This was not surprising as the Otway-Rees protocol is known to be flawed when the names of the agents are not sent with the session key [CJ97]. When Alice receives the last message, she has no way to know that she really shares a key with the person she intended to.

Figure 5.14 shows how the intruder can manage to make Alice use a session key thinking that this key is also known by Bob. The attack is simple, Alice starts to initiate a communication with Bob. Yves, the intruder, has then enough information to send a fraudulent message to the server. This generates a session key which is

sent to Yves in the third message. Then Yves sends Alice the last message. From now on, each time Alice receives a message encrypted with the key Kay, she will think that Bob sent that message.

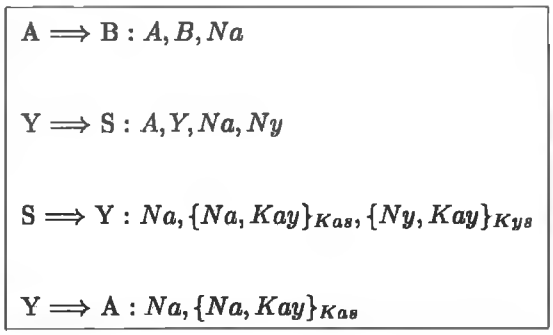


Figure 5.14: Otway Rees protocol attack

Woo Lam Pi3 This protocol is a one way authentication protocol, that means if Alice initiates a communication with Bob, then in the end, Bob is sure to communicate with Alice.

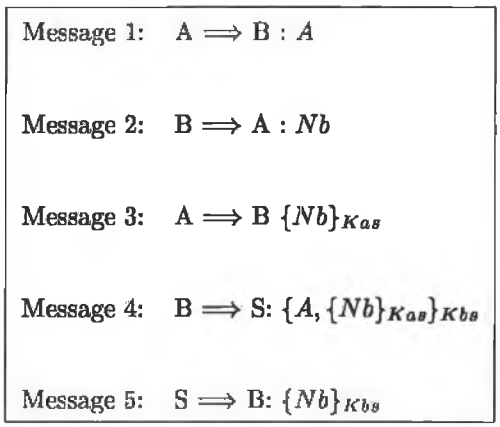


Figure 5.15: Woo Lam protocol

There was no secrecy property to check on this protocol. The authentication property that when Bob receives the last message, he really receives the nonce created to communicate with Alice, was checked. The property was verified by the protocol.

Andrew Secure RPC In this protocol Alice and Bob initially shared a key Kab, and Alice, who wants to communicate with Bob, will trust him to create a session key $K'ab$ (cf. Figure 5.16).

Message 1:	$A \Longrightarrow B : A, \{Na\}_{Kab}$
Message 2:	$B \Longrightarrow A : \{succNa, Nb\}_{Kab}$
Message 3:	$A \Longrightarrow B : \{succNb\}_{Kab}$
Message 4:	$B \Longrightarrow A : \{K'ab, N'b\}_{Kab}$

Figure 5.16: Andrew Secure RPC protocol

It was proven that the intruder could not catch the session key $K'ab$. It was also checked that when Alice received a session key, this key was created by Bob and that when she thinks she is speaking with Bob, she is indeed speaking with him.

5.3.2 Transport Layer Security protocol

If you have bought something on the Internet, you almost certainly saw on the web page a message such as "Secure Mode. SSL (Secure Socket Layer) technology is used to protect your personal information". SSL [KFK96] was originally developed by "Netscape Communications Corporation" in order to protect information conveyed by HTTP applications. Basically, SSL is a protocol where server and client machines compute session keys from nonces they have exchanged. The latest version of this protocol is studied here: Transport Layer Security [Gro96b] (TLS for short).

Let us assume that the client wants to buy something on a commercial website (the server). To conclude the transaction, the client will have to give his credit card number. The credit card number will be encrypted and sent to the server. But to encrypt it, the client needs an encryption key and the client needs to be sure that he is communicating with the server. TLS is there for that. Before critical information is exchanged, at least two actors need to agree on a common secret and be able to identify each other. This is the function of the *handshake* protocol and this is the

part of TLS that is verified here.

Figure 5.17 shows the messages exchanged in TLS. The client initializes the communication by sending his name, a nonce, a session identifier and a set of preferences for encryption and compression. The server replies with a nonce, the session identifier that he received and his encryption and compression preference. Then he sends another message that contains his public key certificate. The client can also send his public key certificate. The client generates a 48-byte pre-master-secret (PMS) and sends it encrypted with the server's public key. The client can also hash the sever's name and nonce, and the pre-master-secret to send them encrypted with his private key (if he has sent his public key certificate to the server). Now both, the client and the server calculate the master-secret from the nonces exchanged and the PMS using a pseudo-random number function (PRF). Then they hash all the previous messages and the PRF, and they encrypt this piece of information with a session key created with the PRF and the nonces. Finally they send this cipher text to each other to confirm the communication.

5.3.2.1 Related work on SSL/TLS

In [WS96], Wagner and Schneier give their conclusion on their analysis of SSL 3.0. They do not use any formal technique for this work. They check the resistance of SSL to well known attacks of protocols, such as replay attacks, cryptanalysis techniques, etc. Their conclusion is that the protocol has some flaws that can be corrected without major modifications of the protocol (i.e. passes from the cryptographic MAC to the HMAC one). Their flaws cannot be found by approximation because these flaws are cryptanalytic flaws and the approximation cannot detect these flaws.

In his PhD thesis [Die97], Dietrich uses the Non-monotonic Cryptographic Protocols (NCP) belief logic to analyze SSL 3.0. He proved that SSL was secure against attacks from a passive eavesdropper. Most of his work was done manually.

In [MSS98], Mitchell, Shmatikov and Stern use a model-checking technique approach to verify SSL. They start from a simple model of the protocol and check the properties they want. Then they insert new information into their model and check new properties. They repeat this process until they arrive at a model that looks

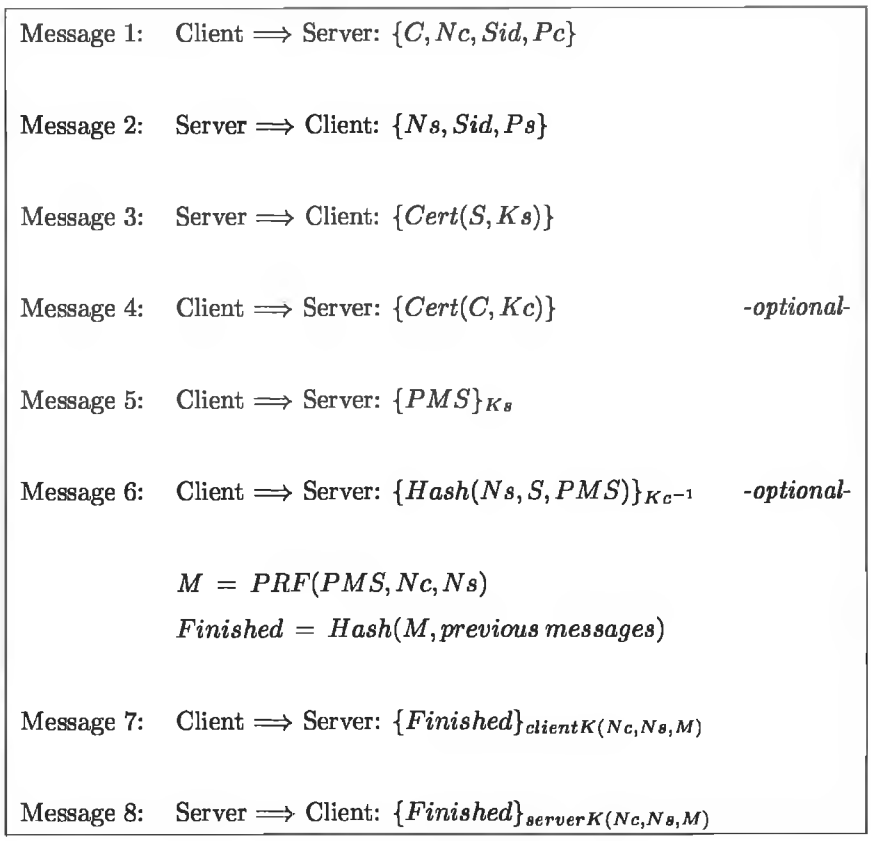


Figure 5.17: TLS protocol

like SSL. But their technique works only for a small number of participants in the protocol, otherwise they will crash the computer memory.

In [Pau99], Paulson, with his inductive approach, successfully used on simple protocols, presents a verification of the TLS handshake protocol. He verified secrecy and authenticity properties on a simplified version of TLS (the same as the one we use in our verification). He wrote the proof script for the Isabelle theorem prover in 2 weeks. And his results are for an unbound number of participants and runs of the protocol.

In [BPST02], a symbolic model-checker NuMAS [BC01] is used to verify a simplified version of SSL/TLS which is modeled with MATL (MultiAgent Temporal Logic). MATL can represent both time and beliefs. With this logic they were able to verify that the protocol guarantees the freshness of the secret information (PMS

and session key).

5.3.2.2 Modelling TLS

A simplified version of TLS (cf. Figure 5.18) is verified in this thesis:

- the optional messages (Messages 4 and 6 in Figure 5.17) are removed;
- Messages 2 and 3 are gathered together;
- In Messages 7 and 8, the only information to be hashed is the information to be exchanged.

These simplifications have been done to reduce the computation time of the approximation. Without the second simplification (gathering together Messages 2 and 3), the computation requires substantial memory. The reason for this is that in order to send the last message the client needs to have received Message 2 and Message 3. This implies a rule of the form "U(Message 3, Message 4) $\rightarrow$ Message 7" in the term rewriting system criticize the protocol steps. Computing "U(Message 3, Message 4)" requires more time as the size of the protocol trace increases.

The pseudo-random number function (PRF) is modeled as a hash function. The reason is that the approximation for a PRF is the same as the one for a hash function.

In the full version Messages 4 and 6 are optional so removing them should not affect the correctness of the protocol.

Gathering together information in Messages 2 and 3 does not affect the correctness of the full protocol as the information can be caught by the intruder whether it is sent in two separate messages or in one.

Finally hashing message components rather than messages in Message 7 and 8 also does not affect the correctness here. In our approach, the intruder cannot guess complex information (concatenation of two information, encrypted information, etc.) from the hashing of that information and some of the information that composed the complex information; he needs to know all the information that composed the complex information. This means that if the intruder knows the hashing of $\{PMS\}_{Ks}$ and the key Ks then he cannot deduce $\{PMS\}_{Ks}$ as long as he does

Message 1:	Client $\Longrightarrow$ Server: $\{C, Nc, Sid, Pc\}$
Message 2:	Server $\Longrightarrow$ Client: $\{Ns, Sid, Ps, Cert(S, Ks)\}$
Message 3:	Client $\Longrightarrow$ Server: $\{PMS\}_{Ks}$
	$M = Hash(PMS, Nc, Ns)$
	$Finished = Hash(M, C, Nc, Sid, Pc, S, Ns, Ps, PMS)$
Message 4:	Client $\Longrightarrow$ Server: $\{Finished\}_{clientK(Nc,Ns,M)}$
Message 5:	Server $\Longrightarrow$ Client: $\{Finished\}_{serverK(Nc,Ns,M)}$

Figure 5.18: Simplified version of TLS

not get PMS. Thus, for our intruder, assuming that he knows Ks, the hashing of $\{PMS\}_{Ks}$ and the hashing of PMS are equivalent, in the sense that he needs to know PMS to deduce $\{PMS\}_{Ks}$.

The TRS[9] for TLS contains five rules for the protocol steps and the intruder's abilities and commutativity rules introduced in Section 3.2.2. A rule to express that when the intruder knows a complex key, he has access to the information encrypted with this key, has to be added: U(key(x, y, z), encr(key(x, y, z), a, m)) $\longrightarrow$ U(LHS, add(m)). The TLS also contains 2 rules to express the authentication properties; one of the form U(message_sent, message_received) $\longrightarrow$ U(LHS, c_init($\cdots$)) for the Client and one of the form U(message_sent, message_received) $\longrightarrow$ U(LHS, c_resp($\cdots$)) for the Server.

The initial automaton is also the same as the one introduced in Section 3.2.2 assuming that A is Client and B is Server. Transitions have been also added to deal with the new information types used in TLS. The intruder knows the session identifier created by the unsafe agents, so the transitions sid(q_3, q_3) $\rightarrow q_{13}$; sid(q_3, q_4)

[9]The syntax and the semantics used are summarized in Table 4.2.

$\rightarrow q_{13}$ and $\text{sid}(q_3, q_5) \rightarrow q_{13}$ are found in the automaton. For the same reason, the pre-master-secret the transitions $\text{pms}(q_3, q_3) \rightarrow q_{13}$; $\text{pms}(q_3, q_4) \rightarrow q_{13}$ and $\text{pms}(q_3, q_5) \rightarrow q_{13}$ have been added to the automaton. The fact that the intruder can hash information and create complex keys must be expressed, so the automaton also has the transitions $\text{hash1}(q_3, q_{13}) \rightarrow q_{13}$ and $\text{key}(q_{13}, q_{13}, q_{13}) \rightarrow q_{13}$.

After having extended IS2TiF to deal with the session identifier and the pre-master-secret, IS2TiF is used to compute the approximation function for TLS.

As for the previous protocols two computations were done; one where an agent is allowed to speak to himself and the other where an agent cannot talk to himself. It took 27 minutes to get the approximation automaton for the first case and 6 hours 23 minutes for the second case. As for the small protocols, the execution times highlight the fact that the complexity of the approach is exponential. Nevertheless, we can be happy as in less than a man day we can verify a medium size protocol.

5.3.2.3 TLS verification

Five properties have been verified for TLS:

- the intruder does not catch the pre-master-secret between the trusted agents (Server and Client);
- the intruder does not catch the master-secret between the trusted agents (Server and Client);
- the intruder does not catch the session key between the trusted agents (Server and Client);
- at the end of the protocol, when the Client thinks that he communicates with the Server, he really does communicate with the Server;
- at the end of the protocol, when the Server thinks that he responds to the Client, he really does communicate with the Client.

Secrecy Properties When the approximation automaton has been computed, it is possible to check secrecy guarantees about:

- the pre-master-secret;
- the master secret (M on Figure 5.18);
- server and client session keys.

For each property, an automaton of the negation of the property ("the intruder has...") is automatically generated by IS2TiF. Thus Figure 5.19, Figure 5.20 and Figure 5.21 respectively model "the intruder caught the pre-master-secret", "the intruder caught the master-secret" and "the intruder caught the session keys" negation automata.

Then intersections of those automata with the approximation automaton are computed. Those intersections are *empty*, so the properties are verified for the protocol.

Automaton	PMS_Not_Secret		
States	q_0 q_1 q_2 q_3 q_4 q_5 q_6 q_{13}		
Final States	q_{13}		
Transitions			
	$0 \rightarrow q_0$	$\text{suc}(q_0) \rightarrow q_0$	$\text{agt}(q_0) \rightarrow q_3$
	$A \rightarrow q_1$	$\text{hash1}(q_4, q_{12}) \rightarrow q_{13}$	$\text{agt}(q_1) \rightarrow q_4$
	$B \rightarrow q_2$	$\text{agt}(q_2) \rightarrow q_5$	
	$U(q_{13}, q_{13}) \rightarrow q_{13}$		
	$\text{pms}(q_5, q_5) \rightarrow q_{13}$	$\text{pms}(q_4, q_5) \rightarrow q_{13}$	
	$\text{pms}(q_5, q_4) \rightarrow q_{13}$	$\text{pms}(q_4, q_4) \rightarrow q_{13}$	

Figure 5.19: PMS between Client and Server

Authentication properties Two authentication properties can be checked on our protocol one for each role, Server and Client. At the end of the protocol the client must be sure that he speaks to the server and vice versa.

The verification is carried out using the terms *c_resp* and *c_init* and the same automaton as for the other protocols.

The intersection of the approximation automaton with the one for the property was empty. So the property is verified by the protocol.

