THESE DE DOCTORAT

DE L'ÉCOLE NORMALE SUPÉRIEURE DE CACHAN

Présentée par
Mathilde ARNAUD

pour obtenir le grade de

DOCTEUR DE L'ÉCOLE NORMALE SUPÉRIEURE DE CACHAN

Domaine : INFORMATIQUE

Formal verification of secured routing protocols

Thèse présentée et soutenue à Cachan le 13 décembre 2012 devant le jury composé de :

Véronique Cortier	Directrice de thèse
Jean-Michel Couvreur	Examinateur
Stéphanie Delaune	Directrice de thèse
Jean Goubault-Larrecq	Directeur de thèse
Thomas Jensen	Examinateur
Hélène Kirchner	Rapporteur
Yassine Lakhnech	Examinateur
Laure Petrucci	Rapporteur

Laboratoire Spécification et Vérification
ENS CACHAN, CNRS, UMR 8643
61 avenue du Président Wilson, 94235 CACHAN CEDEX (France)

Abstract

Mobile ad hoc networks consist of mobile wireless devices which autonomously organize their infrastructure. In such networks, a central issue, ensured by routing protocols, is to find a route from one device to another. Those protocols use cryptographic mechanisms in order to prevent malicious nodes from compromising the discovered route.

We first propose a calculus for modeling and reasoning about security protocols, including in particular secured routing protocols. Our calculus extends standard symbolic models to take into account the characteristics of routing protocols and to model wireless communication in a more accurate way.

We then give decision procedures for analyzing routing protocols. We use a symbolic model and constraint systems to represent the possible executions of a given protocol. We revisit constraint system solving, providing a complete symbolic representation of the attacker knowledge.

We show that it is possible to automatically discover (in NPTIME) whether there exists a network topology that would allow malicious nodes to mount an attack against a secured routing protocol, for a bounded number of sessions. We also provide a decision procedure for detecting attacks in case the network topology is given a priori.

We also analyze protocols with recursive tests. We provide NPTIME decision procedures for two classes of protocols with recursive tests and for a bounded number of sessions.

Résumé

Les réseaux mobiles ad hoc consistent en un assemblage de machines mobiles qui organisent elle-mêmes leur infrastructure. Dans ces réseaux, déterminer comment les messages doivent circuler pour atteindre leur destination est une fonctionnalité primordiale, qui est assurée par les protocoles de routage. Les protocoles de routage sécurisés utilisent des mécanismes cryptographiques pour empêcher des agents mal intentionnés de compromettre les routes.

Nous proposons un calcul de processus pour modéliser les protocoles sécurisés, et en particulier les protocoles de routage sécurisés. Notre calcul se base sur des modèles symboliques connus que nous enrichissons pour prendre en compte les caractéristiques des protocoles de routage et de la communication sans fil.

Nous fournissons ensuite des procédures de décision qui nous permettent d'analyser des protocoles de routage. Nous mettons en place un modèle symbolique avec des systèmes de contraintes pour représenter les exécutions possibles d'un protocole. Nous revisitons les systèmes de contraintes en donnant une représentation symbolique complète de la connaissance de l'intrus.

Nous montrons qu'on peut décider s'il existe une topologie du réseau permettant une attaque du protocole pour un nombre borné de sessions. Nous fournissons aussi une procédure de décision pour détecter des attaques dans le cas où la topologie du réseau est fixée à l'avance.

Nous analysons aussi des protocoles faisant des tests récursifs. Nous fournissons des procédures de décisions en temps NP pour deux classes de protocoles utilisant des tests récursifs et pour un nombre borné de sessions.

Contents

Chapter 1

Introduction

With the development of digital networks, such as Internet, communication protocols are omnipresent. Digital devices have to interact with each other in order to perform the numerous and complex tasks we have come to expect as commonplace, such as using a mobile phone, sending or receiving electronic mail, making purchases online and so on.

In such applications, security is important. For instance, in the case of an online purchase, the right amount of money has to be paid without leaking the buyer personal information to outside parties. Communication protocols are the rules that govern these interactions. In order to make sure that they guarantee a certain level of security, it is desirable to analyze them. Doing so manually or by testing them is not enough, as attacks can be quite subtle. Some protocols have been used for years before an attack was discovered.

Because of their increasing ubiquity in many important applications, e.g. electronic commerce, a very important research challenge consists in developing methods and verification tools to increase our trust on security protocols, and so on the applications that rely on them. For example, more than 28 billion Euros were spent in France using Internet transactions [BHS10], and the number is growing. Moreover, new types of protocols are continuously appearing in order to face new technological and societal challenges, e.g. electronic voting, electronic passport to name a few.

1.1 Ad hoc networks

1.1.1 Wireless ad-hoc networks

Computers, and more generally electronic devices, communicate with each other to perform a large number of various tasks, such as entering into a transaction over the Internet, finding your way thanks to the GPS, or simply retrieving money from a cash dispenser to name a few. Such communication can be achieved either via a physical medium, often wires, or a wireless medium. Mobile phones, portable computers, wireless sensors are able to send and receive messages over radio waves, wi-fi, bluetooth... Using wireless communication is a cost efficient way of setting up a communication network in a short time when there is no existing infrastructure, or if the network is only temporary.

Wireless networks can be structured around a central point. For instance, in managed wireless networks which are routinely used in home networks, the central device is linked to a wired network and serves as a gateway to the Internet for the other devices on the network, it is called the access point. Sensor networks are also often hierarchised, the sensors all have to transmit the information they gather to a central device. However, there is not always such a hierarchy, some wireless networks are decentralized, without any preexisting structure. Such auto adaptive networks are called *ad hoc networks*.

Ad hoc networks are the focus of many recent research efforts. In particular, mobile and self-organizing networks are of high interest. Examples of applications range from military or rescue operations to interaction among meeting attendees or students during a lecture, and can also include self-organizing wireless sensors or vehicular ad hoc networks.

1.1.2 Routing protocols

Ad hoc networks have no fixed infrastructure: the wireless devices making up the network are arbitrarily located, and thus the way to communicate has to be carefully thought out. Basic communication is achieved by *broadcasting* messages. Any machine within a certain range (whose value depends on the power of the antenna and the wireless medium chosen) of the emitting machine can receive the message. Whereas in a managed wireless network, each machine only communicates with the central access point and ignores all other messages, in an ad hoc setting they have to listen to every message, at least until some sort of organisation can be established. Each of the devices can communicate directly only with the devices that are situated within a certain range. As a consequence, when two distant machines wish to communicate, the data traffic has to travel through the other devices making up the network until it reaches its destination. In a wired network, bringing the data to its destination is a task performed by specific devices called routers. In an ad hoc wireless network, each device can act as a router, propagating messages on behalf of some other device.

An ad hoc network can be thought of as a graph, where the nodes of the graph represent the devices making up the network. Two nodes are linked by an edge in the graph if they are within direct communication range of each other, which means that when one of the devices of that link broadcasts a message, his neighbor can hear it. Finding the paths that the messages must follow in an a priori unknown and constantly changing network topology is a crucial functionality of any ad hoc network. Specific protocols, called *routing protocols*, are designed to ensure this functionality known as *route discovery*.

As an illustrative example, we give a brief description of the route discovery functionality of a basic routing protocol, namely the Dynamic Source Routing protocol (DSR). DSR is a simple routing protocol designed to be used in ad hoc networks.