Automaton	PRF_Not_Secret		
States	q_0 q_1 q_2 q_3 q_4 q_5 q_6	q_9 q_{10} q_{11} q_{12} q_{13}	
Final States	q_{13}		
Transitions			
	$0 \rightarrow q_0$	$suc(q_0) \rightarrow q_0$	$agt(q_0) \rightarrow q_3$
	$A \rightarrow q_1$	$agt(q_1) \rightarrow q_4$	
	$B \rightarrow q_2$	$agt(q_2) \rightarrow q_5$	
	$U(q_{13}, q_{13}) \rightarrow q_{13}$		
	$pms(q_5, q_5) \rightarrow q_9$	$pms(q_4, q_5) \rightarrow q_9$	
	$pms(q_5, q_4) \rightarrow q_9$	$pms(q_4, q_4) \rightarrow q_9$	
	$N(q_5, q_5) \rightarrow q_{10}$	$N(q_4, q_5) \rightarrow q_{10}$	
	$N(q_5, q_4) \rightarrow q_{10}$	$N(q_4, q_4) \rightarrow q_{10}$	
	$cons(q_{10},q_{10}) \rightarrow q_{11}$	$cons(q_9,q_{11}) \rightarrow q_{12}$	
	$hash1(q_4, q_{12}) \rightarrow q_{13}$	$hash1(q_5, q_{12}) \rightarrow q_{13}$	
	$hash1(q_3, q_{12}) \rightarrow q_{13}$		

Figure 5.20: Master secret between Client and Server

Consequence in Isabelle proof As for the simple protocols, the previous results can be inserted as axioms in the Isabelle proof.

These axioms were theorems in Paulson's proof, as he had to prove them:

- for the secrecy:
  - Spy_not_see_PMS: the intruder cannot see the pre-master-secret between honest agent;
    [| Notes A {|Agent B, Nonce PMS|} : set evs;
    evs : tls; A ∼: bad; B ∼: bad |] ⟹
    Nonce PMS ∼: analz(knows Spy evs)
  - Spy_not_see_MS: the intruder cannot see the master-secret;
    [| Notes A {|Agent B, Nonce PMS|} : set evs;
    evs : tls; A ∼: bad; B ∼: bad |] ⟹
    Nonce (PRF (PMS, NA, NB)) ∼: analz(knows Spy evs)
  - ClientK_not_spied: the intruder cannot guess the client's key if the key has not been sent to him;

Automaton	SesK_Not_Secret		
States	q_0 q_1 q_2 q_3 q_4 q_5 q_6	q_9 q_{10} q_{11} q_{12} q_{13} q_{14}	
Final States	q_{13}		
Transitions			
	$0 \rightarrow q_0$	$suc(q_0) \rightarrow q_0$	$agt(q_0) \rightarrow q_3$
	$A \rightarrow q_1$	$agt(q_1) \rightarrow q_4$	
	$B \rightarrow q_2$	$agt(q_2) \rightarrow q_5$	
	$U(q_{13}, q_{13}) \rightarrow q_{13}$		
	$pms(q_5, q_5) \rightarrow q_9$	$pms(q_4, q_5) \rightarrow q_9$	
	$pms(q_5, q_4) \rightarrow q_9$	$pms(q_4, q_4) \rightarrow q_9$	
	$N(q_5, q_5) \rightarrow q_{10}$	$N(q_4, q_5) \rightarrow q_{10}$	
	$N(q_5, q_4) \rightarrow q_{10}$	$N(q_4, q_4) \rightarrow q_{10}$	
	$cons(q_{10},q_{10}) \rightarrow q_{11}$	$cons(q_9,q_{11}) \rightarrow q_{12}$	
	$hash1(q_4, q_{12}) \rightarrow q_{14}$	$hash1(q_5, q_{12}) \rightarrow q_{14}$	
	$hash1(q_3, q_{12}) \rightarrow q_{14}$		
	$cons((q_{10}, q_{14}) \rightarrow q_{15}$	$cons((q_{10}, q_{15}) \rightarrow q_{16}$	
	$key(q_4, q_5, q_{16}) \rightarrow q_{13}$	$key(q_5, q_4, q_{16}) \rightarrow q_{13}$	
	$key(q_5, q_5, q_{16}) \rightarrow q_{13}$	$key(q_4, q_4, q_{16}) \rightarrow q_{13}$	

Figure 5.21: Session key between Client and Server

[| Notes A {|Agent B, Nonce PMS|} : set evs;
Says A Spy (Key (clientK (NA, NB, PRF (PMS, NA, NB)))) ∼: set evs;
evs : tls; A ∼: bad; B ∼: bad |] ⟹
Key (clientK (NA, NB, PRF (PMS, NA, NB))) ∼: parts(knows Spy evs)

- ServerK_not_spied: the intruder cannot guess the server's key if the key has not been sent to him;
[| Notes A {|Agent B, Nonce PMS|} : set evs;
Says B Spy (Key (serverK (NA, NB, PRF (PMS, NA, NB)))) ∼: set evs;
evs : tls; A ∼: bad; B ∼: bad |] ⟹
Key (serverK (NA, NB, PRF (PMS, NA, NB))) ∼: parts(knows Spy evs)

- for the authentication:
  - Client_Guarantee: the server had created the last message received by

the client;

[| X = Crypt (serverK (NA, NB, M))
(Hash {| Number SID, Nonce M, Nonce NA, Number PA, Agent A,
Nonce NB, Number PB, Agent B, Nonce PMS|});
M = PRF (PMS, NA, NB); evs : tls; A ~: bad; B ~: bad;
Says B Spy (Key (serverK (NA, NB, M))) ~: set evs;
Notes A {|Agent B, Nonce PMS|} : set evs; X : parts(knows Spy evs) |] ⟹
Says B A X : set evs

– Server_Guarantee: the client had created the last message received by the server;

[| X = Crypt (clientK (NA, NB, M))
(Hash {| Number SID, Nonce M, Nonce NA, Number PA, Agent A,
Nonce NB, Number PB, Agent B, Nonce PMS|});
M = PRF (PMS, NA, NB); evs : tls; A ~: bad; B ~: bad;
Says A Spy (Key (clientK (NA, NB, M))) ~: set evs;
Notes A {|Agent B, Nonce PMS|} : set evs; X : parts(knows Spy evs) |] ⟹
Says A B X : set evs

Then the proofs of the other lemmas regarding possibility properties, unicity properties, forwarding lemmas, the validity of the certificates sent, etc. did not take long, as they could be deduced from other protocols' proofs. The inductive proof was done in one day, by picking the properties to prove from the proof of Paulson[10] and using other protocols' proofs[11] to prove them.

The whole verification (approximation+inductive proof) took 2 days. Without knowing the remaining properties to prove by induction in advance, the process would have taken more time.

After having extended our prototype to deal with the pre-master-secret and complex keys built with three pieces of information, the computation of the approximation automaton with Timbuk and our function took 27 minutes when an agent can talk to himself and 6 hours 23 minutes when an agent cannot talk to himself.

[10] http://www.cl.cam.ac.uk/Research/HVG/Isabelle/library/HOL/Auth/TLS.html
[11] http://www.cl.cam.ac.uk/Research/HVG/Isabelle/library/HOL/Auth/index.html

The verification of properties took only 20 minutes.

5.3.3 Conclusion

Now that the prototype and the results of our experiments have been introduced, it is time to compare our solution with those of others.

5.3.3.1 Genet's approximations

With Timbuk, approximation automata with Genet's approximations had been computed.

Figure 5.22 compares the computation times. Our approximation function is faster than the ancestor most of the time. The only time the ancestor approximation is faster is for a protocol for which no secrecy has to be checked. The basic approximation without user helps does not terminate. One interesting point is the preparation time required to before the computation. For the basic approximation and our approximation, this time is null when for the ancestor approximation it is more than 10 minutes. For the ancestor approximation, the user has to check the states used in every approximation rule. In the other approaches everything is done by the computer.

Protocol Name	Basic approximation	Ancestor approximation	Our approximation
Needham-Schroeder	out of memory	3min 27s	3min 50s
Needham-Schroeder-Lowe	out of memory	9min 50s	1min 59s
Needham-Schroeder symmetric key	out of memory	10min 34s	6min 10s
Woo Lam	out of memory	9min 04s	14min
Otway Rees	out of memory	19min 45s	17 min 12s

Figure 5.22: Time for computation of approximation automaton

With the basic approximation no properties were checked, as no automaton was produced. At the same time, the ancestor approximation was less efficient to verify

properties as it failed with protocols with secrecy properties as shown on Figure 5.23.

Hence, our approximation seems better adapted for the cryptographic protocol verification for these experiments. More experiments should confirm whether this is in fact the case. The reasons are:

- the basic approximation failed to give us an automaton, and
- the ancestor approximation is not suitable for secrecy properties.

Protocol Name	Basic approximation	Ancestor approximation	Our approximation
Needham-Schroeder	no automaton	may be flawed	may be flawed
Needham-Schroeder-Lowe	no automaton	may be flawed	secured
Needham-Schroeder symmetric key	no automaton	may be flawed	secured
Woo Lam	no automaton	safe	safe
Otway Rees Simplified	no automaton	may be flawed	secured

Figure 5.23: Verification of secrecy and authentication properties

5.3.3.2 Other proof approaches

Since this work started, other approaches have been developed to "automatically prove" protocols. This section introduces these approaches and compares them to our approach.

Blanchet Blanchet [Bla01] checks the secrecy by whether the attacker can or cannot access confidential information. He uses Horn clauses to model the protocol and the attacker. Basically, he looks for the answer to the question *attacker(s)* where *s* is the confidential information and *attacker* the predicate for the intruder. He implemented a search algorithm in Prolog that is guaranteed to terminate and is based on two abstractions:

- Any step of a protocol can be repeated several times if the previous step had been executed at least once.
- The freshness of nonces is modeled by considering fresh values as function of the messages previously received by the creator of the value. This means that for the same previous messages, the same nonces are found.

Thus the verification is done for an unbounded number of sessions and for an unbounded message size. When the tool finds an attack, it might be a false attack due to the abstraction on nonces.

Hermes In [BLP02, BLP03], the secrecy is checked by first looking at the evolution of secret information over the set of reachable messages. Then conditions to guarantee the secrecy are deduced, and the verification that the protocol guarantees those conditions is done. The termination of the computation is guaranteed by an algorithm based on the following abstractions:

- two agents are considered: an agent A and an agent I that models the intruder and all the agents different from A.
- four sessions are considered: a session between A and I, a session between I and A, one particular session between A and A and one session that models all the sessions between A and A that are different from the specific session between As.

So with this tool, the secrecy is guaranteed for an unbounded number of sessions, an unbounded message size and an unbounded number of agents for self-talking protocols.

Securify Securify[12] [Cor02] can be seen as the automated version of Paulson's approach for the verification of the secrecy. This tool is based on the conditions defined in [CMR01] to guarantee the secrecy; for every step of the protocol and every part of the message sent:

[12] http://www.lsv.ens-cachan.fr/~cortier/EVA/eva-comp.php

- either this part is a freshly generated nonce, thus it cannot compromise the secret
- or this part was already sent over the network, encrypted with at least the same set of keys,
- or this part is a secret but it is encrypted with a protected key.

Securify runs these tests, if they fail, a backward search is done to get more information on the messages sent over the network, and then it tests again. The process (backward search + tests) is repeated as many times as is necessary. The tool returns:

- "yes" if secret is guaranteed;
- "fail" if the proof fails. No conclusion about the secrecy can be drawn but the "failure" tree returned might help to build an attack;
- no answer the computation does not terminate.

So with this tool the secrecy is guaranteed for an unbounded number of sessions, an unbounded message size and an unbounded number of agents.

Spicasso Aziz [Azi03] introduced Spicasso, a static analyzer for the Spi-calculus [AG98]. This tool was used to check the secrecy and authenticity on some simple protocols. The results are guaranteed for an unbounded number of sessions and an unbounded number of agents. The verification is also guaranteed to terminate. The run-times of the tests carried out are unpublished.

With our prototype, the verification is also done for an infinite number of sessions and agents. Unlike the other tools [Bla01, BLP02, Cor02], the verification of secrecy and authentication properties can be performed. The run-times of the tests carried out are a bit slower than those of other tools [Bla01, BLP02, Cor02], if it is assumed that protocols do not allow an agent to talk to himself. Even when protocols allow an agent to talk to himself, IS2TiF plus Timbuk are still slower (cf. Figure 5.24).

Protocol Name	Blanchet	Hermes	Securify	IS2TiF+Timbuk
Needham-Schroeder shared key	0.76s	0.04s	-	41s
Needham-Schroeder	0.07s	0.01s	0.001s	18s
Needham-Schroeder-Lowe	0.06s	0.02s	0.001s	16s
Woo-Lam	-	0.06s	0.0001s	31s

Figure 5.24: Comparison table of automatic proof approaches

The PC performances are not responsible as tests have been performed for two other tools on less powerful PCs [13]. Our computation is slower as it repeats two searches until no critical pair is found:

1. to find a "critical pair", so it has to check all the terms of the automaton on which rewrite rules can be applied;
2. to find the good approximation rule to apply on the critical pair.

These processes are done by Timbuk. The version used for these experiments runs exponentially as the critical pairs search of the current algorithm applies all the possible substitutions to the term rewriting system for every automaton $\mathcal{A}_{fi}$. A little optimization to reduce the computational time was implemented in Timbuk, $\mathcal{A}_{fi+1}$ is computed by applying on $\mathcal{A}_{fi}$ one rewrite rule with several substitutions not just one rule and one substitution. With the new release of Timbuk we could expect faster run-times as the completion algorithm should be optimized.

Nevertheless, more experiments on more complex protocols should be done in order to appreciate the behaviors and the effectiveness of the different approaches.

[13] Blanchet used a Pentium 233MHz, Hermes was used on a Pentium III 600MHz, Securify was ran on a Pentium III 933MHz + 256Mb of RAM and we used Pentium III 733MHz + 128Mb of RAM

Chapter 6

Conclusion and Future Work

The formal verification of cryptographic protocols is vital in our society due to the proliferation of electronic communications. The problems raised by the verification of protocols are complex [EG83]; potentially an infinite number of exchanges, intruders that can play around the protocol, etc. Nevertheless, several approaches and tools are now available to carry out the task, employing some assumptions (i.e. finite number of session, perfect encryption, etc.). When this work started many automatic tools were available to "search for attacks" against protocols [DY83, Mea94, Low96, DMT98, MMS97]. Now automatic tools to prove properties are becoming available [Bla01, OCKS02, BLP03, Azi03]. Those approaches are essentially based on the proof by abstraction; the idea is to prove a property on an abstract model of the protocol that guarantees the validity of the property on the concrete model. Among these techniques, the association of Timbuk and IS2TiF can be found.

6.1 Work Accomplished

The work is based on a semi-automatic approach introduced by Genet and Klay [GK00a]. Their idea is to compute a tree automaton, $\mathcal{A}_{approximation}$, modeling an over-approximation of the set of messages exchanged. The drawbacks of this approach are:

- the approximation function used must be given by hand and must be chosen

carefully to guarantee the termination of the computation of $\mathcal{A}_{approximation}$;

- when the verification of the properties fails, no conclusion can be drawn about the protocols and its properties;
- the verification is limited to secrecy and authentication properties.

An improved version of Genet and Klay's approach was introduced in this thesis.

A new approximation function was defined, the new features of which are:

- that it is automatically generated;
- that it guarantees the termination of computation of $\mathcal{A}_{approximation}$.

This thesis also presents a concrete way of combining this approximation technique with another technique, for the case where our verification fails. The inductive approach of Paulson was selected because the combination would strengthen both techniques.

The difficulty was not in finding why the completion was not terminating as Genet presented the problem in his thesis [Gen98a]. The problem was in finding an approximation that was precise enough to allow the verification of the secrecy and authentication properties. For example, the ancestor approximation is not able to verify both properties. Moreover the approximation function should also allow the verification of a wild range of protocols in a reasonable time. The results of the validation are limited to the examples covered as they have been manually checked after while. No unit tests and no integration tests have been implemented during the development of the prototype so it is not totally reliable. Nevertheless, on our set of acceptance tests (the protocols that we used) the approximation approach correctly worked and was effective.

The prototype IS2TiF uses the improved version of Genet and Klay's technique to generate an input file for Timbuk. Timbuk then computes $\mathcal{A}_{approximation}$ with the TRS, the initial automaton and the approximation contained in the input file. To

modeling the computation of $\mathcal{A}_{approximation}$, Timbuk uses [CLC03][1] to decide how many agents must be defined in the initial automaton. If the protocol allows agents to talk to themselves, then one honest agent and a set of dishonest agents are in the initial automaton; otherwise two honest agents and a set of dishonest agents.

Depending on the protocol and whether an agent is allowed to talk to himself, the run-times of Timbuk+IS2TiF can be less impressive than the ones of the automatic "attack search" approaches [Low96, Mon99a, GL00, JRV00]. But unlike those approaches, the results are not bounded to a finite number of sessions. The run-times can be also less impressive than those of the other automatic "proof" approaches [Bla01, Cor02, BLP03] however, unlike those approaches, the method can also verify the authentication property. It also offers the possibility to verify protocols that explicitly state that an agent cannot talk to himself, unlike [Bla01, BLP03]. To be able to compare fairly our approach with other existing software (other than the ones presented in the previous chapter); a set of protocols should be verified on the same machine using different tools and the results should be analyzed and compared regarding the intruder abilities, the model constrains and the properties verified.

[BHKO04] offers a comparison between the new model and the initial Genet and Klay model. With the original representation, the protocols ISO611 and Woolham-pi were not automatically checkable; now they are with the new model. The ISO611 protocol uses two nonces, one encrypted and one non-encrypted. Thus when no difference is made between the nonces of the same protocol run (Genet and Klay's model), it is impossible to check the secrecy of the encrypted nonce. With our model that distinguishes the two nonces (ie. *nonce*1 and *nonce*2), if *nonce*2 is the one encrypted it is possible to check its secrecy. In the Woolham-pi protocol, in order to send the third message to Bob, Alice has to remember that she initiated a communication with him, as there is no information about the identities of the sender of the second message. Our model makes sure that

[1] two agents are sufficient for the analysis of security properties of cryptographic protocols when the protocols allow an agent to talk to himself. If the protocol does not allow "agents to talk to them selves" and there is an attack involving n agents, then there is an attack involving at most $k+1$ agents (k is the number of roles that an agent can play)

the variables used in the right hand side of TRS are also in the left hand side, if not, it adds previous messages (sent or received) into the left hand side until the sender has all the information he needs to send the message. For example, the rewrite rule, $mesg(a_6, agt(a), N(a_1, b_1, t_1)) \Longrightarrow U(LHS, mesg(agt(a), agt(b), encr(sharekey(agt(a), serv(S), t_0), agt(a), N(a_1, b_1, t_1))))$, that models the third step of the Woolham-pi protocol is not valid, as the variable b does not appear in the left-hand side. So the TRS is cannot be run. Our approach produces

$U(mesg(agt(a), agt(b), agt(a)),\ mesg(a_6, agt(a), N(a_1, b_1, t_1))) \Longrightarrow$

$U(LHS,\ mesg(agt(a),\ agt(b), encr(sharekey(agt(a), serv(S), t_0), agt(a), N(a_1, b_1, t_1))))$,

which is a valid rewrite rule.

Nevertheless, the approach has some drawbacks:

- computation time is linked to the complexity of the TRS, to "allowing an agent to talk to himself" and to whether the protocol is safe or not. The success of the computation depends on the computer resources (essentially memory) for complex protocols.
- because of the typing used in the term rewriting system, the results of the verification are valid, assuming that protocols are free from type attacks.
- for the verification to be valid, protocols must satisfy the conditions of Proposition 9. This is usually the case, as the variables that will have several occurrences in the TRS will be substituted by the states linked to the agents. These states are deterministic.
- for the combining approach, there is a compatibility problem in practice between the two tools, one is running on SML and the other on OCAML, so it requires the development of a user interface on top of both tools to simplify the use of this approach.

6.2 Future Work

Starting from the current stage of this research three subprojects could be carried out:

The first subproject is the logical continuation of this MSc work. The current cryptographic protocol verification approaches make an initial assumption that the encryption is perfect; that means that the intruder cannot get encrypted information if he does not have the right decryption key (our approach makes no exception). This is a very strong assumption, that could lead some flaws being masked. For example, under this assumption, Paulson proved that the recursive authentication protocol of John Bull was safe, but by considering the algebraic properties of the XOR, attacks can be found [Pau97]. In reality, the intruder could take advantage of some algebraic properties of the encryption (XOR, abelian groups) in his quest for critical information. For example, we have a nonce, Na, encrypted with a public key, Ka, using the XOR. $(Na\ XOR\ Ka) = \{Na\}_{Ka}$ then the XOR properties states that $(Na\ XOR\ \{Na\}_{Ka}) = Ka$ and $(Ka\ XOR\ \{Na\}_{Ka}) = Na$. A rule for the decryption of the information with the correct decryption key, the $(Ka\ XOR\ \{Na\}_{Ka}) = Na$, is already in the TRS. Moreover allowing the intruder to exploit the XOR properties requires in the TRS to also include a rule stating that from the nonce and the encrypted message the key can be recovered, the $(Na\ XOR\ \{Na\}_{Ka}) = Ka$. Thus the integration of those new abilities and their consequences in the verification should be explored. The starting point would be to add into IS2TiF an option such that the user selects algebraic properties that the user can use. Then extra rewriting rules criticize the properties are added into the input file for Timbuk.

Also, to improve the approach, one can develop a tool that gives as many guarantees as possible on the protocols verified and to facilitate the choice of users between the verification tools available. Another part of this subproject is to see if the technique could be extended to verify other properties that might be required by particular protocol; for example freshness, anonymity, equity properties. The primary task here is to see how those properties could be modeled with a tree automaton and if so, then to have a library of automaton that could be used to verify them and have a procedure to automatically build automata for those properties. If we take the freshness property, we will need to distinguish different runs to check if a piece of information is fresh. Thus an idea is to attach a marker to the fresh

information; for example $maker(1, N(A, B))$ indicates that $N(A, B)$ was created for the run 1 of the protocol. IS2TiF must be updated to propose initial automata and term rewriting systems dealing with this new term. The computation of the approximation for all the possible runs of a particular protocol is then impossible as the number of markers will infinitely increase. Thus second idea, reducing the number of marker to two (to only have $marker(1, ...)$ and $marker(2, ...)$ in our model) and proving that is sufficient to verify freshness properties.

The second subproject involves the use of the main result of this research, the approximation function, for the invariant generation and verification. An invariant is a clause/condition that is satisfied by all the reachable states of a system.

Properties such as safety properties, which must be guaranteed at any stage of runs, can be seen as invariants. Finding and strengthening invariants is crucial for the analysis and verification of re-active systems, especially for infinite state systems. The invariant search and verification is a very active field of research [BL03, BS00, TRSS01]. As for protocol verification, the current techniques to find and verify invariants might fail due to the size of the system studied.

With the approximation function defined in this thesis, the computation of an over approximation of the reachable states of the system can be done; then either an invariant can be checked on the last automaton or an invariant can be deduced from the final automaton. In fact on simple examples, the new approximation function appears to be efficient in that domain. However a tool must be developed that automatically returns the invariant of a system (on a simple example Appendix F, the abstract model computed had to be analyzed manually to find the invariant). A new generator of input files for Timbuk should be implemented as IS2TiF is taking in input an ISABELLE file that models a protocol. In order to be able to deal with a large range of systems, the prototype will be taking specifications in Z or B language to produce the term rewriting system, the initial automaton and the approximation. Then Timbuk will be used to compute an over-approximation of all the configuration of the system. The prototype will either take an invariant and check if the system verifies it or return a possible invariant. The verification of the

invariant is easier as it only requires to check the intersection of the approximation automaton with the automaton modeling the negation of the invariant. Returning the invariant will be more difficult, it will require algorithms (depth search or best first search or ...) to be implemented in the prototype to explore the transitions of the final automaton in order to extract an invariant. The invariant will be an *over invariant* as it will be valid for the configurations of the concrete system but also for the configurations that are not in the concrete system.

Once the prototype is available, tests on real systems and comparisons of the results given by the prototype with the ones on other tools should be carried out.

The last subproject involves the use of another model to reason about cryptographic protocols: the Strand Spaces. A strand is a sequence of messages sent and received by an agent. A strand space is a set of strands (agents' strands plus intruder ones). A bundle consists of a number of strands hooked together where one strand sends a message and another one receives that same message. A protocol will be correct when each bundle consists of one strand for each agent all agreeing on the participants, nonces (random numbers), and session keys. Intruder strands are also included in a bundle but as long as they do not prevent honest agents agreeing on a secret or keeping their secrets. An interesting aspect of this model is that it allows reasoning for an unbounded number of sessions.

[CDL+00] explains how to pass from a multiset rewriting model to a strand space model and vice versa. This paper concludes by saying that algorithms developed for rewriting models could be brought to deal with strands.

Therefore an attempt to adapt the technique presented in this thesis to strand space models could be possible. The first part will be to translate the rewriting model used here into strands using [CDL+00] and then modify the completion algorithm to get a completion algorithm for Strand Spaces. After that the new results should be compared with our current ones, and also with the ones of the CASRUL+IS2TiF (ongoing work at Laboratoire Informatique de Franche-Comté) to see if this idea is efficient or not.

Bibliography

[Aba99] Martín Abadi. Secrecy by Typing in Security Protocols. In *Journal of the ACM*, volume 46(5), pages 749–786, 1999.

[Aba00] M. Abadi. Security Protocols and their Properties. In *20th Int. Summer School, Foundations of Secure Computation*, pages 39–60, 2000.

[Aba02] M. Abadi. Private Authentication. In *Proceedings of Workshop on Privacy Enhancing Technologies*, pages 27–40, 2002.

[ABB+02] A. Armando, D. Basin, M. Bouallagui, Y. Chevalier, L. Compagna, S. Moedersheim, M. Rusinowitch, M. Turuani, L. Viganò, and L. Vigneron. The AVISS Security Protocol Analysis Tool. In *Proceedings of the 14th Conference on Computer-Aided Verification (CAV'02)*, Lecture Notes in Computer Science, pages 349–353. Springer-Verlag, 2002.

[AG98] M. Abadi and A. D. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. Technical report, DIGITAL, Systems Research Center, N 149, January 1998. http://www.research.digital.com/SRC/publications/.

[AKN89a] H. Ait-Kaci and M. Nivat. *Resolution of Equations in Algebraic Structures (Volume I): Algebraic Techniques.* Academic Press, 1989.

[AKN89b] H. Ait-Kaci and M. Nivat. *Resolution of Equations in Algebraic Structures (Volume II): Rewriting Techniques.* Academic Press, 1989.

[AN95] M. Abadi and R. Needham. Prudent Engineering Practice for Cryptographic Protocols. Technical report, Digital Systems Research Center, N 125, November 1995.

[AN96] M. Abadi and R. Needham. Prudent Engineering Practice for Cryptographic Protocols. In *IEEE Transactions on Software Engineering*, volume 22(1), pages 6–15, 1996.

[AT91] M. Abadi and M. Tuttle. A Semantics for a Logic of Authentication. In *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing*, pages 201–216, 1991.

[Azi03] B. Aziz. *A Static Analysis Framework for Security Properties of Mobile Systems and Cryptographic Protocols.* PhD thesis, Dublin City University. School of Computing, 2003.

[BAN89] M. Burrows, M. Abadi, and R. Needham. A Logic of Authentication. Technical report, Digital Systems Research Center, N 39, February 1989. http://www.research.digital.com/SRC/publications/.

[BBC+97] B. Barras, S. Boutin, C. Cornes, J. Courant, J.-C. Filliatre, E. Gimenez, H. Herbelin, G. Huet, C. Munoz, C. Murthy, C. Parent, C. Paulin-Mohring, A. Saibi, and B. Werner. The Coq Proof Assistant Reference Manual : Version 6.1, 1997. RT-0203.

[BC01] M. Benerecetti and A. Cimatti. Symbolic Model Checking for Multi-Agent Systems. In *Proceedings of Workshop on Computational Logic in Multi-Agent Systems, CLIMA-2001*, pages 312–323, 2001.

[BCF02] C. Braghin, A. Cortesi, and R. Focardi. Freshness Analysis in Security Protocols. In *Proceedings of 14th Nordic Workshop on Programming Theory (NWPT'02)*, pages 30–33, 2002.

[BDD+05] M. Backes, A. Datta, A. Derek, J. C. Mitchell, and M. Turuani. Compositional analysis of contract signing protocols. In *CSFW '05: Pro-*

ceedings of the 18th IEEE Computer Security Foundations Workshop (CSFW'05), pages 94–110, 2005.

[BG00] M. Benerecetti and F. Giunchiglia. Model Checking Security Protocols Using a Logic of Belief. In *TACAS 2000*, pages 519–534, 2000.

[BHKO04] Y. Boichut, P.-C. Héam, O. Kouchnarenko, and F. Oehl. Improvements on the Genet and Klay Technique to Automatically Verify Security Protocols. In *3rd International Workshop on Automated Verification of Infinite-State Systems (AVIS'04)*, pages 1–11, April 2004.

[BL03] S. Bensalem and Y. Lakhnech. Automatic Generation of Invariants. In *Formal Methods in System Design*, pages 75–92, 2003.

[Bla01] B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *Proceedings of the 14th Computer Security Foundations Workshop*, pages 82–96. IEEE Computer Society Press, 2001.

[BLP02] L. Bozga, Y. Lakhnech, and M. Périn. L'Outil de Vérification HERMES. Technical report, Verimag, 2002.

[BLP03] L. Bozga, Y. Lakhnech, and M. Périn. Pattern-Based Abstraction for Verifying Secrecy in Protocols. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, pages 299–314, 2003.

[BLR00] P. J. Broadfoot, G. Lowe, and A. W. Roscoe. Automating Data Independence. In *The Proceedings of 6th European Symposium on Research in Computer Security*, Lecture Notes in Computer Science, pages 175–190. Springer-Verlag, 2000.

[BMP02] G. Bella, F. Massacci, and L. C. Paulson. The Verification of an Industrial Payment Protocol: The SET Purchase Phase. In *9th ACM Conference on Computer and Communications Security (ACM Press)*, pages 12–20, 2002.

[BMP03] G. Bella, F. Massacci, and L. C. Paulson. Verifying the SET Registration Protocols. In *IEEE Journal on Selected Areas in Communications, volume 21(1)*, pages 77–87, 2003.

[BMPT00] G. Bella, F. Massacci, L. C. Paulson, and P. Tramontano. Formal Verification of Cardholder Registration in SET. In *ESORICS*, pages 159–174, 2000.

[Bol96] D. Bolignano. An Approach to the Formal Verification of Cryptographic Protocols. In *ACM Conference on Computer and Communications Security*, pages 106–118, 1996.

[Bol97] D. Bolignano. Towards the Formal Verification of Electronic Commerce Protocols. In *Proceeding of the 1997 IEEE Computer Security Foundations Workshop X*, pages 133–146. IEEE Computer Society Press, 1997.

[BP97] G. Bella and L. C. Paulson. Using Isabelle to Prove Properties of the Kerberos Authentication System. In *Workshop on Design and Formal Verification of Security Protocols. DIMACS*, 1997.

[BP98a] G. Bella and L. C. Paulson. Kerberos Version IV: Inductive Analysis of the Secrecy Goals. In *Proceedings of the 5th European Symposium on Research in Computer Security*, pages 361–375. Springer-Verlag LNCS 1485, 1998.

[BP98b] G. Bella and L. C. Paulson. Mechanising BAN Kerberos by the Inductive Method. In *Computer Aided Verification*, pages 416–427, 1998.

[BPST02] M. Benerecetti, M. Panti, L. Spalazzi, and S. Tacconi. Verification of the SSL/TLS Protocol Using a Model Checkable Logic of Belief and Time. In *Proceedings of Computer Safety, Reliability and Security, 21st International Conference, SAFECOMP 2002*, volume 2434 of *Lecture Notes in Computer Science*. Springer, 2002.

[Bra96] S. H. Brackin. A HOL Extension of GNY for Automatically Analysing Cryptographic Protocols. In *Proceedings of the 1996 IEEE Computer Security Foundations Workshop IX*, pages 62–76. IEEE Computer Society Press, 1996.

[BS00] R. Bharadwaj and S. Sims. Salsa: Combining Constraint Solvers with BBDs for Automatic Invariant Checking. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'00)*, pages 378–394, 2000.

[CC92] P. Cousot and R. Cousot. Abstract Interpretation and Application to Logic Programs. In *Journal of Logic Programming*, volume 13(2-3), pages 103–179, 1992.

[CDG+98] H. Comon, M. Dauchet, R. Gilleron, D. Lugiez, S. Tison, and M. Tommasi. Tree Automata Techniques and Applications, 1998.

[CDL+99] I. Cervesato, N. A. Durgin, P. Lincoln, J. C. Mitchell, and A. Scedrov. A Meta-Notation for Protocol Analysis. In *Proc. 12th IEEE Computer. Security Foundations Workshop (CSFW'99)*, pages 55–69, 1999.