This protocol is used when a node, that we denote as the *source* node S, wishes to communicate with another node that we will denote as the *destination* node D. The routing method used to transfer data makes use of a *route*, a path in the graph that the data must follow to reach its final destination. In order to perform the routing operation, S thus needs to discover a path in the network leading to D. The route discovery in DSR is divided in two phases: a request phase and a reply phase. During the request phase, messages are sent everywhere over the network, in an effort to reach D. First, the source node broadcasts a message signalling that it is looking for a route towards D. The nodes receiving this message that do not correspond to the intended destination are called *intermediate* nodes. They forward the message after appending their name to the request. When D is reached, the reply phase begins, where the discovered route is conveyed back to S through the intermediate nodes. D. Jonhson and D. Maltz provide a full description of the protocol in [JMB01].

1.1.3 Classification of routing protocols.

Routing protocols can be classified into two groups:

- *proactive* (or periodic): those protocols try to maintain up-to date routing information. In other words, routes are periodically updated. At any time, each nodes knows how to route a packet.
- *reactive* (or on-demand): those protocols establish routes only when it is needed. In other words, a route discovery is initiated only when a source node S wishes to communicate with a destination node D which he does not already know how to reach.

In general, proactive approaches are thought to have shorter latency, as routes are instantly available, while on-demand approaches have a lower overhead, since route discovery only occurs when it is useful. The main drawback of on-demand routing protocols is the fact that communication must be delayed until a route is found. However, information is updated in an on-demand manner, in contrast with proactive protocols. As these protocols maintain up-to-date routing information for all possible destinations, they will establish some routes that will never be used. There are also some *hybrid* protocols that try to combine the advantages of both approaches.

In wired networks, routing protocols usually adopt the proactive approach, which is better adapted to a network whose configuration changes rarely: the updates need not be too frequent. For instance, the routing protocol used in the Internet, the Border Gateway Protocol (BGP) [BGP95], is proactive. Some routing protocols for ad hoc networks also use this method, such as SEAD [HJP03].

Furthermore, routing protocols can also be sorted between the source routing and table routing methods.

- *source routing*: the source node needs to know the entire route between itself and the destination. During the routing phase, the source node provides the route that the message he is sending has to follow, and the intermediate nodes forward the message along this route. This means that every packet carries the route in its header, as each node needs this information in order to forward the data along the way to the destination. Examples of source routing protocols include DSR [JMB01], AnonDSR [SKY05], Ariadne [HPJ05] and endairA [BV04].
- *table routing* (or hop-by-hop routing): each node knows only which is the following node on the route towards a given destination. This information is stored in routing tables. Several methods can be used to compute this next node, mainly link-state and distance-vector. In link-state algorithms, the nodes share information about their neighbors, and each node computes the shortest path towards every destination. In distance vector algorithms, nodes share their estimates of the shortest path for all known nodes, and the neighbors update their routing tables accordingly. Examples of table routing protocols include AODV [PBR99] and its secure version SAODV [ZA02].

Source routing allows one to easily prevent the presence of loops in a route, but in compensation each packet carries the entire route in its header. It is more complicated to detect the formation of loops in the case where table routing is used. This is a serious drawback, as loops are serious flaws: when a routing loop forms, it generates excessive traffic in the loop, and may prevent the nodes of the loop to function in a normal way.

Most of the time, on-demand routing protocols are made of several separate mechanisms: *route discovery*, *route maintenance* and *data transmission*. Route discovery consists in finding a suitable path from a source node to a given destination, and route maintenance is the mechanism used when a link breaks, which can be quite frequent in a mobile network. However, route maintenance often consists in advertising the link failure and prompting the source node to initiate a new route discovery if needed, without any attempt at fixing the existing route. Data transmission is the actual communication phase and occurs when a route has been established through route discovery.

In order to secure data transmission, the route discovery has to be secure too. We focus on the security of route discovery, and when describing a routing protocol we consider only the route discovery mechanism. The security of route discovery is crucial for the functioning of ad hoc wireless networks. Indeed, an adversary can easily paralyze the entire network by attacking the underlying routing protocol. While the routing aspects of mobile ad hoc networks (MANETs) are well-understood, the research activities about security in those networks are still at their beginning.

1.2 Secure Ad Hoc Routing Protocols

Assuming that the routing protocols operate in a friendly environment is unrealistic. Secure routing protocols have been proposed in order to take the adversarial setting into account and protect against an attacker.

AODV [PBR99] and DSR [JMB01] are not secure and in particular do not ensure correction of the discovered route in the presence of malicious nodes. Some protocols aim at fixing this vulnerability. For instance, SAODV [ZA02] is a secured version of AODV, and SRP [PH02] is a security mechanism designed to be used with source routing protocols such as DSR.

Some protocols [SKY05, ZWK$^+$04, BEKXK04] moreover want to ensure anonymity of the participants. They make use of similar techniques as some secure source routing protocols. The computations performed are however more complex in general, as the security guarantees desired are more difficult to obtain.

In order to ensure correctness of the route, nodes executing a routing protocol may have to perform some checks, typically checking that some other node claiming to be their neighbor actually is. Specific protocols are designed to discover neighbors, and they have to be secure in order to derive security of the routing protocol above. The importance of this functionality is explained in [PPS$^+$08], and a method to check whether such protocols are indeed secure is introduced by [PPH08]. We use secure neighborhood discovery as a black box, in the same way as cryptography: we consider that each node knows the list of its neighbors when executing a routing protocol.

1.2.1 Cryptographic primitives

In order to build protocols that protect information, we make use of cryptography: from the Greek words kryptos = "hidden, secret" and graphein = "writing", it denotes the science of hiding information. The need for protecting sensitive data over digital media has prompted a huge development of this field of study. In the symbolic model, the workings of cryptography are captured by cryptographic primitives, that are used as a black box hiding the computational workings of cryptography. The data on which these methods are applied is called the message.

Encryption. Symmetric encryption is the oldest known cryptographic method. Coding a message in order to keep it hidden is called *encryption*. The inverse method, i.e. retrieving the initial message is called *decryption*. Encryption and decryption make use of a *key*. The encryption may be symmetric or asymmetric. For symmetric encryption, the same key is used for encrypting and decrypting a message. Asymmetric encryption involves different keys for the two operations: a *public key* for encryption and a *secret key* for decryption. Decryption should only be possible with the secret key.

Hash function. A cryptographic hash function is a function that takes an arbitrary block of data and returns a bitstring of fixed size called the hash value. A change in the data, whether accidental or intentional, will change the hash value. Computing a hash value is easy, but the opposite operation, i.e. generating a message that has a given hash, should be infeasible. Furthermore, it should be infeasible to find two different messages with the same hash.

Message authentication code. A message authentication code, or MAC, is a short bitstring used to authenticate a message. The function producing a MAC from an arbitrary-length message needs a secret key to perform the computation. Agents possessing the same key will be able to detect changes in the message by computing the MAC and comparing them. The algorithm is sometimes called keyed hash function. The MAC value protects both a message's data integrity and its authenticity.

Digital signatures. A digital signature is a scheme for proving the authenticity of a digital message, as traditional handwritten signatures are proofs of the authenticity of a document. Digital signatures are commonly used for example in software distribution, financial transactions, and in other cases where it is important to detect forgery or tampering. Digital signatures employ a type of asymmetric cryptography, two algorithms make up a signature scheme. The signing algorithm produces a signature, given a message and a private key. The signature verifying algorithm considers as input a message, a verifying key and a signature, and either accepts or rejects the message's claim to authenticity. It should be infeasible to generate a valid signature without the private key.