[CDL+00] I. Cervesato, N. A. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. Relating Strands and Multiset Rewriting for Security Protocol Analysis. In *Proceedings of the 13th Computer Security Foundations Workshop*, pages 35–51. IEEE Computer Society Press, 2000.

[CHSV03] B. Canvel, A. Hiltgen, S.Vaudenay, and M. Vuagnoux. Password Interception in a SSL/TLS Channel. In *CRYPTO 2003*, number 2729 in Lecture Notes in Computer Science, pages 583–599. Springer-Verlag, 2003.

[CJ97] J. Clark and J. Jacob. A Survey of Authentication Protocol literature: Version 1.0., 1997.

[CJM00] E. M. Clarke, S. Jha, and W. Marrero. Verifying Security Protocols with Brutus. In *ACM Transactions on Software Engineering and Methodology*, volume 9(4), pages 443–487, 2000.

[CLC03] H. Comon-Lundh and V. Cortier. Security Properties: Two Agents Are Sufficient. In *Proc. 12th European Symposium on Programming (ESOP'2003)*, Lecture Notes in Computer Science, pages 99–113. Springer-Verlag, 2003.

[CLT03] H. Comon-Lundh and R. Treinen. Easy Intruder Deductions. In *Verification : Theory and Practice, Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday, LNCS Volume 2772*, pages 225–242, 2003.

[CMR01] V. Cortier, J. Millen, and H. Ruess. Proving Secrecy is Easy Enough. In *Proc. 14th IEEE Computer. Security Foundations Workshop (CSFW'01)*, pages 97–108, 2001.

[Cor02] V. Cortier. Outil de Vérification SECURIFY. Technical report, Ecole Normale Supérieure de Cachan, 2002.

[Cou01] P. Cousot. Abstract Interpretation Based Formal Methods and Future Challenges, invited paper. In Wilhelm, R., editor, *Informatics — 10 Years Back, 10 Years Ahead*, volume 2000 of *Lecture Notes in Computer Science*, pages 138–156. Springer-Verlag, 2001.

[CRZ05] V. Cortier, M. Rusinowitch, and E. Zalinescu. A resolution strategy for verifying cryptographic protocols with CBC encryption and blind signatures. In *Proceedings of the 7th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming*, pages 12–22, 2005.

[CS96] D. Craigen and M. Saaltink. Using EVES to Analyze Authentication Protocols. Technical report, Technical Report TR-96-5508-05, ORA Canada, March 1996. http://www.ora.on.ca/eves/documentation.html.

[DDHY92] D. L. Dill, A. J. Drexler, A. J. Hu, and C. Han Yang. Protocol Verification as a Hardware Design Aid. In *International Conference on Computer Design*, pages 522–525, 1992.

[Der82] N. Dershowitz. Orderings for Term-Rewriting Systems, Theoretical Computer Science, 17:279–301, 1982.

[Die97] S. Dietrich. *A Formal Analysis of the Secure Sockets Layer Protocol.* PhD thesis, Adephi University, Garden City, New York. Department of Mathematics and Computer Science, 1997.

[DJ90] N. Dershowitz and J.P. Jouannaud. Chapter 6 of Handbook of Theoretical Computer Science (Volume B): Formal Methods and Semantics, 1990.

[DM00] G. Denker and J. Millen. CAPSL Intermediate Language. In *DARPA Information Survivability Conference (DISCEX 2000), IEEE Computer Society*, pages 207–221, 2000.

[DMT98] G. Denker, J. Meseguer, and C. Talcott. Protocol Specification and Analysis in Maude. In *Proc. of Workshop on Formal Methods and Security Protocols*, 1998.

[DMTY97] M. Debbabi, M. Mejri, N. Tawbi, and I. Yahmadi. Formal Automatic Verification of Authentication Cryptographic Protocols. In *Proceedings of 1st International Conference on Formal Engineering Methods (ICFEM '97).* IEEE Computer Society Press, 1997.

[DNL99] B. Donovan, P. Norris, and G. Lowe. Analyzing a library of Security Protocols using Casper and FDR. In *Proceedings of the Workshop on Formal Methods and Security Protocols*, 1999.

[DP01] N. Dershowitz and D. A. Plaisted. *Chap. 9 in: Handbook of Automated Reasoning*, volume 1. Elsevier and MIT Press, 2001.

[DY83] D. Dolev and A. Yao. On the Security of Public-Key Protocols. In *IEEE Transactions on Information Theory, 2(29)*, pages 198–208, 1983.

[EG83] S. Even and O. Goldreich. On the Security of Multi-Party Ping-Pong Protocols. In *IEEE Symposium on Foundations of Computer Science*, pages 34–39, 1983.

[FA01] M. Fiore and M. Abadi. Computing Symbolic Models for Verifying Cryptographic Protocols. In *Proceedings of 14th IEEE Computer Security Foundations Workshop (CSFW'01)*, pages 160–173. IEEE Computer Society Press, 2001.

[Gen98a] T. Genet. *Contraintes dOrdre et Automates dArbres pour les Preuves de Terminaison.* PhD thesis, INRIA, 1998.

[Gen98b] T. Genet. Decidable Approximations of Sets of Descendants and Sets of Normal Forms. In *Rewriting Techniques and Applications: 9th International Conference, RTA-98*, Lecture Notes in Computer Science, pages 151–165. Springer-Verlag, 1998.

[GFT03] T. Genet G. Feuillade and V. Viet Triem Tong. Reachability Analysis of Term Rewriting Systems. Technical Report 4970, I.N.R.I.A., October 2003.

[GHvRP05] F. D. Garcia, I. Hasuo, P. van Rossum, and W. Pieters. Provable anonymity. In Ralf Küsters and John Mitchell, editors, *Proceedings of the 2005 ACM Workshop on Formal Methods in Security Engineering (FMSE '05)*, pages 63–72. ACM, 2005.

[GK00a] T. Genet and F. Klay. Rewriting for Cryptographic Protocol Verification. In *CADE: International Conference on Automated Deduction*, pages 271–290, 2000. http://citeseer.nj.nec.com/genet99rewriting.html.

[GK00b] T. Genet and F. Klay. Rewriting for Cryptographic Protocol Verification. Technical Report 3921, I.N.R.I.A., Avril 2000.

[GL00] J. Goubault-Larrecq. A Method for Automatic Cryptographic Protocol Verification (extended abstract). In *Proc. Workshop on Formal Methods in Parallel Programming, Theory and*

Applications (FMPPTA'2000), number 1800 in Lecture Notes in Computer Science, pages 977–984. Springer-Verlag, 2000. http://www.dyade.fr/fr/actions/vip/publications.html.

[GLL00] S. Gnesi, D. Latella, and G. Lenzini. A BRUTUS Logic for the Spi-Calculus Dialect. In *Proceedings of the Workshop on Formal Methods and Computer Security (FMCS 2000)*, 2000.

[GNY90] L. Gong, R. Needham, and R. Yahalom. Reasoning About Belief in Cryptographic Protocols. In Deborah Cooper and Teresa Lunt, editors, *Proceedings 1990 IEEE Symposium on Research in Security and Privacy*, pages 234–248. IEEE Computer Society Press, 1990.

[Gon93] L. Gong. Variations on the Themes of Message Freshness and Replay. In *Proceedings of the IEEE Computer Security Foundations Workshop VI*, pages 131–136, 1993.

[Gro96a] SET Working Group. *SET^{TM} Specification, books 1,2 and 3.* SETCO, 1996. http://www.setco.org/set_specifications.html.

[Gro96b] TLS Working Group. *The TLS Protocol Version 1.0.* The Internet Engineering Task Force, 1996. http://www.ietf.org/html.charters/tls-charter.html.

[GRS99] D. Goldschlag, M. Reed, and P. Syverson. Onion Routing for Anonymous and Private Internet Connections. In *Communications of the ACM (USA)*, volume 42(2), pages 39–41, 1999.

[GS84] F. Gecseg and M. Steinby. *Tree Automata.* Budapest : Akademiai Kiado, 1984.

[GT95] R. Gilleron and S. Tison. Regular Tree Languages and Rewrite Systems. In *Fundamenta Informaticae*, volume 24, pages 157–175, 1995.

[GT01] T. Genet and V. Viet Triem Tong. Reachability Analysis of Term Rewriting Systems with Timbuk. In *Proc. Logic for Programming, Ar-*

tificial Intelligence and Reasoning (LPAR 2001), LNAI, pages 695–706. Springer-Verlag, 2001.

[HL78] G. Huet and D. S. Lankford. On the Uniform Halting Problem for Term Rewriting Systems. Technical report, INRIA, 1978.

[HLS03] J. Heather, G. Lowe, and S. Schneider. How to Prevent type Flaw Attacks on Security Protocols. In *Journal of Computer Security*, volume 11(2), pages 217–244, 2003.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[Höl89] S. Hölldobler. *Foundations of Equational Logic Programming.* Springer-Verlag, 1989.

[Hul80] J. M. Hullot. Canonical Forms and Unification. In *Proceedings of 5th International Conference on Automated Deduction*, Lecture Notes in Computer Science, pages 318–334. Springer-Verlag, 1980.

[Jac96] F. Jacquemard. Decidable Approximations of Term Rewriting Systems. In *Proceedings of 7th Conference on Rewriting Techniques and Applications*, number 1103 in Lecture Notes in Computer Science, pages 362–376. Springer-Verlag, 1996.

[JRV00] F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and Verifying Security Protocols. In *Logic Programming and Automated Reasoning*, pages 131–160, 2000.

[Kai95] R. Kailar. Reasoning about Accountability in Protocols for Electronic Commerce. In *Proceeding of the 1995 IEEE Symposium on Security and Privacy*, pages 236–250. IEEE Compter Society Press, 1995.

[Kai96] R. Kailar. Accountability in Electronic Commerce Protocols. In *IEEE Transaction on Software Engineering*, volume 22(5), pages 313–328, 1996.

[KB70] D. E. Knuth and P. B. Bendix. Simple Word Problem in Universal Algebra. In *Computational Problems in Abstract Algebras*, pages 263–297. Pergamon Press, Oxford, 1970.

[KFK96] P. Kocker, A. Freier, and P. Karlton. The SSL Protocol Version 3.0, 1996. http://wp.netscape.com/eng/ssl3/index.html.

[KG94] V. Kessler and G.Wedel. AUTLOG-An advanced Logic of Authentication. In *Proceedings of the 1994 IEEE Computer Security Foundations Workshop VII*, pages 90–99. IEEE Compter Society Press, 1994.

[Kin99] D. Kindred. *Theory Generation for Security Protocols.* PhD thesis, Technical Report CMU-CS-99-130, Computer Science Department, Carnegie Mellon University, Pittsburg, PA, 1999.

[KN98] V. Kessler and H. Neumann. A Sound Logic for Analysing Electronic Commerce Protocols. In *Proceedings of the 5th European Symposium on Research in Computer Security*, pages 345–360. Springer Verlag, 1998.

[KR02] S. Kremer and J.-F. Raskin. Game Analysis of Abuse-free Contract Signing. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop, Cape Breton*, pages 206–222, 2002.

[Lan79] D. S. Lankford. On Proving Term Rewriting Systems are Noetherian. Technical report, Louisiana Tech. Univ., 1979.

[LDG+01] X. Leroy, D. Doligez, J. Garrigue, D. Rémy, and J. Vouillon. The Objective Caml system release 3.02. In *Documentation and user's manual*, 2001.

[Low95] G. Lowe. An Attack on the Needham-Schroeder Public-Key Authentication Protocol. In *Information Processing Letters*, volume 56(3), pages 131–133, 1995.

[Low96] G. Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. In *Proceedings of TACAS, LNCS 1055*, pages 147–166. Springer-Verlag, 1996.

[Low97a] G. Lowe. A Hierarchy of Authentication Specifications. In *PCSFW: Proceedings of the 10th Computer Security Foundations Workshop*, pages 31–44. IEEE Computer Society Press, 1997.

[Low97b] G. Lowe. Casper: A Compiler for the Analysis of Security Protocols. In *PCSFW: Proceedings of The 10th Computer Security Foundations Workshop*, pages 18–30. IEEE Computer Society Press, 1997.

[Low99] G. Lowe. Towards a Completeness Result for Model Checking of Security Protocols. In *Journal of Computer Security*, volume 7(1), pages 89–146, 1999.

[MCJ97] W. Marrero, E. Clarke, and S. Jha. A Model Checker for Authentication Protocols. In *Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

[Mea92] C. Meadows. Applying Formal Methods to the Analysis of a Key Management Protocol. In *Journal of Computer Security*, volume 1(1), pages 5–53, 1992.

[Mea94] C. Meadows. A Model of Computation for the NRL Protocol Analyzer. In *Proceedings of 7th Computer Security Foundations Workshop*, pages 84–89, 1994.

[Mea96] C. Meadows. The NRL Protocol Analyser: An Overview. In *Journal of Logic Programming, 26(2):113–131*, February 1996.

[Mea00] C. Meadows. Extending formal cryptographic protocol analysis techniques for group protocols and low-level cryptographic primitives. In *First Workshop on Issues in the Theory of Security*, pages 87–92, 2000.

[MGK02] O. Markowitch, D. Gollmann, and S. Kremer. On Fairness in Exchange Protocols. In *Proceedings 5th Int. Conf. on Information Security and Cryptology (ISISC 2002)*, pages 451–464, 2002.

[MMS97] J. C. Mitchell, M. Mitchell, and U. Stern. Automated Analysis of Cryptographic Protocols Using Murφ. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy* , pages 141–153. IEEE Computer Society Press, 1997.

[MN70] Z. Manna and S. Ness. On the Termination of Markov Algorithms. In *Proceedings 3rd International Conference System Science*, pages 789–792, 1970.

[Mon99a] D. Monniaux. Abstracting Cryptographic Protocols with Tree Automata. In *Static Analysis Symposium*, Lecture Notes in Computer Science, pages 149–163. Springer-Verlag, 1999. http://citeseer.nj.nec.com/monniaux99abstracting.html.

[Mon99b] D. Monniaux. Decision Procedures for the Analysis of Cryptographic Protocols by Logics of Belief. In *PCSFW: Proceedings of the 12th Computer Security Foundations Workshop*, pages 44–54. IEEE Computer Society Press, 1999.

[MSS98] J.C. Mitchell, V. Shmatikov, and U. Stern. Finite-state analysis of SSL 3.0. In *7th USENIX Security Symposium*, pages 201– 216, 1998.

[Nes90] D. M. Nessett. A Critique of the BAN Logic. In *ACM Operating System Review*, pages 35–38, 1990.

[New42] M. H. A. Newman. On Theories with a Combinatorial Definition of Equivalence. In *Annals of Mathematics*, volume 43(2), pages 223–243, 1942.

[NRV04] M. Nesi, G. Rucci, and M. Verdesca. On Rewriting Protocol Specifications. In *Proceedings of International Workshop on Security Analysis of Systems: Formalisms and Tools*, 2004.

[NS78] R. Needham and M. Schroeder. Using Encryption for Authentication in Large Networks of Computers. In *Communications of the ACM*, volume 21(12), pages 993–999, 1978.

[OCKS02] F. Oehl, G. Cécé, O. Kouchnarenko, and D. Sinclair. Automatic Approximation for the Verification of Cryptographic Protocols. Technical report, Dublin City University and I.N.R.I.A., October 2002.

[Pau94] L. C. Paulson. *Isabelle: a Generic Theorem Prover.* Springer-Verlag Inc., LNCS 828, 1994.

[Pau97] L. Paulson. Mechanized Proofs for a Recursive Authentication Protocol. In *10th IEEE Computer Security Foundations Workshop (1997)*, pages 84–95, 1997.

[Pau98] L. C. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. In *Journal of Computer Security*, volume 6, pages 85–128, 1998. http://www.cl.cam.ac.uk/users/lcp/papers/protocols.html.

[Pau99] L. C. Paulson. Inductive Analysis of the Internet Protocol TLS. In *ACM Transactions on Information and System Security*, volume 2(3), pages 332–351, 1999.

[Pau01] L. C. Paulson. SET Cardholder Registration: the Secrecy Proofs. In *International Joint Conference on Automated (IJCAR 2001)*, pages 5–12, 2001.

[PSW+01] A. Perrig, R. Szewczyk, V. Wen, D. E. Culler, and J. D. Tygar. SPINS: Security Protocols for Sensor Networks. In *Mobile Computing and Networking*, pages 189–199, 2001.

[Ros95] A. W. Roscoe. Modelling and Verifying Key-Exchange Protocols Using CSP and FDR. In *8th Computer Security Foundations Workshop*, pages 98–107. IEEE Press, 1995.

[RR98] M. K. Reiter and A. D. Rubin. Crowds: Anonymity for Web Transactions. In *ACM Transactions on Information and System Security*, volume 1(1), pages 66–92, 1998.

[RSA78] R. L. Rivest, A. Shamir, and L. M. Adelman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. In *Communications of the ACM*, volume 21(2), pages 120–126, 1978.

[RV98] D. Rémy and J. Vouillon. Objective ML: An Effective Object-Oriented Extension to ML. In *Theory And Practice of Objects Systems, 4(1):27–50*, 1998.

[Sat89] M. Satyanarayanan. Integrating Security in Large Distributed System. In *ACM Transactions on Computer Systems*, volume 7(3), pages 247–280, 1989.

[SBP01] D. Song, S. Berezin, and A. Perrig. Athena: A Novel Approach to Efficient Automatic Security Protocol Analysis. In *Journal of Computer Security*, volume 9(1/2), pages 47–74, 2001.

[Sch96] J. Schumann. Automatic Verification of Cryptographic Protocols Using SETHEO. Technical report, Technical Report AR-96-03, TU München, Institut für Informatik, 1996. http://wwwjessen.informatik.tu-muenchen.de/~schumann/crypt.html.

[Sch97] S. Schneider. Verifying Authentication Protocols with CSP. In *PCSFW: Proceedings of The 10th Computer Security Foundations Workshop*, pages 3–17. IEEE Computer Society Press, 1997.

[Sch98] S. Schneider. Verifying Authentication Protocols in CSP. In *IEEE Trans. Softw. Eng.*, volume 24(9), pages 741–758. IEEE Press, 1998.

[Sci02] Sciences & Avenir, N.659, Janvier 2002.

[Sue00] Suetonius. *Lives of the Twelve Caesars.* Publisher Penguin, 2000.

[SvO94] P. Syverson and P. C. van Oorschot. On Unifying some Cryptographic Protocol Logics. In *Proceedings of the 1994 IEEE Computer Security Foundations Workshop VII*, pages 14–29. IEEE Compter Society Press, 1994.

[Syv94] P. Syverson. A taxonomy of replay attacks. In *Computer Security Foundations Workshop VII*, pages 211–254. IEEE Computer Society Press, 1994.

[THG99] J. Thayer, J. Herzog, and J. Guttman. Strand Spaces: Proving Security Protocols Correct. In *Journal of Computer Security 7*, pages 191–230, 1999.

[TRSS01] A. Tiwari, H. Rueß, H. Saïdi, and N. Shankar. A Technique for Invariant Generation. In *Lecture Notes in Computer Science*, volume 2031, pages 113–127, 2001.

[Tur36] A. Turing. On Computable Numbers, With an Application to the Entscheidungsproblem. In *Proceedings of the London Mathematical Society*, volume 42(2), pages 230–265, 1936.

[TVV05] M. Turuani and H. Vu-Van. Validation of the asw contract signing protocol. In *Proceedings of APPSEM II Workshop (APPSEM05)*, 2005.

[Vau02] S. Vaudenay. Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS... In *Advances in Cryptology EUROCRYPT'02*, number 2332 in Lecture Notes in Computer Science, pages 534–545. Springer-Verlag, 2002.

[Vig95] L. Vigneron. Positive Deduction modulo Regular Theories. In *Proceedings of the Annual Conference of the European Association for Computer Science Logic*, Lecture Notes in Computer Science, pages 468–485. Springer-Verlag, 1995.

[vO93] P. van Oorschot. Extending Cryptographic Logics of Belief to Key Agreement Protocols. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 232–243, 1993.

[WL94] T. Y. C. Woo and S. S. Lam. A Lesson on Authentication Protocol Design. In *Operating Systems Review*, volume 28(3), pages 24–37, 1994.

[WS96] D. Wagner and B. Schneier. Analysis of the SSL 3.0 Protocol. In *The Second USENIX Workshop on Electronic Commerce Proceedings*, pages 29–40, 1996.

[Zan97] H. Zantema. Termination of Term Rewriting. Technical report, University of Utrecht, 1997.

[Zim99] P. Zimmermann. http://www.loria.fr/ zimmerma/records/rsa155.html, 1999.

Appendix A

Example of completion with the Knuth-Bendix algorithm

To demonstrate how the completion algorithm [KB70] works, let E be our set of identities and $\mathcal{R}$ the TRS that is built. E=$\left\{ \begin{array}{l} e.x = x \\ I(x).x = e \\ (x.y).z = x(y.z) \end{array} \right\}$

The completion algorithm:

- orientates the equation to produce $\mathcal{R}$=$\left\{ \begin{array}{ll} 1. & e.x \rightarrow x \\ 2. & I(x).x \rightarrow e \\ 3. & (x.y).z \rightarrow x(y.z) \end{array} \right\}$ using a "sensible" reduction ordering relation based > on the format of the terms.
- searches a critical pair and finds $(e.z,\ I(x).(x.z))$ with rules 2 and 3.

  e.z can be reduced to z, so we have now the pair $(z,\ I(x).(x.z))$. Using >, we add to $\mathcal{R}$ the rule $I(x).(x.z) \rightarrow z$,

  so $\mathcal{R}$=$\left\{ \begin{array}{ll} 1. & e.x \rightarrow x \\ 2. & I(x).x \rightarrow e \\ 3. & (x.y).z \rightarrow x(y.z) \\ 4. & I(x).(x.z) \rightarrow z \end{array} \right\}$
- searches a critical pair and finds $(y,\ I(I(y)).e)$ with rules 2 and 4.

$$\text{So } \mathcal{R}=\left\{\begin{array}{ll} 1. & e.x \to x \\ 2. & I(x).x \to e \\ 3. & (x.y).z \to x(y.z) \\ 4. & I(x).(x.z) \to z \\ 5. & I(I(y)).e \to y \end{array}\right\}$$

- searches a critical pair and finds $(z,\ I(e).z)$ with rules 1 and 4.

$$\text{So } \mathcal{R}=\left\{\begin{array}{ll} 1. & e.x \to x \\ 2. & I(x).x \to e \\ 3. & (x.y).z \to x(y.z) \\ 4. & I(x).(x.z) \to z \\ 5. & I(I(y)).e \to y \\ 6. & I(e).z \to z \end{array}\right\}$$

- searches a critical pair and finds $(I(I(y)).(e.x),\ y.x)$ with rules 3 and 5.

$$\text{So } \mathcal{R}=\left\{\begin{array}{ll} 1. & e.x \to x \\ 2. & I(x).x \to e \\ 3. & (x.y).z \to x(y.z) \\ 4. & I(x).(x.z) \to z \\ 5. & I(I(y)).e \to y \\ 6. & I(e).z \to z \\ 7. & I(I(y)).(e.x) \to y.x \end{array}\right\}$$

- searches a critical pair and finds $(y.e,\ y)$ with rules 7 and 5.

$$\text{So } \mathcal{R}=\left\{\begin{array}{ll} 1. & e.x \to x \\ 2. & I(x).x \to e \\ 3. & (x.y).z \to x(y.z) \\ 4. & I(x).(x.z) \to z \\ 5. & I(I(y)).e \to y \\ 6. & I(e).z \to z \\ 7. & I(I(y)).(e.x) \to y.x \\ 8. & y.e \to y \end{array}\right\}$$

- searches a critical pair and finds $(y,\ I(I(y)))$ with rules 8 and 5. It gives the rule $I(I(y)) \to y$ that makes rule 5 redundant as $I(I(y)).e \to^*_{\mathcal{R}} y$ by applying firstly the new rule, and then rule 8; thus rule 5 is removed from the TRS. Rule 7 can be removed for the same reason.

$$\text{So } \mathcal{R}=\left\{\begin{array}{ll} 1. & e.x \to x \\ 2. & I(x).x \to e \\ 3. & (x.y).z \to x(y.z) \\ 4. & I(x).(x.z) \to z \\ 6. & I(e).z \to z \\ 8. & y.e \to y \\ 9. & I(I(y)) \to y \end{array}\right\}$$

- searches a critical pair and finds $(e,\ I(e))$ with rules 8 and 6. Rule 6 is not necessary anymore as the rule $I(I(y)) \to y$ is deduced.

$$\text{So } \mathcal{R}=\left\{\begin{array}{ll} 1. & e.x \to x \\ 2. & I(x).x \to e \\ 3. & (x.y).z \to x(y.z) \\ 4. & I(x).(x.z) \to z \\ 8. & y.e \to y \\ 9. & I(I(y)) \to y \\ 10. & I(e) \to e \end{array}\right\}$$

- searches a critical pair and finds $(e,\ y.I(y))$ with rules 2 and 9.

$$\text{So } \mathcal{R}=\left\{\begin{array}{llll} 1. & e.x \to x & 2. & I(x).x \to e \\ 3. & (x.y).z \to x(y.z) & 4. & I(x).(x.z) \to z \\ 8. & y.e \to y & 9. & I(I(y)) \to y \\ 10. & I(e) \to e & 11. & y.I(y) \to e \end{array}\right\}$$

- searches a critical pair and finds $(x,\ y.(I(y).x))$ with rules 3 and 11.

$$\text{So } \mathcal{R}=\left\{\begin{array}{llll} 1. & e.x \to x & 2. & I(x).x \to e \\ 3. & (x.y).z \to x(y.z) & 4. & I(x).(x.z) \to z \\ 8. & y.e \to y & 9. & I(I(y)) \to y \\ 10. & I(e) \to e & 11. & y.I(y) \to e \\ 12. & y.(I(y).x) \to x & & \end{array}\right\}$$

- searches a critical pair and finds $(e,\ x.(y.I(x.y)))$ with rules 3 and 11.

$$\text{So } \mathcal{R}=\left\{\begin{array}{llll} 1. & e.x \to x & 2. & I(x).x \to e \\ 3. & (x.y).z \to x(y.z) & 4. & I(x).(x.z) \to z \\ 8. & y.e \to y & 9. & I(I(y)) \to y \\ 10. & I(e) \to e & 11. & y.I(y) \to e \\ 12. & y.(I(y).x) \to x & 13. & x.(y.I(x.y)) \to e \end{array}\right\}$$

- searches a critical pair and finds $(y.I(x.y),\ I(x).e)$ with rules 4 and 13. Rule 13 becomes redundant.

$$\text{So } \mathcal{R}=\left\{\begin{array}{llll} 1. & e.x \to x & 2. & I(x).x \to e \\ 3. & (x.y).z \to x(y.z) & 4. & I(x).(x.z) \to z \\ 8. & y.e \to y & 9. & I(I(y)) \to y \\ 10. & I(e) \to e & 11. & y.I(y) \to e \\ 12. & y.(I(y).x) \to x & 14. & y.I(x.y) \to I(x).e \end{array}\right\}$$

- searches a critical pair and finds $(I(x.y),\ I(x).I(y))$ with rules 4 and 14. With this new rule, rule 14 can be removed.

$$\text{So } \mathcal{R}=\left\{\begin{array}{llll} 1. & e.x \to x & 2. & I(x).x \to e \\ 3. & (x.y).z \to x(y.z) & 4. & I(x).(x.z) \to z \\ 8. & y.e \to y & 9. & I(I(y)) \to y \\ 10. & I(e) \to e & 11. & y.I(y) \to e \\ 12. & y.(I(y).x) \to x & 15. & I(x.y) \to I(x).I(y) \end{array}\right\}$$

- no more critical pairs are found and the computation stops.

Appendix B

Proof that the ancestor approximation gives a finite automaton [Gen98a]

The ancestor approximation does not depend of the substitution, so $\gamma(l \to r, q, \sigma)$ can be written $\gamma(l \to r, q)$.

If the arity of the symbols of $\mathcal{F}$ and $\mathcal{F}$ are finite, if $\mathcal{Q}$ is finite and if the new states $\mathcal{Q}_{new}$ is finite then $\mathcal{Q}_u$ is finite. Moreover the number of transitions that can be added to Δ is finite, thus the automaton $\mathcal{L}(\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A}))$ is finite.

By assumption $\mathcal{F}$ and $\mathcal{Q}$ are finite, it is sufficient to prove that $\mathcal{Q}_{new}$ is finite to prove the theorem.

For the ancestor approximation,

(1) $\mathcal{Q}_{new} = \{x_i(\gamma(l \to r, q)) | l \to r \in \mathcal{R}, q \in \mathcal{Q}_u, 1 \leq i \leq Card(Pos_{\mathcal{F}}(r))\}$.

If we apply $\mathcal{Q}_u = \mathcal{Q} \bigcup \mathcal{Q}_{new}$ on (), then $\mathcal{Q}_{new} = \mathcal{Q}_1 \bigcup \mathcal{Q}_2$ with:

- $\mathcal{Q}_1 = \{x_i(\gamma(l \to r, q)) | l \to r \in \mathcal{R}, q \in \mathcal{Q}, 1 \leq i \leq Card(Pos_{\mathcal{F}}(r))\}$
- $\mathcal{Q}_2 = \{x_i(\gamma(l \to r, q)) | l \to r \in \mathcal{R}, q \in \mathcal{Q}_{new}, 1 \leq i \leq Card(Pos_{\mathcal{F}}(r))\}$

Every state of $\mathcal{Q}_2$ has the form:

$$x_{i_1}(\gamma(l_1 \to r_1, x_{i_2}(\gamma(l_2 \to r_2, \cdots x_{i_n}(\gamma(l_n \to r_n, q)) \cdots))))$$

With $l_j \to r_j \in \mathcal{R}, q \in \mathcal{Q}, 1 \leq i_j \leq Card(Pos_{\mathcal{F}}(r))$ and $1 \leq j \leq n$. The second condition of Definition 14 states:

$\forall l_1 \to r_1,\ l_2 \to r_2 \in \mathcal{R},\ \forall q \in \mathcal{Q}_u,\ 1 \leq i \leq Card(Pos_{\mathcal{F}}(r))$

$$\gamma(l_2 \to r_2, x_i(\gamma(l_1 \to r_1, q))) = \gamma(l_2 \to r_2, q)$$

Thus

$$\gamma(l_1 \to r_1, x_{i_2}(\gamma(l_2 \to r_2, \cdots x_{i_n}(\gamma(l_n \to r_n, q)) \cdots))) = \gamma(l_1 \to r_1, q)$$

And so

$$x_{i_1}(\gamma(l_1 \to r_1, x_{i_2}(\gamma(l_2 \to r_2, \cdots x_{i_n}(\gamma(l_n \to r_n, q)) \cdots)))) = x_{i_1}(\gamma(l_1 \to r_1, q))$$

It gives $\mathcal{Q}_2 \subseteq \mathcal{Q}_1$ and $\mathcal{Q}_{new} = \mathcal{Q}_1$. As $\mathcal{Q}$, $\mathcal{R}$, $\mathcal{X}$, $Pos_{\mathcal{F}}(r)$ are finite, then $\mathcal{Q}_1$ and $\mathcal{Q}_{new}$ are also finite.

Appendix C

Proof of the completeness extended to non left-linear TRS

The Theorem 4 says: Given a tree automaton $\mathcal{A}$ and a TRS $\mathcal{R}$, $\mathcal{L}(\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A}))$ the corresponding approximation automaton and $\mathcal{Q}$ its set of states. For all non left-linear rule $l \to r \in \mathcal{R}$, for all non linear variables x of l, for all states $q_1, ..., q_n \in \mathcal{Q}$ matched by x, if either $q_1 = ... = q_n$ or $\mathcal{L}(\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A}), q_1) \bigcap ... \bigcap \mathcal{L}(\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A}), q_n) = \emptyset$ then:

$$\mathcal{R}^*(\mathcal{L}(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A}))$$

To prove this theorem we need to introduce the notion of *ground context*. A *ground context* is a term of $\mathcal{T}(\mathcal{F} \cup \{\Box\})$ with exactly one occurremce of $\Box$, where $\Box$ is a special constant not occurring in $\mathcal{F}$. For any term $t \in \mathcal{T}(\mathcal{F})$, $C[t]$ denotes the term obtained after the replacement of $\Box$ by t in the ground context $C[]$.

The proof had been extracted from [GK00b].