1.2.2 Examples of routing protocols

Designing secure version of routing protocols is a difficult task. Actually, most routing protocols proposed for wireless ad hoc networks are insecure, attacks have been discovered against them. Those who have no known attacks have mostly only been analysed by informal reasoning. We describe here two routing protocols claimed to be secure that will be used as illustrating examples in this dissertation.

Secure Routing Protocol (SRP) [PH02] is actually not a routing protocol in itself. In fact, it is designed to be applied as an extension of an existing on-demand source routing protocols, such as DSR. The goal of the protocol obtained after extension is to provide correct connectivity information, even in the presence of (non-colluding) attackers. In order to be able to use SRP, the source and destination of the route discovery are required to have a security association, for instance sharing a key K_{SD}.

A syntactic description of SRP is given in Figure 1.1. To discover a route to the destination, the source S constructs a request packet and broadcasts it to its neighbors. The request packet

$\mathsf{req}, \mathsf{rep}$:	identifiers indicating the phase of the execution
Id:	request identifier
q_n:	query sequence number
ℓ, ℓ':	list of nodes
K_{SD}:	key shared between S and D
MAC :	function computing a message authentication code
m, m':	message authentication codes

Request phase:

S broadcasts $\langle \mathsf{req}, S, D, q_n, Id, [], \mathsf{MAC}(K_{SD}, \langle S, D, q_n, Id\rangle)\rangle$

V receives $\langle \mathsf{req}, S, D, q_n, Id, \ell, m\rangle$
V checks that the message was last processed by a neighbor
V broadcasts $\langle \mathsf{req}, S, D, q_n, Id, V :: \ell, m\rangle$

D receives $\langle \mathsf{req}, S, D, q_n, Id, \ell', \mathsf{MAC}(K_{SD}, \langle S, D, q_n, Id\rangle)\rangle$

Reply phase:

D sends $\langle \mathsf{rep}, D, S, \ell', \mathsf{MAC}(K_{SD}, \langle S, D, q_n, Id, \ell'\rangle)\rangle$

V receives $\langle \mathsf{rep}, D, S, \ell', m'\rangle$
V checks that ℓ' is plausible from its point of view
V sends $\langle \mathsf{rep}, D, S, \ell', m'\rangle$

S receives $\langle \mathsf{rep}, D, S, q_n, Id, \ell', \mathsf{MAC}(K_{SD}, \langle S, D, q_n, Id, \ell'\rangle)\rangle$

Figure 1.1: Specification of SRP

contains its name S, the name of the destination D, an identifier of the request Id, a request sequence number q_n to prove the freshness of the route request and to prevent replaying of old requests, a list containing the beginning of a route to D, and a MAC computed over the content of the request with a key K_{SD} shared by S and D. It then waits for an answer containing a route to D with a MAC matching this route, and tests whether it is a plausible route by checking that the route does not contain a loop and that his neighbor in the route is indeed a neighbor of his in the network. Each intermediate node that receives the message first checks that the list representing the route begins with the identifier of one of his neighbors. Then, he appends his identifier to the route accumulated so far in the request and broadcasts the modified request message to his immediate neighbors. During the reply phase, they behave in a similar way, they check that the route is plausible according to their view of the network and they forward the reply along the way. Upon receiving the request packet, the destination checks that the MAC is correct and initiates the reply phase. He sends a message containing the route discovered with a MAC computed over it with the key K_{SD}.

This protocol, although it was analyzed informally and considered secure [PH02], is subject

$\mathsf{req}, \mathsf{rep}$:	identifiers indicating the execution phase
Id:	request identifier
ℓ, ℓ':	list of nodes
l_{sig}, l'_{sig}:	lists of signatures
sig_D, sig_V :	signatures over the reply message

Request phase:

S broadcasts $\langle \mathsf{req}, S, D, Id, [] \rangle$

V receives $\langle \mathsf{req}, S, D, Id, \ell \rangle$
V checks that the message was last processed by a neighbor
V broadcasts $\langle \mathsf{req}, S, D, V :: \ell \rangle$

D receives $\langle \mathsf{req}, S, D, \ell' \rangle$
D checks that the message was last processed by a neighbor

Reply phase:

D sends $\langle \mathsf{rep}, D, S, \ell', sig_D \rangle$

V receives $\langle \mathsf{rep}, D, S, \ell', l'_{sig} \rangle$
V checks that (ℓ', l_{sig}) is a valid pair node list/list of signatures
V sends $\langle \mathsf{rep}, D, S, \ell', sig_V ::\ _{sig} \rangle$

S receives $\langle \mathsf{rep}, D, S, \ell', l'_{sig} \rangle$
S checks that (ℓ', l'_{sig}) is a valid pair node list/list of signatures

Figure 1.2: Specification of EndairA

to attacks [Mar03, BV04]. We describe one of these attacks in Section 1.2.3.

EndairA is a secured routing protocol inspired by another one called Ariadne. The authors, G. Ács, L. Buttyán and I. Vajda, discovered an attack on Ariadne [BV04]. They established a formal model to analyze routing protocols and proved EndairA to be secure for a slightly modified notion of correctness, called route validity. The protocol is called endairA (which is the reverse of Ariadne) because, instead of signing the request, they propose that intermediate nodes should sign the route reply. The aim of signing the route reply is to protect the protocol against the attacks found against Ariadne, which take advantage of the fact that the list could be tampered with during the request phase. There are no known attacks against this routing protocol.

A syntactic description of EndairA is given in Figure 1.2. The initiator of the route discovery process, S, generates a route request containing the identifiers of the source S and the destination D, and a randomly generated identifier Id. Each intermediate node that receives the message for the first time appends its identifier to the route accumulated so far in the request and broadcasts the modified request message to its immediate neighbors.

When the request reaches the destination D, it generates a route reply $\langle \mathsf{rep}, D, S, \ell', sig_D \rangle$ where ℓ' is the accumulated route obtained from the request and $sig_D = [\![\mathsf{rep}, D, S, \ell']\!]_{K_D}$ is a digital signature of D on the other fields of the reply. The reply is sent back to S on the reverse of the route found in the request.

Upon receiving the reply $\langle \mathsf{rep}, D, S, \ell', l_{sig} \rangle$, the intermediate node V verifies that the route ℓ' contains its identifier V and that both the preceding and following identifiers in the list belong to neighboring nodes. The node V also checks that the list of signatures l_{sig} is valid and corresponds to the list of names ℓ'. If these verifications fail, then the message is dropped. Otherwise, V appends its signature to l_{sig} and forwards it to the next node.

When S receives the route reply $\langle \mathsf{rep}, D, S, \ell', l'_{sig} \rangle$, he checks that the signatures in l'_{sig} are valid and correspond with the sequence of nodes in list ℓ'. If these verifications are successful, then S accepts the route ℓ'.

1.2.3 Attacks on routing protocols

We can consider two types of attacks [HPJ05]: *routing disruption* and *resource consumption.* During routing disruption attacks, the aim of the intruder is to prevent the routing protocol from executing in a correct way. In resource consumption attacks, the intruder sends requests over the network to make honest nodes consume their (limited) resources such as bandwidth or memory.