Proof Assume that there exists a term t such that $t \in \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$ and $t \notin \mathcal{L}(\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A}))$. The term t is such that $t \notin \mathcal{L}(\mathcal{A})$. Otherwise, $t \in \mathcal{L}(\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A}))$ by Theorem 3, since by contruction of $\mathcal{L}(\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A}))$ we trivially have $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A}))$. Hence, there exists a term $s \in \mathcal{L}(\mathcal{A})$ such that $s \to_{\mathcal{R}}^{+} t$. On this rewite chain, from s to t, let t_1, t_2 be the first two terms such that $t_1 \in \mathcal{L}(\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A}))$, $t_1 \to_{\mathcal{R}} t_2$ and $t_2 \notin \mathcal{L}(\mathcal{T}_{\mathcal{R}}\uparrow(\mathcal{A}))$. Assume that $t_1 = C[l\sigma]$, $t_2 = C[r\sigma]$ and $l \to r \in \mathcal{R}$. Furthermore, let $C'[]$ be a ground context such that $l = C'[x_1, \cdots, x_n]$ with $\{x_1, \cdots, x_n\} = Varl(l)$. Thus $l\sigma =$

$C'[x_1\sigma, \cdots, x_n\sigma]$. Since $t_1 = C[l\sigma] \in \mathcal{L}(\mathcal{T}_\mathcal{R}\uparrow(\mathcal{A}))$, we know that there exists a final state $q \in \mathcal{Q}_f$ of $\mathcal{T}_\mathcal{R}\uparrow(\mathcal{A})$ such that $C[l\sigma] \rightarrow_{\mathcal{T}_\mathcal{R}\uparrow(\mathcal{A})} q$. Furthermore, by construction of tree automata, we obtain that there exists also a state $q' \in \mathcal{Q}$ such that $l\sigma \rightarrow^*_{\mathcal{T}_\mathcal{R}\uparrow(\mathcal{A})} q'$ and $C[q'] \rightarrow^*_{\mathcal{T}_\mathcal{R}\uparrow(\mathcal{A})} q'$. Similarly, from $l\sigma = C'[x_1\sigma, \cdots, x_n\sigma] \rightarrow^*_{\mathcal{T}_\mathcal{R}\uparrow(\mathcal{A})} q'$, we can deduce that there exists states $q_1, \cdots, q_n \in \mathcal{Q}$ such that $x_1\sigma \rightarrow^*_{\mathcal{T}_\mathcal{R}\uparrow(\mathcal{A})} q_1$, $\cdots$, $x_n\sigma \rightarrow^*_{\mathcal{T}_\mathcal{R}\uparrow(\mathcal{A})} q_n$ and $C'[q_1, \cdots, q_n] \rightarrow^*_{\mathcal{T}_\mathcal{R}\uparrow(\mathcal{A})} q'$.

Now, assume that there exists a $\mathcal{Q}$-substitution $\mu \in \Sigma(\mathcal{Q}, \mathcal{X})$ such that $\mu x_i = q_i$ for $i \in [1, \cdots, n]$. Then, we would have $l\mu = C'[q_1, \cdots, q_n]$ and thus $l\mu \rightarrow^*_{\mathcal{T}_\mathcal{R}\uparrow(\mathcal{A})} q'$. By construction of $\mathcal{T}_\mathcal{R}\uparrow(\mathcal{A})$, we know that $l\mu \rightarrow^*_{\mathcal{T}_\mathcal{R}\uparrow(\mathcal{A})} q'$ implies $r\mu \rightarrow^*_{\mathcal{T}_\mathcal{R}\uparrow(\mathcal{A})} q'$. We thus have $x_1\sigma \rightarrow^*_{\mathcal{T}_\mathcal{R}\uparrow(\mathcal{A})} q_1$, $\cdots$, $x_n\sigma \rightarrow^*_{\mathcal{T}_\mathcal{R}\uparrow(\mathcal{A})} q_n$ and $r\mu \rightarrow^*_{\mathcal{T}_\mathcal{R}\uparrow(\mathcal{A})} q'$ where μ maps x_i to q_i for $i \in [1, \cdots, n]$. Hence, $r\sigma \rightarrow^*_{\mathcal{T}_\mathcal{R}\uparrow(\mathcal{A})} q'$ and finally $t_2 = C[r\sigma] \rightarrow_{\mathcal{T}_\mathcal{R}\uparrow(\mathcal{A})} q$ with $q \in \mathcal{Q}_f$, which is a contradiction with the fact that $t_2 \notin \mathcal{L}(\mathcal{T}_\mathcal{R}\uparrow(\mathcal{A}))$.

Consequently, it is not possible to build a $\mathcal{Q}$-substitution $\mu \in \Sigma(\mathcal{Q}, \mathcal{X})$ such that $\mu x_i = q_i$ for $i \in [1, \cdots, n]$. The only reason why μ cannot be a $\mathcal{Q}$-substitution is that there is at least two distinct indexes $i, j \in [1, \cdots, n]$ such that $x_i = x_j$ and $q_i \neq q_j$. Hence the rule is not left linear and the non linear variable $x_i = x_j$ matches, at least, two distinct states q_i and q_j. We can generalize this to all the occurrences of varible x_i. Let $\mathcal{C} = \{k | x_k = x_i\}$. Since all variable x_k with $k \in \mathcal{C}$ are the same, we obtain that there exists a term $u \in \mathcal{T}(\mathcal{F})$ such that $\forall k \in \mathcal{C} : x_k\sigma = u \rightarrow_{\mathcal{T}_\mathcal{R}\uparrow(\mathcal{A})} q_k$. Hence, $\forall k \in \mathcal{C} : \{u\} \subseteq \mathcal{L}(\mathcal{T}_\mathcal{R}\uparrow(\mathcal{A}), q_k)$. Moreover, since we already know that we have at least $i, j \in \mathcal{C}$ and $q_i \neq q_j$ we obtain that $\{u\} \subseteq \bigcap_{k \in \mathcal{C}} \mathcal{L}(\mathcal{T}_\mathcal{R}\uparrow(\mathcal{A}), q_k) \neq \emptyset$, which contredicts the hypothesis of the theorem.

Hence, $t_2 \in \mathcal{L}(\mathcal{T}_\mathcal{R}\uparrow(\mathcal{A}))$ and by applying the same reasoning on all the terms on the rewrite chain between t_2 and t, we finally obtain that $t \in \mathcal{L}(\mathcal{T}_\mathcal{R}\uparrow(\mathcal{A}))$. $\diamond$

Appendix D

Needham-Schroeder input file for Timbuk

Ops mesg:3 encr:3 N:3 cons:2 A:0 B:0 S:0 o:0 suc:1 agt:1 serv:1 U:2 sharekey:2 pubkey:1 c_init:3 c_resp:3 add:1 goal:2 LHS:0 hash1:2 hash2:3 pref:1 sid:2 key:3 cert:2 pms:2 session:2 un:0 deux:0 null:0 t0:0

Vars x x1 y z u s m t a b a_18 a_17 a_16 a_15 a_14 a_13 a_12 a_11 a_10 a_9 a_8 a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0 b_18 b_17 b_16 b_15 b_14 b_13 b_12 b_11 b_10 b_9 b_8 b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0 t_18 t_17 t_16 t_15 t_14 t_13 t_12 t_11 t_10 t_9 t_8 t_7 t_6 t_5 t_4 t_3 t_2 t_1 t_0

TRS R

goal(agt(a), agt(b)) -> U(LHS, mesg(agt(a), agt(b), encr(pubkey(agt(b)), agt(a), cons(N(agt(a), agt(b), t0), agt(a)))))

mesg(a_4, agt(b), encr(pubkey(agt(b)), a_3, cons(N(a_1, b_1, t_1), agt(a)))) -> U(LHS, mesg(agt(b), agt(a), encr(pubkey(agt(a)), agt(b), cons(N(a_1, b_1, t_1), N(agt(b), agt(a), t0)))))

mesg(a_6, agt(a), encr(pubkey(agt(a)), a_5, cons(N(agt(a), agt(b), t0), N(a_2, b_2, t_2)))) -> U(LHS, mesg(agt(a), agt(b), encr(pubkey(agt(b)), agt(a), N(a_2, b_2, t_2))))

mesg(a_6, agt(a), encr(pubkey(agt(a)), a_5, cons(N(agt(a), agt(b), t0), N(a_2, b_2, t_2)))) -> U(LHS, c_init(agt(a), agt(b), a_5))

mesg(a_8, agt(b), encr(pubkey(agt(b)), a_7, N(agt(b), agt(a), t0))) ->
U(LHS, c_resp(agt(b), agt(a), a_7))

U(cons(x, y), z) -> U(LHS, add(x))
U(cons(x, y), z) -> U(LHS, add(y))
U(encr(pubkey(agt(o)), y, z), u) -> U(LHS, add(z))
U(encr(pubkey(agt(suc(x))), y, z), u) -> U(LHS, add(z))
U(mesg(x, y, z), u) -> U(LHS, add(z))
add(x) -> x
U(x, U(y, z)) -> U(U(x, y), z)
U(U(x, y), z) -> U(x, U(y, z))
U(x, y) -> U(y, x)

Automaton automat
States q0 q1 q2 q3 q4 q5 q6 q7 q11 q13 qs1 qs2 qt[0–5]
Final States q13
Prior
null -> q13 t0-> qt0
Transitions

o -> q0	suc(q0) -> q0	agt(q0) -> q3
A -> q1		agt(q1) -> q4
B -> q2		agt(q2) -> q5
U(q13, q13) -> q13		
goal(q4, q5) -> q13	goal(q5, q4) -> q13	
goal(q3, q3) -> q13	goal(q4, q3) -> q13	
goal(q3, q4) -> q13	goal(q5, q3) -> q13	goal(q3, q5) -> q13
agt(q0) -> q13	agt(q1) -> q13	agt(q2) -> q13
mesg(q13, q13, q13) -> q13	cons(q13, q13) -> q13	encr(q13, q3, q13) -> q13
pubkey(q3) -> q13	pubkey(q4) -> q13	pubkey(q5) -> q13
N(q3, q3, qt0) -> q13	N(q3, q4, qt0) -> q13	N(q3, q5, qt0) -> q13

Approximation R1

States q[0–89] qt[0–5]

Rules

[U(LHS, mesg(agt(q2), agt(q2), encr(pubkey(agt(q2)), agt(q2), cons(N(agt(q2), agt(q2), qt0), agt(q2))))) -> q13] -> [LHS -> q13 agt(q2) -> q5 serv(q7) -> q6 agt(q2) -> q5 N(q5, q5, qt0) -> q15 cons(q15, q5) -> q16 pubkey(q5) -> q17 encr(q17, q5, q16) -> q13 mesg(q5, q5, q13) -> q13]

[U(LHS, mesg(agt(q2), agt(q1), encr(pubkey(agt(q1)), agt(q2), cons(N(agt(q2), agt(q1), qt0), agt(q2))))) -> q13] -> [LHS -> q13 agt(q2) -> q5 serv(q7) -> q6 agt(q1) -> q4 N(q5, q4, qt0) -> q19 cons(q19, q5) -> q20 pubkey(q4) -> q21 encr(q21, q5, q20) -> q13 mesg(q5, q4, q13) -> q13]

[U(LHS, mesg(agt(q2), agt(q0), encr(pubkey(agt(q0)), agt(q2), cons(N(agt(q2), agt(q0), qt0), agt(q2))))) -> q13] -> [LHS -> q13 agt(q2) -> q5 serv(q7) -> q6 agt(q0) -> q3 N(q5, q3, qt0) -> q23 cons(q23, q5) -> q24 pubkey(q3) -> q25 encr(q25, q5, q24) -> q13 mesg(q5, q3, q13) -> q13]

[U(LHS, mesg(agt(q1), agt(q2), encr(pubkey(agt(q2)), agt(q1), cons(N(agt(q1), agt(q2), qt0), agt(q1))))) -> q13] -> [LHS -> q13 agt(q1) -> q4 serv(q7) -> q6 agt(q2) -> q5 N(q4, q5, qt0) -> q27 cons(q27, q4) -> q28 pubkey(q5) -> q17 encr(q17, q4, q28) -> q13 mesg(q4, q5, q13) -> q13]

[U(LHS, mesg(agt(q1), agt(q1), encr(pubkey(agt(q1)), agt(q1), cons(N(agt(q1), agt(q1), qt0), agt(q1))))) -> q13] -> [LHS -> q13 agt(q1) -> q4 serv(q7) -> q6 agt(q1) -> q4 N(q4, q4, qt0) -> q30 cons(q30, q4) -> q31 pubkey(q4) -> q21 encr(q21, q4, q31) -> q13 mesg(q4, q4, q13) -> q13]

[U(LHS, mesg(agt(q1), agt(q0), encr(pubkey(agt(q0)), agt(q1), cons(N(agt(q1), agt(q0), qt0), agt(q1))))) -> q13] -> [LHS -> q13 agt(q1) -> q4 serv(q7) -> q6 agt(q0) -> q3 N(q4, q3, qt0) -> q33 cons(q33, q4) -> q34 pubkey(q3) -> q25 encr(q25, q4, q34) -> q13 mesg(q4, q3, q13) -> q13]

[U(LHS, mesg(agt(q0), agt(q2), encr(pubkey(agt(q2)), agt(q0), cons(N(agt(q0),

agt(q2), qt0), agt(q0))))) -> q13] -> [LHS -> q13 agt(q0) -> q3 serv(q7) -> q6 agt(q2) -> q5 N(q3, q5, qt0) -> q36 cons(q36, q3) -> q37 pubkey(q5) -> q17 encr(q17, q3, q37) -> q13 mesg(q3, q5, q13) -> q13]

[U(LHS, mesg(agt(q0), agt(q1), encr(pubkey(agt(q1)), agt(q0), cons(N(agt(q0), agt(q1), qt0), agt(q0))))) -> q13] -> [LHS -> q13 agt(q0) -> q3 serv(q7) -> q6 agt(q1) -> q4 N(q3, q4, qt0) -> q39 cons(q39, q3) -> q40 pubkey(q4) -> q21 encr(q21, q3, q40) -> q13 mesg(q3, q4, q13) -> q13]

[U(LHS, mesg(agt(q0), agt(q0), encr(pubkey(agt(q0)), agt(q0), cons(N(agt(q0), agt(q0), qt0), agt(q0))))) -> q13] -> [LHS -> q13 agt(q0) -> q3 serv(q7) -> q6 agt(q0) -> q3 N(q3, q3, qt0) -> q42 cons(q42, q3) -> q43 pubkey(q3) -> q25 encr(q25, q3, q43) -> q13 mesg(q3, q3, q13) -> q13]

[U(LHS, mesg(agt(q2), agt(q2), encr(pubkey(agt(q2)), agt(q2), cons(N(q5, q5, qt0), N(agt(q2), agt(q2), qt0))))) -> q13] -> [LHS -> q13 agt(q2) -> q5 serv(q7) -> q6 agt(q2) -> q5 N(q5, q5, qt0) -> q15 N(q5, q5, qt0) -> q15 cons(q15, q15) -> q47 pubkey(q5) -> q17 encr(q17, q5, q47) -> q13 mesg(q5, q5, q13) -> q13]

[U(LHS, mesg(agt(q2), agt(q2), encr(pubkey(agt(q2)), agt(q2), cons(N(a_1, b_1, t_1), N(agt(q2), agt(q2), qt0))))) -> q13] -> [LHS -> q13 agt(q2) -> q5 serv(q7) -> q6 agt(q2) -> q5 N(q5, q5, qt0) -> q15 N(a_1, b_1, t_1) -> q45 cons(q45, q15) -> q46 pubkey(q5) -> q17 encr(q17, q5, q46) -> q13 mesg(q5, q5, q13) -> q13]

[U(LHS, mesg(agt(q2), agt(q1), encr(pubkey(agt(q1)), agt(q2), cons(N(q5, q4, qt0), N(agt(q2), agt(q1), qt0))))) -> q13] -> [LHS -> q13 agt(q2) -> q5 serv(q7) -> q6 agt(q1) -> q4 N(q5, q4, qt0) -> q19 N(q5, q4, qt0) -> q19 cons(q19, q19) -> q50 pubkey(q4) -> q21 encr(q21, q5, q50) -> q13 mesg(q5, q4, q13) -> q13]

[U(LHS, mesg(agt(q2), agt(q1), encr(pubkey(agt(q1)), agt(q2), cons(N(a_2, b_2, t_2), N(agt(q2), agt(q1), qt0))))) -> q13] -> [LHS -> q13 agt(q2) -> q5 serv(q7) -> q6 agt(q1) -> q4 N(q5, q4, qt0) -> q19 N(a_2, b_2, t_2) -> q48 cons(q48, q19) -> q49 pubkey(q4) -> q21 encr(q21, q5, q49) -> q13 mesg(q5, q4, q13) -> q13]