The first kind of attacks can be performed by an intruder trying to route all the traffic through nodes he controls, so as to be able to listen to all communications. He could also try to prevent two nodes from communicating by advertising a false route. The second type of attacks are denial of service attacks. If a protocol satisfies the route correctness property, it is secure against route disruption attacks but not necessarily against resource consumption attacks. We describe below some generic ways of mounting attacks.

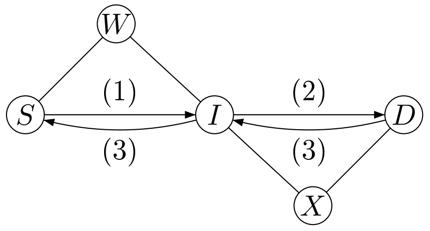
Replay attacks. A replay attack consists in broadcasting without modification a message that the intruder received from an honest agent. A replay attack may for example enable the intruder to steal the identity of an honest agent. In the context of routing protocols, this could shorten the paths going through the intruder, and thus possibly prompt nodes to choose these paths to communicate. As an illustration, we consider again SRP.

In [BV04], an attack is found on SRP for an intruder controlling one node in the network. This attack, described in Figure 1.3 is similar to a replay attack, but the message transferred is modified slightly, in order to fool the destination into signing a path that appears correct.

First, the source node S initiates the protocol by sending a request message

$$\langle \mathsf{req}, S, D, q_n, Id, [], \mathsf{MAC}(K_{SD}, \langle S, D, q_n, Id \rangle) \rangle.$$

As the intruder is a neighbor of the source, he receives the message. This enables him to send the forged message $\langle \mathsf{req}, S, D, q_n, Id, [X, W], \mathsf{MAC}(K_{SD}, \langle S, D, q_n, Id \rangle) \rangle$. When the destination D receives this message, he checks that the node at the head of the list is one of his neighbors. In this case, the test succeeds, as X is really a neighbor of D, even though X has not sent the message. The destination node then computes a MAC over the *false* route $[X, W]$, and sends



1. $\langle \mathsf{req}, S, D, q_n, Id, [], \mathsf{MAC}(K_{SD}, \langle S, D, q_n, Id\rangle)\rangle$
2. $\langle \mathsf{req}, S, D, q_n, Id, [X, W], \mathsf{MAC}(K_{SD}, \langle S, D, q_n, Id\rangle)\rangle$
3. $\langle \mathsf{req}, S, D, q_n, Id, [X, W], \mathsf{MAC}(K_{SD}, \langle S, D, q_n, Id, [X, W]\rangle)\rangle$

Figure 1.3: Example of an attack on SRP

a reply message that the intruder forwards to the source. Notice that this attack is possible even if nodes perform secure neighborhood discovery: as X is a neighbor of D, the destination node D accepts $[X, W]$ as a route between him and S.

***Tunneling* attacks.** This attack requires the intruder to control at least two nodes. A *tunneling* attack (or *wormhole attack*) is similar to a replay attack: when the first node receives a request sent by an honest agent, he sends it (over a private channel) to the other node, who processes it as if he had really received it. This can lead other agents to believe that a route going through the dishonest node is short, and so to choose this route to communicate.

Rushing attacks. For this attack, the intruder needs to be faster than the honest agents (see a description and model in [KHG06]). To prevent flooding of the network, in most of the routing protocols, the agents process only the first request that they receive. If the intruder is faster, the probability is higher that the routes found by the protocol go through him.

Intruders in a network can be *passive* or *active*. A passive intruder does not send any message, he only overhears the communications. Passive intruders are considered a threat for secrecy or anonymity, but not for the correct execution of protocols. An active intruder can send messages over the network. In an ad hoc network, the power of the intruder also depends on the number of nodes that he controls. In a wireless network, the intruder is constrained by his location in addition to his computing abilities. An intruder controlling only one node has therefore less power to disrupt the routing protocols than an intruder controlling several nodes across the network. For example, an intruder controlling only one node can not mount tunneling attacks.

1.3 Formal verification

In the previous section, we have seen an attack against SRP, although the authors analyzed their protocol with BAN logic and concluded that it was secure. They showed that, after a successful run of the protocol, the source node S believes that the entire route reply originates from the destination node D. Unfortunately, even though this analysis is sound, it is not sufficient to ensure route correctness([Mar03]).This illustrates how difficult it is to correctly analyze cryptographic protocols. Furthermore, verification has to be performed for each new protocol, and verification by hand is tedious and error prone. Other methods to test whether routing protocols are secure proceed by checking whether they withstand known attacks, but this is not sufficient to prove that they are secure.

It has been recognized that designing a secure protocol is a difficult and error-prone task. Indeed, protocols are very sensitive to small changes in their description and many protocols have been shown to be flawed several years after their publication (and deployment).

Formal methods include techniques that can be used for the specification, development and verification of protocols. Performing appropriate analyses can contribute to the reliability and robustness of software using the protocols. Formal methods make use of a variety of theoretical computer science fundamentals, in particular logic calculi, formal languages, automata theory, and program semantics, but also type systems and algebraic data types.

1.3.1 Symbolic approach

Symbolic models are highly astracted models used to reason about protocols. In symbolic models, the network is represented by a set of agents that can exchange messages. These messages are represented by symbolic terms. Furthermore, in the traditional Dolev-Yao model [DY81], the intruder controls the network: he can overhear, intercept, and forge messages within the constraints of the cryptography. The symbolic models make the hypothesis of *perfect cryptography*: all the cryptographic primitives behave in an ideal way. For instance, it is impossible to decrypt without the corresponding key, or to sign a message without a private key.

A multitude of effective frameworks have been proposed to analyze protocols in a symbolic way. The Paulson inductive model [Pau98] is an algebraic model where a protocol is modeled inductively as a set of traces. A trace is a sequence of communication operations representing an execution of the protocol. Proofs in this model can be generated with the theorem prover Isabelle/HOL [Pau89]. The strand spaces model [THG99] introduces the notion of strands, which represent a sequence of events either legitimate or malicious. A strand space is a collection of strands with links representing causal interaction. The applied pi-calculus [AF01] is an extension of the pi calculus with value passing, primitive functions, and equations among terms. Constraints systems [RT01] represent each protocol execution as a set of constraints which represent the intruder knowledge and the terms the intruder has to be able to build in order to perform an attack.

Formal modeling and analysis techniques are well-adapted for checking correctness of security protocols. Formal methods have for example been successfully used for analyzing authentication or key establishment security protocols. Symbolic methods have been successfully applied to the analysis of security protocols, yielding the discovery of new attacks like the famous man-in-the-middle attack in the Needham-Schroeder public key protocol [Low96] or, more recently, a flaw in Gmail [ACC$^+$08]. While secrecy and authentication properties are undecidable in the general case [DLMS99], many decision procedures have been proposed. For example, secrecy and authentication become NP-complete for a bounded number of sessions [RT01] and B. Blanchet has developed a procedure for security protocols encoded as Horn clauses [Bla01]. This yielded various efficient tools for detecting flaws and proving security (e.g. ProVerif [Bla05] or Avispa [ABB$^+$05]).

Symbolic models are highly abstracted approaches, but there are results that show that the security guarantees they provide are nonetheless reasonable. For instance, let us consider a more precise model, the *computational model*. In this approach, messages are bitstrings and the intruder is a probabilistic polynomial time Turing machine. Results of computational soundness show that it is possible to prove security in the symbolic model and to lift the result to the computational model, under certain conditions. The concept of computational soundness was introduced by M. Abadi and Ph. Rogaway [AR00] ([CKW09] is a recent survey of computational soundness results). In light of these works, symbolic models seem to be a reasonable approach for analyzing protocols in an efficient way while having sufficient security guarantees.

1.3.2 Characteristics of routing protocols

While key-exchange protocols in traditional frameworks are well studied, there are very few attempts to develop formal techniques allowing an automated analysis of secured routing protocols. Up to our knowledge, tools that would allow the security analysis of routing protocols are also missing. Those protocols indeed involve several subtleties that cannot be reflected in existing work, that we will describe now.

Routing protocols involve different elements that distinguish them from other cryptographic protocols. For instance, most cryptographic protocols involve two agents who want to communicate while preserving some privacy or anonymity or other property. They could for example be executed once the routing protocol has been run and has established a route between two nodes. We give here the main characterisitcs of routing protocols that have to be accurately modeled in order to formally verify them.

Number of agents involved. It is impossible to know in advance how many devices will be involved in the execution of a route discovery protocol. In fact, route discovery can involve an unbounded number of nodes. The first phase of route discovery potentially involves all the nodes of the network. Intuitively, when a node S wants to interact with another node D and he has no idea of how to reach D he will send messages everywhere on the network in order to reach him. Furthermore, the number of nodes in the network is not fixed. By nature, anyone can participate in an ad hoc network. The only requirement is to emit and receive messages via the same wireless medium as the other nodes. The infrastructure is much more flexible than in a wired setting. Nodes can appear, disappear, move. It is thus impossible to know in advance how many devices will be involved in the execution of a route discovery protocol, even by fixing the source and the destination. Fixing the network is a first step towards knowing which nodes will be involved. We believe that considering a truly unbounded number of participants would make any reachability problem, and thus basic security properties, undecidable, as it is the case for an unbounded number of sessions in traditional protocols. However, it is not clear whether some simple conditions could circumvent this problem. Unfortunately, we do not deal with the case of an unknown number of participants, but we deal with an unbounded number of nodes in the network (with a bounded number of nodes in the network that actively participate in the protocol execution).

Network topology. The underlying network topology is crucial to define who can receive the messages sent by a node. Moreover, the intruder is localized to some specific nodes (possibly several). The natural way to model an ad hoc network is to use a notion of graph where there is an edge between two nodes if they can communicate directly. For a protocol used in a wired or a fixed setting, there also exists an underlying graph, but it does not play a role in the interaction. In a traditional malicious setting, the intruder is assumed to control the network. He can receive all the messages that are sent between the participants, intercept them, forge and send messages in the name of an honest agent. In a wireless network where communication can only occur node to node, it is not realistic to assume that an intruder can receive all messages. The intruder has to be located somewhere, and can thus only receive the messages sent by his immediate neighbors. Similarly, he can only send messages directly to his neighbors. The physical aspect of the equipment used by the intruder can play a role. If he has the use of an antenna that can send messages in a given direction, it is possible for him to send messages to a given agent. Otherwise, he has to broadcast his messages, as do the honest agents. The broadcast nature of transmission means that the intruder can no longer intercept messages. It is possible to jam the communications in a physical way, but this would block any message during a certain amount of time without discrimination. We do not consider denial of service attacks. We do not give the intruder the ability to jam communications either. But we give him the ability to send messages to only one of his neighbors. This only adds to his power, so we do not miss attacks by doing such a thing. We also let the intruder control several nodes of the network.

Specific data structure: list. When studying source routing protocol, we have to consider a data structure not usually treated in verification tools: the list. Indeed, the route discovered is represented by a list of nodes. Intuitively, during the first phase of the discovery process, also known as request phase, the list is built incrementally, each node adds its name to the route field in the request message, and when the message reaches the destination, the reply phase begins. During the reply phase, he only has to send a message back to the source, containing the fully built list, possibly protected so as not to be modified. The source needs this list to route the normal data traffic between him and the destination. So messages contain lists, whose size are not known in advance. We model lists and we show that not knowing their size beforehand does not prevent us from establishing some decidability results.

Security Properties. What seems to be a fundamental property is that when a routing protocol discovers a route, this route matches a real path in the network. This property can not be reduced to an authentication or security property, due in particular to the fact that it is a graph property. In the existing frameworks, this property can therefore not be defined. We model this property by using a logic that reasons about lists. Some routing protocols aim to ensure other security properties about the route, for instance that the route discovered by the protocol is as short as possible, or that the route does not go through any malicious node, or yet that the protocol always discover a route when a path exists in the graph . . . All these properties are properties of the route, which means properties of a list in a source routing protocol. They could probably also be modeled similarly to route correctness. However, in table based routing protocols, the same properties are more complex to express, as the information is scattered across various routing tables in the network. Yet other routing protocols aim at providing anonymity, which is a property that we did not consider.

Neighbor tests. As we have seen after describing the DSR protocol, a very straightforward attack on a routing protocol designed to be used in a friendly environment is to send a forged message in the name of the destination, which could be situated at the other end of the network. To protect against these basic attacks, nodes have to check, when they receive a message supposedly from an agent A, that this agent is within communication range, or in other words in our model, is a neighbor. The neighbor discovery protocol must thus also be secured, if we want to make checks that are useful. We consider that a secure neighborhood discovery protocol has been used, and that each node can check whether a node is his neighbor. Another test involving neighbors can be performed on lists. Typically, during a reply phase, a list of nodes is forwarded, and this list is supposed to represent a valid route. Nodes forwarding this reply message can check whether the list contained in the route field is locally correct by making sure that their name is in the list and appears between the names of two neighbors. Intuitively this test is also essential, as the unprotected route is easily modifiable.

Recursivity. In the EndairA example, we can see that to protect the route during the reply phase against malicious tampering, each node performs an operation on the same field. They each sign the list, and build a list out of these signatures. To check that the list has not been tampered with, the source then has to check that the result of this construction is valid. This means checking that a recursively built list is valid. So, in order to model protocols that use these recursively built lists to authenticate the route, we have to be able to deal with a form of recursivity.

1.3.3 Verification of secure routing protocols

Recently, several results have been proposed for studying routing protocols. For example, S. Yang and J. Baras [YB03] provide a first symbolic model for routing protocols based on strand spaces, modeling the network topology. They implement a semi-decision procedure to search for attacks and find an attack on AODV [PBR99], a routing protocol (built for friendly environments) that does not include cryptography. Their approach however does not apply to routing protocols using cryptographic primitives for securing communications.

Case studies

Several case studies of important secured routing protocols have been performed. J. Godskesen [God06] provides an analysis of a simplified version of the ARAN [SDL$^+$02] protocol with ProVerif, for a given configuration, and captures a relay attack. J. Marshall [Mar03] uses Cryptographic Protocol Analysis Language Evaluation System (CPAL-ES) to specify the SRP protocol and analyze it. The encoding of SRP is done on a precise fixed topology, without broadcast, and a replay attack is retrieved.

T. Andel [And07] uses model checking in his PhD thesis to analyze the ARAN protocol in the SPIN tool, for a fixed topology. In order to be able to analyze a priori unknown topologies, the authors propose a reduction of the search space by establishing equivalence between different topologies, and showing that it is enough to test the smallest topology in an equivalence class to decide security.