[U(LHS, mesg(agt(q2), agt(q0), encr(pubkey(agt(q0)), agt(q2), cons(N(q5, q3, qt0), N(agt(q2), agt(q0), qt0))))) -> q13] -> [LHS -> q13 agt(q2) -> q5 serv(q7) -> q6 agt(q0) -> q3 N(q5, q3, qt0) -> q23 N(q5, q3, qt0) -> q23 cons(q23, q23) -> q53 pubkey(q3) -> q25 encr(q25, q5, q53) -> q13 mesg(q5, q3, q13) -> q13]

[U(LHS, mesg(agt(q2), agt(q0), encr(pubkey(agt(q0)), agt(q2), cons(N(a_3, b_3, t_3), N(agt(q2), agt(q0), qt0))))) -> q13] -> [LHS -> q13 agt(q2) -> q5 serv(q7) -> q6 agt(q0) -> q3 N(q5, q3, qt0) -> q23 N(a_3, b_3, t_3) -> q51 cons(q51, q23) -> q52 pubkey(q3) -> q25 encr(q25, q5, q52) -> q13 mesg(q5, q3, q13) -> q13]

[U(LHS, mesg(agt(q1), agt(q2), encr(pubkey(agt(q2)), agt(q1), cons(N(q4, q5, qt0), N(agt(q1), agt(q2), qt0))))) -> q13] -> [LHS -> q13 agt(q1) -> q4 serv(q7) -> q6 agt(q2) -> q5 N(q4, q5, qt0) -> q27 N(q4, q5, qt0) -> q27 cons(q27, q27) -> q56 pubkey(q5) -> q17 encr(q17, q4, q56) -> q13 mesg(q4, q5, q13) -> q13]

[U(LHS, mesg(agt(q1), agt(q2), encr(pubkey(agt(q2)), agt(q1), cons(N(a_4, b_4, t_4), N(agt(q1), agt(q2), qt0))))) -> q13] -> [LHS -> q13 agt(q1) -> q4 serv(q7) -> q6 agt(q2) -> q5 N(q4, q5, qt0) -> q27 N(a_4, b_4, t_4) -> q54 cons(q54, q27) -> q55 pubkey(q5) -> q17 encr(q17, q4, q55) -> q13 mesg(q4, q5, q13) -> q13]

[U(LHS, mesg(agt(q1), agt(q1), encr(pubkey(agt(q1)), agt(q1), cons(N(q4, q4, qt0), N(agt(q1), agt(q1), qt0))))) -> q13] -> [LHS -> q13 agt(q1) -> q4 serv(q7) -> q6 agt(q1) -> q4 N(q4, q4, qt0) -> q30 N(q4, q4, qt0) -> q30 cons(q30, q30) -> q59 pubkey(q4) -> q21 encr(q21, q4, q59) -> q13 mesg(q4, q4, q13) -> q13]

[U(LHS, mesg(agt(q1), agt(q1), encr(pubkey(agt(q1)), agt(q1), cons(N(a_5, b_5, t_5), N(agt(q1), agt(q1), qt0))))) -> q13] -> [LHS -> q13 agt(q1) -> q4 serv(q7) -> q6 agt(q1) -> q4 N(q4, q4, qt0) -> q30 N(a_5, b_5, t_5) -> q57 cons(q57, q30) -> q58 pubkey(q4) -> q21 encr(q21, q4, q58) -> q13 mesg(q4, q4, q13) -> q13]

[U(LHS, mesg(agt(q1), agt(q0), encr(pubkey(agt(q0)), agt(q1), cons(N(q4, q3, qt0), N(agt(q1), agt(q0), qt0))))) -> q13] -> [LHS -> q13 agt(q1) -> q4 serv(q7) -> q6 agt(q0) -> q3 N(q4, q3, qt0) -> q33 N(q4, q3, qt0) -> q33 cons(q33, q33) ->

q62 pubkey(q3) -> q25 encr(q25, q4, q62) -> q13 mesg(q4, q3, q13) -> q13]

[U(LHS, mesg(agt(q1), agt(q0), encr(pubkey(agt(q0)), agt(q1), cons(N(a_6, b_6, t_6), N(agt(q1), agt(q0), qt0))))) -> q13] -> [LHS -> q13 agt(q1) -> q4 serv(q7) -> q6 agt(q0) -> q3 N(q4, q3, qt0) -> q33 N(a_6, b_6, t_6) -> q60 cons(q60, q33) -> q61 pubkey(q3) -> q25 encr(q25, q4, q61) -> q13 mesg(q4, q3, q13) -> q13]

[U(LHS, mesg(agt(q0), agt(q2), encr(pubkey(agt(q2)), agt(q0), cons(N(q3, q5, qt0), N(agt(q0), agt(q2), qt0))))) -> q13] -> [LHS -> q13 agt(q0) -> q3 serv(q7) -> q6 agt(q2) -> q5 N(q3, q5, qt0) -> q36 N(q3, q5, qt0) -> q36 cons(q36, q36) -> q65 pubkey(q5) -> q17 encr(q17, q3, q65) -> q13 mesg(q3, q5, q13) -> q13]

[U(LHS, mesg(agt(q0), agt(q2), encr(pubkey(agt(q2)), agt(q0), cons(N(a_7, b_7, t_7), N(agt(q0), agt(q2), qt0))))) -> q13] -> [LHS -> q13 agt(q0) -> q3 serv(q7) -> q6 agt(q2) -> q5 N(q3, q5, qt0) -> q36 N(a_7, b_7, t_7) -> q63 cons(q63, q36) -> q64 pubkey(q5) -> q17 encr(q17, q3, q64) -> q13 mesg(q3, q5, q13) -> q13]

[U(LHS, mesg(agt(q0), agt(q1), encr(pubkey(agt(q1)), agt(q0), cons(N(q3, q4, qt0), N(agt(q0), agt(q1), qt0))))) -> q13] -> [LHS -> q13 agt(q0) -> q3 serv(q7) -> q6 agt(q1) -> q4 N(q3, q4, qt0) -> q39 N(q3, q4, qt0) -> q39 cons(q39, q39) -> q68 pubkey(q4) -> q21 encr(q21, q3, q68) -> q13 mesg(q3, q4, q13) -> q13]

[U(LHS, mesg(agt(q0), agt(q1), encr(pubkey(agt(q1)), agt(q0), cons(N(a_8, b_8, t_8), N(agt(q0), agt(q1), qt0))))) -> q13] -> [LHS -> q13 agt(q0) -> q3 serv(q7) -> q6 agt(q1) -> q4 N(q3, q4, qt0) -> q39 N(a_8, b_8, t_8) -> q66 cons(q66, q39) -> q67 pubkey(q4) -> q21 encr(q21, q3, q67) -> q13 mesg(q3, q4, q13) -> q13]

[U(LHS, mesg(agt(q0), agt(q0), encr(pubkey(agt(q0)), agt(q0), cons(N(q3, q3, qt0), N(agt(q0), agt(q0), qt0))))) -> q13] -> [LHS -> q13 agt(q0) -> q3 serv(q7) -> q6 agt(q0) -> q3 N(q3, q3, qt0) -> q42 N(q3, q3, qt0) -> q42 cons(q42, q42) -> q71 pubkey(q3) -> q25 encr(q25, q3, q71) -> q13 mesg(q3, q3, q13) -> q13]

[U(LHS, mesg(agt(q0), agt(q0), encr(pubkey(agt(q0)), agt(q0), cons(N(a_9, b_9,

t_9), N(agt(q0), agt(q0), qt0))))) -> q13] -> [LHS -> q13 agt(q0) -> q3 serv(q7) -> q6 agt(q0) -> q3 N(q3, q3, qt0) -> q42 N(a_9, b_9, t_9) -> q69 cons(q69, q42) -> q70 pubkey(q3) -> q25 encr(q25, q3, q70) -> q13 mesg(q3, q3, q13) -> q13]

[U(LHS, mesg(agt(q2), agt(q2), encr(pubkey(agt(q2)), agt(q2), N(a_10, b_10, t_10)))) -> q13] -> [LHS -> q13 agt(q2) -> q5 serv(q7) -> q6 agt(q2) -> q5 N(a_10, b_10, t_10) -> q72 pubkey(q5) -> q17 encr(q17, q5, q72) -> q13 mesg(q5, q5, q13) -> q13]

[U(LHS, mesg(agt(q2), agt(q1), encr(pubkey(agt(q1)), agt(q2), N(a_11, b_11, t_11)))) -> q13] -> [LHS -> q13 agt(q2) -> q5 serv(q7) -> q6 agt(q1) -> q4 N(a_11, b_11, t_11) -> q74 pubkey(q4) -> q21 encr(q21, q5, q74) -> q13 mesg(q5, q4, q13) -> q13]

[U(LHS, mesg(agt(q2), agt(q0), encr(pubkey(agt(q0)), agt(q2), N(a_12, b_12, t_12)))) -> q13] -> [LHS -> q13 agt(q2) -> q5 serv(q7) -> q6 agt(q0) -> q3 N(a_12, b_12, t_12) -> q76 pubkey(q3) -> q25 encr(q25, q5, q76) -> q13 mesg(q5, q3, q13) -> q13]

[U(LHS, mesg(agt(q1), agt(q2), encr(pubkey(agt(q2)), agt(q1), N(a_13, b_13, t_13)))) -> q13] -> [LHS -> q13 agt(q1) -> q4 serv(q7) -> q6 agt(q2) -> q5 N(a_13, b_13, t_13) -> q78 pubkey(q5) -> q17 encr(q17, q4, q78) -> q13 mesg(q4, q5, q13) -> q13]

[U(LHS, mesg(agt(q1), agt(q1), encr(pubkey(agt(q1)), agt(q1), N(a_14, b_14, t_14)))) -> q13] -> [LHS -> q13 agt(q1) -> q4 serv(q7) -> q6 agt(q1) -> q4 N(a_14, b_14, t_14) -> q80 pubkey(q4) -> q21 encr(q21, q4, q80) -> q13 mesg(q4, q4, q13) -> q13]

[U(LHS, mesg(agt(q1), agt(q0), encr(pubkey(agt(q0)), agt(q1), N(a_15, b_15, t_15)))) -> q13] -> [LHS -> q13 agt(q1) -> q4 serv(q7) -> q6 agt(q0) -> q3 N(a_15, b_15, t_15) -> q82 pubkey(q3) -> q25 encr(q25, q4, q82) -> q13 mesg(q4, q3, q13) -> q13]

[U(LHS, mesg(agt(q0), agt(q2), encr(pubkey(agt(q2)), agt(q0), N(a_16, b_16, t_16)))) -> q13] -> [LHS -> q13 agt(q0) -> q3 serv(q7) -> q6 agt(q2) -> q5 N(a_16, b_16, t_16) -> q84 pubkey(q5) -> q17 encr(q17, q3, q84) -> q13 mesg(q3, q5, q13) -> q13]

[U(LHS, mesg(agt(q0), agt(q1), encr(pubkey(agt(q1)), agt(q0), N(a_17, b_17, t_17)))) -> q13] -> [LHS -> q13 agt(q0) -> q3 serv(q7) -> q6 agt(q1) -> q4 N(a_17, b_17, t_17) -> q86 pubkey(q4) -> q21 encr(q21, q3, q86) -> q13 mesg(q3, q4, q13) -> q13]

[U(LHS, mesg(agt(q0), agt(q0), encr(pubkey(agt(q0)), agt(q0), N(a_18, b_18, t_18)))) -> q13] -> [LHS -> q13 agt(q0) -> q3 serv(q7) -> q6 agt(q0) -> q3 N(a_18, b_18, t_18) -> q88 pubkey(q3) -> q25 encr(q25, q3, q88) -> q13 mesg(q3, q3, q13) -> q13]

[U(LHS, c_init(agt(q2),agt(q2),z)) -> q13] -> [LHS -> q13 agt(q2) -> q5 c_init(q5,q5,z) -> q13]

[U(LHS, c_init(agt(q2),agt(q1),z)) -> q13] -> [LHS -> q13 agt(q2) -> q5 agt(q1) -> q4 c_init(q5,q4,z) -> q13]

[U(LHS, c_init(agt(q2),agt(q0),z)) -> q13] -> [LHS -> q13 agt(q2) -> q5 agt(q0) -> q3 c_init(q5,q3,z) -> q13]

[U(LHS, c_init(agt(q1),agt(q1),z)) -> q13] -> [LHS -> q13 agt(q1) -> q4 c_init(q4,q4,z) -> q13]

[U(LHS, c_init(agt(q1),agt(q2),z)) -> q13] -> [LHS -> q13 agt(q1) -> q4 agt(q2) -> q5 c_init(q4,q5,z) -> q13]

[U(LHS, c_init(agt(q1),agt(q0),z)) -> q13] -> [LHS -> q13 agt(q1) -> q4 agt(y) -> q3 c_init(q4,q3,z) -> q13]

[U(LHS, c_init(agt(q0),agt(q2),z)) -> q13] -> [LHS -> q13 agt(q0) -> q3 agt(q2)

-> q5 c_init(q3,q5,z) -> q13]

[U(LHS, c_init(agt(q0),agt(q1),z)) -> q13] -> [LHS -> q13 agt(q0) -> q3 agt(q1) -> q4 c_init(q3,q4,z) -> q13]

[U(LHS, c_init(agt(q0),agt(q0),z)) -> q13] -> [LHS -> q13 agt(q0) -> q3 c_init(q3,q3,z) -> q13]

[U(LHS, c_resp(agt(q2),agt(q2),z)) -> q13] -> [LHS -> q13 agt(q2) -> q5 c_resp(q5,q5,z) -> q13]

[U(LHS, c_resp(agt(q2),agt(q1),z)) -> q13] -> [LHS -> q13 agt(q2) -> q5 agt(q1) -> q4 c_resp(q5,q4,z) -> q13]

[U(LHS, c_resp(agt(q2),agt(q0),z)) -> q13] -> [LHS -> q13 agt(q2) -> q5 agt(q0) -> q3 c_resp(q5,q3,z) -> q13]

[U(LHS, c_resp(agt(q1),agt(q1),z)) -> q13] -> [LHS -> q13 agt(q1) -> q4 c_resp(q4,q4,z) -> q13]

[U(LHS, c_resp(agt(q1),agt(q2),z)) -> q13] -> [LHS -> q13 agt(q1) -> q4 agt(q2) -> q5 c_resp(q4,q5,z) -> q13]

[U(LHS, c_resp(agt(q1),agt(q0),z)) -> q13] -> [LHS -> q13 agt(q1) -> q4 agt(y) -> q3 c_resp(q4,q3,z) -> q13]

[U(LHS, c_resp(agt(q0),agt(q2),z)) -> q13] -> [LHS -> q13 agt(q0) -> q3 agt(q2) -> q5 c_resp(q3,q5,z) -> q13]

[U(LHS, c_resp(agt(q0),agt(q1),z)) -> q13] -> [LHS -> q13 agt(q0) -> q3 agt(q1) -> q4 c_resp(q3,q4,z) -> q13]

[U(LHS, c_resp(agt(q0),agt(q0),z)) -> q13] -> [LHS -> q13 agt(q0) -> q3 c_resp(q3,q3,z) -> q13]

[U(LHS, add(x)) -> q13] -> [LHS -> q13 add(x) -> q13]

[U(U(x,y),z) -> q13] -> [U(x,y) -> q13]

[U(x, U(y,z)) -> q13] -> [U(y,z) -> q13]

Appendix E

Needham-Schroeder approximation automaton

Ops mesg:3 encr:3 N:3 cons:2 A:0 B:0 S:0 o:0 suc:1 agt:1 serv:1 U:2 sharekey:2 pubkey:1 c_init:3 c_resp:3 add:1 goal:2 LHS:0 hash1:2 hash2:3 pref:1 sid:2 key:3 cert:2 pms:2 session:2 un:0 deux:0 null:0 t0:0

Automaton current

States q8:0 q9:0 q10:0 q12:0 q14:0 q15:0 q16:0 q17:0 q18:0 q19:0 q20:0 q21:0 q22:0 q23:0 q24:0 q25:0 q26:0 q27:0 q28:0 q29:0 q30:0 q31:0 q32:0 q33:0 q34:0 q35:0 q36:0 q37:0 q38:0 q39:0 q40:0 q41:0 q42:0 q43:0 q44:0 q45:0 q46:0 q47:0 q48:0 q49:0 q50:0 q51:0 q52:0 q53:0 q54:0 q55:0 q56:0 q57:0 q58:0 q59:0 q60:0 q61:0 q62:0 q63:0 q64:0 q65:0 q66:0 q67:0 q68:0 q69:0 q70:0 q71:0 q72:0 q73:0 q74:0 q75:0 q76:0 q77:0 q78:0 q79:0 q80:0 q81:0 q82:0 q83:0 q84:0 q85:0 q86:0 q87:0 q88:0 q89:0 q0:0 q1:0 q2:0 q3:0 q4:0 q5:0 q6:0 q7:0 q11:0 q13:0 qs1:0 qs2:0 qt0:0 qt1:0 qt2:0 qt3:0 qt4:0 qt5:0