General framework

While these last results focus on particular routing protocols, some frameworks have been proposed to model wireless communication and/or routing protocols in a more generic way. L. Buttyán and I. Vajda [BV04] provide a model for routing protocols, in a cryptographic setting. Their model enables them to find attacks on SRP and Ariadne [HPJ05]. They provide a security proof (by hand) for a fixed protocol they propose, endairA. G. Àcs, L. Buttyàn and I. Vajda then develop their framework for distance vector routing protocols [ABV05], analyzing SAODV [ZA02] and ARAN. They also apply their framework to sensor networks [ABV06], analyzing TinyOS [PSW$^+$02].

S. Nanz and C. Hankin [NH06] propose a process calculus to model the network topology and broadcast communications, extending the Calculus of Broadcasting System. They analyze scenarios with special topologies and attacker configuration by computing an over-approximation of reachable states. They develop a static analysis based on a flow logic. Their analysis is safe in the sense that the discovered attacks correspond to real ones. They also propose a decision procedure but for an intruder that is already specified by the user. This allows to check security only against fixed, known in advance scenarios.

Up to our knowledge, we are the first to provide decidability or complexity result for routing protocols, for arbitrary intruders and network topologies.

1.4 Contributions

Ad hoc routing protocols have several particularities that distinguish them from traditional key-exchange protocols. Among them are predominant the network topology, the neighborhood tests and security properties, as well as recursivity. We propose a model for ad hoc routing protocols that takes into account the topology of the network and the specific broadcast primitive. We then analyze separately protocols that perform neighborhood tests and protocols that make use of recursivity. In order to analyze both types of protocols, we use constraint systems [MS01, CLCZ10]. We extend constraint systems with the primitives useful to deal with routing protocols and recursivity, such as lists. Furthermore, we also revisit them to deal with an infinite number of nodes. Thanks to the representation of protocol execution using constraint systems, we obtain decidability results for source routing protocols with their specific security properties on the one hand, and for protocols with recursive tests on the other hand.

Modeling routing protocols

We propose a calculus, inspired from CBS# [NH06], which allows ad hoc networks and their security properties to be formally described and analyzed. As for standard symbolic models for security protocols, we model cryptography as a black box (the perfect cryptography assumption), thus the attacker cannot break cryptography, e.g. decrypt a message without having the appropriate decryption key. To take the features of ad hoc routing protocols into account, we first propose a logic to express the neighbor tests performed by the nodes at each step. There are also some implications for the attacker model. Indeed, in most existing formal approaches, the intruder model consists in the Dolev Yao attacker that controls the entire network. We have explained why this attacker model is too strong in the context of routing protocols: the topology of the network plays a crucial role in the execution of the protocol and the possible communications. Considering an intruder with a total control over the network and not localized in one particular node would lead to a number of false attacks. Our model reflects the fact that a malicious node can interfere directly only with his neighbors.

In order to analyze protocols such as EndairA, we have to be able to deal with recursivity. These protocols share a way of authenticating the route during the reply phase of the route discovery. Each of the nodes on the route compute a signature, and the source has to check that the list of signatures has been properly built.

Recursivity quickly yields undecidability, as even a single input/output step in a protocol may reveal complex information, as soon as it involves a recursive computation [KW04]. In order to circumvent this, we consider protocols that perform standard input/output actions (modeled using usual pattern matching) but that are allowed to perform *recursive tests* such as checking the validity of a route or the validity of a chain of certificates. Indeed, secure routing protocols use recursivity only for performing sanity checks at some steps of the protocol. It is also the case of other protocols such as distributed right delegation, and PKI certification paths.

Analyzing routing protocols

Our formal model represents all possible executions against an adversary that controls some of the nodes and acts maliciously in these nodes by sending any message that he can construct. Our model is thus infinitely branching. As a first step towards automation, we provide an alternative symbolic semantics, based on constraint systems and we show its correctness and completeness w.r.t. the concrete semantics. This result holds for arbitrary processes (possibly with replication) and for any set of primitives.

We provide two NP decision procedures for analyzing routing protocols for a bounded number of sessions and for a large set of standard primitives. For a fixed set of roles and sessions, we show that it is possible to discover whether there exists a network topology and a malicious behavior of some nodes that yield an attack. We can also decide whether there exists an attack, for a network topology chosen by the user. These two procedures hold for any property that can be expressed in our logic, which includes classical properties such as secrecy as well as properties more specific to routing protocols such as route validity.

Analyzing protocols with recursive tests

For checking security of protocols with recursive tests (for a bounded number of sessions), we reuse the setting of constraint systems and add tests of membership to recursive languages. We propose (NPTIME) decision procedures for two classes of recursive languages (used for tests): *link-based recursive languages* and *mapping-based languages*. A link-based recursive language contains chains of links where consecutive links have to satisfy a given relation. A typical example is X.509 public key certificates as defined in [HFP98] that consist in a chain of signatures of the form:

$$[[\![\langle A_1, \mathsf{pub}(A_1)\rangle]\!]_{\mathsf{sk}(A_2)}; [\![\langle A_2, \mathsf{pub}(A_2)\rangle]\!]_{\mathsf{sk}(A_3)}; \cdots ; [\![\langle A_n, \mathsf{pub}(A_n)\rangle]\!]_{\mathsf{sk}(S)}].$$

A mapping-based language contains lists that are based on a list of names (typically names of agents involved in the protocol session) and are uniquely defined by it. Typical examples can be found in the context of routing protocols, when nodes check for the validity of the route. For example, in the endairA protocol [BV04], a route from the source A_0 to the destination A_n is represented by a list $l_{route} = [A_n; \ldots; A_1]$. This list is accepted by the source node A_0 only if the received message contains a list of signatures authenticating it of the form:

$$[\underbrace{[\![\langle A_n, A_0, l_{route}, [sig_2; \ldots; sig_n]\rangle]\!]_{\mathsf{sk}(A_1)}}_{sig_1}; \underbrace{[\![\langle A_n, A_0, l_{route}, [sig_3; \ldots; sig_n]\rangle]\!]_{\mathsf{sk}(A_2)}}_{sig_2}; \ldots$$

$$\ldots; \underbrace{[\![\langle A_n, A_0, l_{route}, [sig_n]\rangle]\!]_{\mathsf{sk}(A_{n-1})}}_{sig_{n-1}}; \underbrace{[\![\langle A_n, A_0, l_{route}, []\rangle]\!]_{\mathsf{sk}(A_n)}}_{sig_n}].$$

Note that a link $[\![\langle A_n, A_0, l_{route}, [sig_i; \ldots; sig_n]\rangle]\!]_{\mathsf{sk}(A_i)}$ depends on the list l_{route}, on its i-th element and on the following links in the list.

Some of those results were published in [ACD10], with a preliminary version in [ACD09]. A journal version has been submitted to the special issue of Information and Computation on Security and Rewriting techniques. The results regarding recursivity were published in [ACD11].

1.5 Outline of the Dissertation

In order to provide decision procedures for routing protocols, we reuse the setting of constraint systems. We first show in Chapter 2 how to transform the constraint systems corresponding to the execution of those protocols into *solved* constraint systems. We define constraint systems and the simplification rules that are used to obtain solved constraint systems. Our constraint systems give the intruder the power of generating any IP address. To achieve that, we assume that he has at his disposal a potentially infinite number of names. In the context of routing protocols, these names represent IP addresses. We also revisit the procedure of [CLCZ10] for solving constraint systems and obtain a complete symbolic representation of the knowledge of the attacker, in the spirit of the characterization obtained in [AC06] in the passive case (with no active attacker). We show that all the terms that the intruder can build are obtained by combining terms from a particular set of terms. We give a characterization of the solutions to these solved constraint systems.

In Chapter 3, we propose a way to model and analyze ad hoc routing protocols. Section 3.1 presents our formal model for routing protocols, using a process calculus with an underlying graph. It is illustrated with the modeling of the SRP protocol. We explain how to abstract some parts in the execution of the protocol in order to get a finite number of representations of the possible runs. This allows us to provide two NP decision procedures for analyzing routing protocols for a bounded number of sessions, for a fixed network and an unknown one.

In Chapter 4, we show that it is possible to analyze protocols with recursive tests and obtain decidability results. We use the results obtained in Chapter 2, and we add tests of membership to recursive languages. We provide two NPTIME decision procedures for two classes of recursive languages that encompass most of recursive tests involved in secured routing protocols.

In Chapter 5, we conclude by discussing possible further works.

Chapter 2

Constraint systems

Constraint systems are quite common in modeling security protocols in the case of an active intruder. A constraint system represents the execution of a protocol for a finite number of sessions and a fixed interleaving. Symbolic constraint systems are thus well suited to express a reachability (e.g., secrecy) property. J. Millen and V. Shmatikov first introduced constraint systems in [MS01] to solve a reachability problem for cryptographic protocols. M. Rusinowitch and M. Turuani showed that protocol insecurity is NP-complete in the case of a finite number of sessions by establishing a small attack property [RT01]. Both of these two first approaches consider rather basic cryptographic primitives. The notion of constraint systems has since been used in several works, with decidability results for different primitives, such as exclusive or operator [CS03], modular exponentiation [CKRT03], monoidal theories [DLLT08]. In [CS03], and later in [CLCZ10], a more generic approach is provided to decide general security properties. It consists in transforming any constraint system into simpler constraint systems. This procedure preserves all the solutions of the initial constraint system. In this chapter, we follow a similar approach. We consider a large signature, encompassing symmetric and asymmetric encryption, signature, hashes, and lists. Lists are particularly useful for modeling routing protocols (see Chapter 3). Moreover, we provide the intruder with an infinite set of names that he can use however he wants. This is also important in the context of routing protocols to model an arbitrary number of nodes.

We also revisit the procedure of [CLCZ10] for solving constraint systems and obtain a complete symbolic representation of the knowledge of the attacker, in the spirit of the characterization obtained in [AC06] in the passive case (with no active attacker).

We work in symbolic models, where messages are represented by elements in some term algebra. In the next chapters, we also make use of this model, that we introduce in Section 2.1. We define constraint systems in §2.2.1 and give the simplification rules associated in §2.2.2. Those simplification rules are sound, complete and terminate in polynomial time (proofs can be found in §2.3.1, §2.3.3 and §2.3.2 respectively). Our procedure furthermore allows us to reduce the search for solutions to a specific form of solutions (non-confusing solutions, defined page 23). We show in Section 2.4 that, when considering these solutions, any term of the intruder knowledge may be obtained by composition only from a clearly defined set of terms.

2.1 Model for security protocols

In this section, we first introduce term algebra, our model for messages, in §2.1.1. We explain how the power of the intruder is modeled through deduction in §2.1.2. We then illustrate in §2.1.3 how protocol execution can be represented using constraint systems.

2.1.1 Messages

In our model, messages are represented using a term algebra. Cryptographic primitives are represented by function symbols. For instance, symmetric encryption will be represented by the function symbol senc. Hence, a message m encrypted with a symmetric key k will be represented by $\mathsf{senc}(m, k)$. We work under the perfect cryptography assumption: an encrypted message $\mathsf{senc}(m, k)$ can only decrypted by somebody who knows the value of the key k used to perform the encryption.

We consider a *signature* $(\mathcal{S}, \mathcal{F})$ consisting in a set of *sorts* $\mathcal{S}$ and a set of *function symbols* $\mathcal{F}$. Each function symbol f is associated with an arity $ar(f)$, which is a mapping from $\mathcal{F}$ to $\mathcal{S}^* \times \mathcal{S}$. We write $ar(f) = s_1 \times \ldots \times s_k \to s$ (with $s_1, \ldots, s_k, s \in \mathcal{S}$). Furthermore, we distinguish a set $\mathcal{F}_{priv}$ of functions symbols of $\mathcal{F}$ that will contain private function symbols, i.e. functions that the intruder can not use. These functions typically encompass the generation of keys.

We consider an infinite set of *variables* $\mathcal{X}$ and an infinite set of *names* $\mathcal{N}$ that typically represent nonces or agent names. We assume that names and variables are given with sorts. The set of *terms of sort s* is defined inductively by:

$$\begin{array}{lllll}
t & ::= & & \text{term of sort } s \\
& | & x & \text{variable } x \text{ of sort } s \\
& | & a & \text{name } a \text{ of sort } s \\
& | & f(t_1, \ldots, t_k) & \text{application of symbol } f \in \mathcal{F} \text{ such that} \\
& & & ar(f) = s_1 \times \ldots \times s_k \to s \text{ and each } t_i \text{ is a term of sort } s_i
\end{array}$$

Sorts will mostly be left unspecified in this chapter, except in specific examples.

We consider an infinite set of *names* $\mathcal{N}$ having Base sort. These names typically represent constants, nonces, symmetric keys, or agent names. We write $vars(t)$ for the set of variables occurring in a term t. The term t is said to be a *ground* term if $vars(t) = \emptyset$.

We write $st(t)$ for the set of syntactic *subterms* of a term t. This notion is extended as expected to sets of terms. If S is a set, we denote by $\#S$ the cardinal of S. Let u be a term, u can be represented in different ways. In general, it is represented by a tree. We write $\|u\|$ for the size of u, i.e. the size of the tree representing u. We can also represent u as a directed acyclic graph where subgraphs are all distinct. This is called the *dag representation* of u. We denote by $\|u\|_{dag}$ the size of the dag representation of u, that is the number of distinct subterms of u.

Substitutions are written $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ with $dom(\sigma) = \{x_1, \ldots, x_n\}$. They are assumed to be well-sorted substitutions, that is each t_i is of the same sort as x_i. Such a substitution σ is *ground* if all the t_i are ground terms. The application of a substitution σ to a term u is written $u\sigma$ or $\sigma(u)$. A most general unifier of terms u_1 and u_2 is a substitution (when it exists) denoted by $mgu(u_1, u_2)$.