Final States q13

Prior

null -> q13 S -> q7 t0-> qt0

Transitions

add(q53) -> q13	add(q62) -> q13	add(q76) -> q13
add(q82) -> q13	add(q3) -> q13	add(q4) -> q13
add(q5) -> q13	add(q42) -> q13	add(q33) -> q13
add(q23) -> q13	add(q60) -> q13	add(q51) -> q13
add(q88) -> q13	add(q43) -> q13	add(q34) -> q13
add(q24) -> q13	add(q61) -> q13	add(q52) -> q13
add(q13) -> q13		
c_resp(q4,q4,q5) -> q13	c_resp(q4,q3,q5) -> q13	c_resp(q4,q5,q4) -> q13
c_resp(q4,q3,q4) -> q13	c_resp(q5,q4,q5) -> q13	c_resp(q5,q3,q5) -> q13
c_resp(q5,q5,q4) -> q13	c_resp(q5,q3,q4) -> q13	c_resp(q4,q5,q3) -> q13
c_resp(q4,q4,q3) -> q13	c_resp(q5,q5,q3) -> q13	c_resp(q5,q4,q3) -> q13
c_resp(q3,q3,q5) -> q13	c_resp(q3,q4,q5) -> q13	c_resp(q3,q3,q4) -> q13
c_resp(q3,q5,q4) -> q13	c_resp(q4,q5,q5) -> q13	c_resp(q4,q4,q4) -> q13
c_resp(q3,q5,q5) -> q13	c_resp(q3,q4,q4) -> q13	c_resp(q5,q5,q5) -> q13
c_resp(q5,q4,q4) -> q13	c_resp(q4,q3,q3) -> q13	c_resp(q5,q3,q3) -> q13
c_resp(q3,q5,q3) -> q13	c_resp(q3,q4,q3) -> q13	c_resp(q3,q3,q3) -> q13
c_init(q4,q5,q4) -> q13	c_init(q4,q4,q5) -> q13	c_init(q5,q5,q4) -> q13
c_init(q5,q4,q5) -> q13	c_init(q4,q5,q3) -> q13	c_init(q4,q4,q3) -> q13
c_init(q5,q5,q3) -> q13	c_init(q5,q4,q3) -> q13	c_init(q4,q3,q4) -> q13
c_init(q4,q3,q5) -> q13	c_init(q5,q3,q4) -> q13	c_init(q5,q3,q5) -> q13
c_init(q4,q5,q5) -> q13	c_init(q4,q3,q3) -> q13	c_init(q4,q4,q4) -> q13
c_init(q5,q4,q4) -> q13	c_init(q5,q5,q5) -> q13	c_init(q5,q3,q3) -> q13
c_init(q3,q3,q4) -> q13	c_init(q3,q4,q4) -> q13	c_init(q3,q5,q4) -> q13
c_init(q3,q3,q5) -> q13	c_init(q3,q4,q5) -> q13	c_init(q3,q5,q5) -> q13
c_init(q3,q5,q3) -> q13	c_init(q3,q4,q3) -> q13	c_init(q3,q3,q3) -> q13
LHS -> q13	null -> q13	S -> q7
t0 -> qt0	o -> q0	suc(q0) -> q0
A -> q1	B -> q2	U(q13,q13) -> q13
goal(q4,q5) -> q13	goal(q5,q4) -> q13	goal(q4,q4) -> q13
goal(q5,q5) -> q13	goal(q3,q3) -> q13	goal(q4,q3) -> q13
goal(q3,q4) -> q13	goal(q5,q3) -> q13	goal(q3,q5) -> q13
agt(q0) -> q3	agt(q1) -> q4	agt(q2) -> q5
agt(q0) -> q13	agt(q1) -> q13	agt(q2) -> q13

mesg(q3,q5,q13) -> q13 mesg(q5,q3,q13) -> q13 mesg(q3,q4,q13) -> q13
mesg(q4,q3,q13) -> q13 mesg(q3,q3,q13) -> q13 mesg(q5,q5,q13) -> q13
mesg(q4,q4,q13) -> q13 mesg(q5,q4,q13) -> q13 mesg(q4,q5,q13) -> q13
mesg(q13,q13,q13) -> q13
cons(q27,q27) -> q56 cons(q19,q19) -> q50 cons(q33,q33) -> q62
cons(q23,q23) -> q53 cons(q30,q30) -> q59 cons(q66,q39) -> q67
cons(q63,q36) -> q64 cons(q15,q15) -> q47 cons(q45,q15) -> q46
cons(q48,q19) -> q49 cons(q51,q23) -> q52 cons(q54,q27) -> q55
cons(q57,q30) -> q58 cons(q60,q33) -> q61 cons(q36,q3) -> q37
cons(q23,q5) -> q24 cons(q39,q3) -> q40 cons(q33,q4) -> q34
cons(q42,q3) -> q43 cons(q15,q5) -> q16 cons(q30,q4) -> q31
cons(q19,q5) -> q20 cons(q27,q4) -> q28 cons(q33,q33) -> q13
cons(q23,q23) -> q13 cons(q51,q23) -> q13 cons(q60,q33) -> q13
cons(q23,q5) -> q13 cons(q33,q4) -> q13 cons(q42,q3) -> q13
cons(q13,q13) -> q13
pubkey(q3) -> q25 pubkey(q4) -> q21 pubkey(q5) -> q17
pubkey(q3) -> q13 pubkey(q4) -> q13 pubkey(q5) -> q13
encr(q17,q4,q56) -> q13 encr(q21,q5,q50) -> q13 encr(q25,q4,q62) -> q13
encr(q25,q5,q53) -> q13 encr(q17,q4,q78) -> q13 encr(q25,q4,q82) -> q13
encr(q21,q4,q80) -> q13 encr(q21,q5,q74) -> q13 encr(q17,q5,q72) -> q13
encr(q25,q5,q76) -> q13 encr(q25,q3,q88) -> q13 encr(q21,q3,q86) -> q13
encr(q17,q3,q84) -> q13 encr(q21,q4,q59) -> q13 encr(q21,q3,q67) -> q13
encr(q17,q3,q64) -> q13 encr(q17,q5,q47) -> q13 encr(q17,q5,q46) -> q13
encr(q21,q5,q49) -> q13 encr(q25,q5,q52) -> q13 encr(q17,q4,q55) -> q13
encr(q21,q4,q58) -> q13 encr(q25,q4,q61) -> q13 encr(q17,q3,q37) -> q13
encr(q25,q5,q24) -> q13 encr(q21,q3,q40) -> q13 encr(q25,q4,q34) -> q13
encr(q25,q3,q43) -> q13 encr(q17,q5,q16) -> q13 encr(q21,q4,q31) -> q13
encr(q21,q5,q20) -> q13 encr(q17,q4,q28) -> q13 encr(q13,q3,q13) -> q13
hash1(q3,q13) -> q13 key(q13,q13,q13) -> q13 pref(q3) -> q13

N(q3,q5,qt0) -> q78 N(q3,q4,qt0) -> q78 N(q3,q3,qt0) -> q78
N(q5,q3,qt0) -> q78 N(q4,q3,qt0) -> q78 N(q5,q5,qt0) -> q78
N(q4,q5,qt0) -> q78 N(q4,q4,qt0) -> q78 N(q5,q4,qt0) -> q78
N(q5,q5,qt0) -> q82 N(q4,q5,qt0) -> q82 N(q3,q5,qt0) -> q82
N(q3,q3,qt0) -> q82 N(q5,q3,qt0) -> q82 N(q4,q3,qt0) -> q82
N(q4,q4,qt0) -> q82 N(q5,q4,qt0) -> q82 N(q3,q4,qt0) -> q82
N(q3,q5,qt0) -> q80 N(q3,q4,qt0) -> q80 N(q3,q3,qt0) -> q80
N(q5,q3,qt0) -> q80 N(q4,q3,qt0) -> q80 N(q5,q5,qt0) -> q80
N(q4,q5,qt0) -> q80 N(q5,q4,qt0) -> q80 N(q4,q4,qt0) -> q80
N(q3,q5,qt0) -> q74 N(q3,q4,qt0) -> q74 N(q3,q3,qt0) -> q74
N(q5,q3,qt0) -> q74 N(q4,q3,qt0) -> q74 N(q5,q5,qt0) -> q74
N(q5,q4,qt0) -> q74 N(q4,q4,qt0) -> q74 N(q4,q5,qt0) -> q74
N(q3,q5,qt0) -> q72 N(q3,q4,qt0) -> q72 N(q3,q3,qt0) -> q72
N(q5,q3,qt0) -> q72 N(q4,q3,qt0) -> q72 N(q4,q5,qt0) -> q72
N(q5,q4,qt0) -> q72 N(q4,q4,qt0) -> q72 N(q5,q5,qt0) -> q72
N(q5,q4,qt0) -> q76 N(q4,q4,qt0) -> q76 N(q3,q4,qt0) -> q76
N(q3,q3,qt0) -> q76 N(q5,q3,qt0) -> q76 N(q4,q3,qt0) -> q76
N(q4,q5,qt0) -> q76 N(q5,q5,qt0) -> q76 N(q3,q5,qt0) -> q76
N(q4,q3,qt0) -> q88 N(q5,q3,qt0) -> q88 N(q4,q3,qt0) -> q86
N(q5,q3,qt0) -> q86 N(q4,q3,qt0) -> q84 N(q5,q3,qt0) -> q84
N(q4,q3,qt0) -> q66 N(q5,q3,qt0) -> q63 N(q4,q5,qt0) -> q45
N(q5,q4,qt0) -> q45 N(q4,q4,qt0) -> q45 N(q4,q3,qt0) -> q45
N(q5,q3,qt0) -> q45 N(q3,q5,qt0) -> q45 N(q3,q4,qt0) -> q45
N(q3,q3,qt0) -> q45 N(q5,q5,qt0) -> q48 N(q4,q4,qt0) -> q48
N(q5,q3,qt0) -> q48 N(q4,q3,qt0) -> q48 N(q4,q5,qt0) -> q48
N(q3,q5,qt0) -> q48 N(q3,q4,qt0) -> q48 N(q3,q3,qt0) -> q48
N(q5,q5,qt0) -> q51 N(q4,q5,qt0) -> q51 N(q5,q4,qt0) -> q51
N(q4,q4,qt0) -> q51 N(q4,q3,qt0) -> q51 N(q3,q5,qt0) -> q51
N(q3,q4,qt0) -> q51 N(q3,q3,qt0) -> q51 N(q5,q5,qt0) -> q54
N(q4,q4,qt0) -> q54 N(q4,q3,qt0) -> q54 N(q5,q3,qt0) -> q54
N(q5,q4,qt0) -> q54 N(q3,q5,qt0) -> q54 N(q3,q4,qt0) -> q54
N(q3,q3,qt0) -> q54 N(q5,q5,qt0) -> q57 N(q4,q5,qt0) -> q57
N(q5,q4,qt0) -> q57 N(q5,q3,qt0) -> q57 N(q4,q3,qt0) -> q57

N(q3,q5,qt0) -> q57	N(q3,q4,qt0) -> q57	N(q3,q3,qt0) -> q57
N(q5,q5,qt0) -> q60	N(q4,q5,qt0) -> q60	N(q5,q4,qt0) -> q60
N(q4,q4,qt0) -> q60	N(q5,q3,qt0) -> q60	N(q3,q5,qt0) -> q60
N(q3,q4,qt0) -> q60	N(q3,q3,qt0) -> q60	N(q3,q5,qt0) -> q36
N(q5,q3,qt0) -> q23	N(q3,q4,qt0) -> q39	N(q4,q3,qt0) -> q33
N(q3,q3,qt0) -> q42	N(q5,q5,qt0) -> q15	N(q4,q4,qt0) -> q30
N(q5,q4,qt0) -> q19	N(q4,q5,qt0) -> q27	N(q4,q4,qt0) -> q13
N(q5,q4,qt0) -> q13	N(q4,q5,qt0) -> q13	N(q5,q5,qt0) -> q13
N(q4,q3,qt0) -> q13	N(q5,q3,qt0) -> q13	N(q3,q3,qt0) -> q13
N(q3,q4,qt0) -> q13	N(q3,q5,qt0) -> q13	

Appendix F

Invariant Example

We are looking here at an example [TRSS01] from the theroy of linear arithmetic.

We have our system that runs as follow:

$$pc{=}1 \longrightarrow x{:=}x{+}2;\ y{:=}y{+}2;\ pc{:=}2$$

$$pc{=}2 \longrightarrow x{:=}x{-}2;\ y{:=}y{+}2;\ pc{:=}1$$

The variable pc can only takes two values, 1 or 2, and the variables x and y are integers.

The initial state of the system is: $pc{=}1 \wedge x{=}0 \wedge y{=}0$.

We can model the system behaviour with a TRS:

system(one, x, y) → system(two, s(s(x)), s(s(y)))
system(two, s(s(w)), z) → system(one, w, s(s(z)))

The initial configuration of the system can be recognized by a tree automaton:

States	q0 q1 q2 qf
Final States	qf
Transitions	
	o → q0
	one → q1
	system(q1,q0,q0) → qf

We can also define an approximation function:

[system(two, s(s(q0)), s(s(q0))) → qf] ⟶ [two → q2 s(q0) → q3 s(q3) → q4]
[system(two, s(s(q0)), s(s(y))) → qf] ⟶ [two → q2 s(q0) → q3 s(q3) → q4 s(y) → q7 s(q7) → q8]
[system(two, s(s(x)), s(s(q0))) → qf] ⟶ [two → q2 s(x) → q5 s(q5) → q6 s(q0) → q3 s(q3) → q4]
[system(two, s(s(x)), s(s(y))) → qf] ⟶ [two → q2 s(x) → q5 s(q5) → q6 s(y) → q7 s(q7) → q8]

[system(one, q0, s(s(q0))) → qf] ⟶ [one → q1 s(q0) → q3 s(q3) → q4]
[system(one, w, s(s(q0))) → qf] ⟶ [one → q1 s(q0) → q3 s(q3) → q4]
[system(one, q0, s(s(z))) → qf] ⟶ [one → q1 s(z) → q9 s(q9) → q10]
[system(one, w, s(s(z))) → qf] ⟶ [one → q1 s(z) → q9 s(q9) → q10]

Using the completion algorithm (Algorithm 2) we get the following approximation automaton:

States	q3 q4 q5 q6 q7 q8 q0 q1 q2 qf
Final States	qf
Transitions	
	s(q8) → q5
	s(q5) → q6
	s(q6) → q7
	s(q4) → q7
	s(q7) → q8
	s(q0) → q3
	s(q3) → q4
	o → q0
	one → q1
	two → q2
	system(q2,q4,q6) → qf
	system(q1,q0,q8) → qf
	system(q2,q4,q4) → qf
	system(q1,q0,q0) → qf

By looking at this automaton we can deduce the following invariant:

$$pc{=}1 \Longrightarrow (x{=}0 \wedge y{=}0) \vee (x{=}0 \wedge y = 4) \vee (x{=}0 \wedge y \geq 8)$$

$$pc{=}2 \Longrightarrow (x{=}2 \wedge y{=}2) \vee (x{=}2 \wedge y = 6) \vee (x{=}2 \wedge y \geq 10)$$

(x=0 $\wedge$ y=0) and (x=0 $\wedge$ y = 4) for pc=1 are easy to deduce, they are transitions of the automaton. It is the same for (x=2 $\wedge$ y=2) and (x=2 $\wedge$ y = 6) for pc=2. The last parts of the invariant, (x=0 $\wedge$ y $\geq$ 8) and (x=2 $\wedge$ y $\geq$ 10), are deduced from the looping on the states q6 and q8 with s(q6) $\rightarrow$ q7, s(q7) $\rightarrow$ q8, s(q8) $\rightarrow$ q5 and s(q5) $\rightarrow$ q6.

Our invariant is identical to the one in [TRSS01]. The computation of the approximation automaton took less than 2 seconds.