Example 2.1.1. *In our examples, we will consider the specific signature* $(\mathcal{S}, \mathcal{F})$ *defined by* $\mathcal{S} = \{\mathit{Msg}, \mathit{Base}, \mathit{List}\}$ *and* $\mathcal{F} = \{\mathsf{senc}, \mathsf{aenc}, [\![\_]\!]\_, \langle\_,\_\rangle, \mathsf{h}, \mathsf{hmac}, ::, [], \mathsf{pub}, \mathsf{priv}, \mathsf{vk}, \mathsf{sk}\}$ *with corresponding arities:*

- $ar(\mathsf{f}) = \mathit{Msg} \times \mathit{Msg} \to \mathit{Msg}$ *for* $\mathsf{f} \in \{\mathsf{senc}, \mathsf{aenc}, [\![\_]\!]_\_, \langle\_,\_\rangle, \mathsf{hmac}\}$,
- $ar(\mathsf{h}) = \mathit{Msg} \to \mathit{Msg}$,
- $ar(::) = \mathit{Msg} \times \mathit{List} \to \mathit{List}$, *and* $ar([]) = \mathit{List}$,
- $ar(\mathsf{f}) = \mathit{Base} \to \mathit{Msg}$ *for* $\mathsf{f} \in \mathcal{F}_{priv} = \{\mathsf{pub}, \mathsf{priv}, \mathsf{vk}, \mathsf{sk}\}$.

The sort Msg *subsumes the two other sorts. The symbol* $\langle\rangle$ *represents the pairing function,* $::$ *is the list constructor, and* $[]$ *represents the empty list. For the sake of clarity, we write* $\langle u_1, u_2, u_3\rangle$ *for the term* $\langle u_1, \langle u_2, u_3\rangle\rangle$*, and* $[u_1; u_2; u_3]$ *for* $u_1 :: (u_2 :: (u_3 :: []))$*. The terms* $\mathsf{pub}(A)$ *and* $\mathsf{priv}(A)$ *represent respectively the public and private keys associated to an agent* A*, whereas the terms* $\mathsf{sk}(A)$ *and* $\mathsf{vk}(A)$ *represent respectively the signature and verification keys associated to an agent* A*. The function symbol* senc *(resp.* aenc*) is used to model symmetric (resp. asymmetric) encryption whereas the term* $[\![m]\!]_{\mathsf{sk}(A)}$ *represents the message* m *signed by the agent* A*.*

2.1.2 Intruder Capabilities

We model the ability of the intruder by a deduction relation $\vdash \subseteq 2^{\mathsf{terms}} \times \mathsf{terms}$. The relation $T \vdash t$ represents the fact that the term t is computable from the set of terms T. It is typically defined through a deduction system.

The deduction system we use to model the ability of the intruder is described in Figure 2.1:

$$\frac{u_1 \ \ldots \ u_n}{\mathsf{f}(u_1, \ldots, u_n)}\ \mathsf{f} \in \mathcal{F} \smallsetminus \mathcal{F}_{priv} \qquad \frac{\langle u_1, u_2\rangle}{u_i}\ i \in \{1,2\} \qquad \frac{u_1 :: u_2}{u_i}\ i \in \{1,2\}$$

$$\frac{\mathsf{senc}(u_1, u_2) \quad u_2}{u_1} \qquad \frac{\mathsf{aenc}(u_1, \mathsf{pub}(u_2)) \quad \mathsf{priv}(u_2)}{u_1}$$

Figure 2.1: Deduction system

The first inference rule describes the *composition rules.* The remaining inference rules describe the *decomposition* rules. Intuitively, these deduction rules say that an intruder can compose messages by pairing, building lists, encrypting and signing messages provided he has the corresponding keys. Conversely, he can retrieve the components of a pair or a list, and he can also decompose messages by decrypting provided he has the decryption keys. However, he can not use the functions included in the specific set $\mathcal{F}_{priv}$ of private functions. For instance, he may not be able to create secret keys ($\mathcal{F}_{priv} = \{\mathsf{pub}, \mathsf{priv}, \mathsf{vk}, \mathsf{sk}\}$ in our running example). The intruder is also able to *verify* whether a signature $[\![m]\!]_{\mathsf{sk}(a)}$ and a message m match (provided he has the verification key $\mathsf{vk}(a)$), but this operation does not allow him to learn any new message. For this reason, this capability is not represented in the deduction system. We also consider an optional rule

$$\frac{[\![u_1]\!]_{\mathsf{sk}(u_2)}}{u_1}$$

that expresses that an intruder can retrieve the whole message from its signature. This property may or may not hold depending on the signature scheme, and that is why this rule is optional. Our results hold in both cases (that is, when the deduction relation $\vdash$ is defined with or without this rule).

The deduction system allows us to define formally which terms the intruder can deduce from his knowledge. We define proof trees, that are intuitively trees where every intermediate node is an instance of one of the rules of the deduction system. If the leaves of a proof tree are terms of the knowledge of the intruder, then the root is a term that the intruder is able to create.

Definition 2.1.1 (proof tree). *A* proof tree *is a tree whose nodes are labeled by terms and defined recursively in the following way: trees with only one node are proof trees and, if $\pi_1, \dots, \pi_n$ are proof trees whose respective roots are $u_1, \dots, u_n$ and if*

$$\frac{t_1 \dots t_n}{t}$$

is an inference rule such that for some well-sorted substitution σ, $t\sigma = u, t_1\sigma = u_1, \dots, t_n\sigma = u_n$, then the following tree is a proof tree:

$$\frac{\pi_1 \dots \pi_n}{u}$$

Example 2.1.2. *The following tree π is a proof tree:*

$$\frac{\langle N_B, [\![\langle B, \mathsf{pub}(B)\rangle]\!]_{\mathsf{sk}(S)}\rangle}{[\![\langle B, \mathsf{pub}(B)\rangle]\!]_{\mathsf{sk}(S)}}$$

According to the definition, the tree consisting of the node $\langle N_B, [\![\langle B, \mathsf{pub}(B)\rangle]\!]_{\mathsf{sk}(S)}\rangle$ is a proof tree. Furthermore, consider the inference rule given by

$$\frac{\langle u_1, u_2\rangle}{u_2}$$

and the substitution $\sigma = \{u_1 \mapsto N_B, u_2 \mapsto [\![\langle B, \mathsf{pub}(B)\rangle]\!]_{\mathsf{sk}(S)}\}$ to complete the proof that π is a proof tree.

Definition 2.1.2 (deducible term). *A term u is* deducible *from a set of terms T, denoted by $T \vdash u$, if there exists a* proof tree *whose root is labeled with u and whose leaves are labeled by terms in T.*

Example 2.1.3. *The proof tree π as defined in Example 2.1.2 has its root labeled by a term $u = [\![\langle B, \mathsf{pub}(B)\rangle]\!]_{\mathsf{sk}(S)}$. Furthermore, its leaves are labeled by terms in*

$$T_2 = T_1 \cup \{\langle N_B, [\![\langle B, \mathsf{pub}(B)\rangle]\!]_{\mathsf{sk}(S)}\}\rangle$$

so we deduce that $T_2 \vdash u$, i.e. u is deducible from T_2.

2.1.3 From protocols to constraint systems

Constraint systems were first introduced in [MS01] in order to model security protocols. They are used to specify trace-based property, *e.g.* secrecy preservation of security protocols under a particular, finite scenario. The model of constraint system we use is close to the model in [CLCZ10], with slight differences in the definitions of constraint systems and the way we simplify them. But before defining constraint systems, we motivate their use by showing how they can represent the execution of a protocol in the presence of an active intruder. We will use an example in order to illustrate this. Note that in Chapter 3, we describe precisely how the execution of a protocol modeled using our process calculus may be represented by constraint systems.

Example 2.1.4. *We consider part of the TLS handshake protocol [DR08], designed to exchange enough information between an agent A (the client) and an agent B (the server) to compute a shared key while authenticating the agent B:*

$$\begin{array}{ll} A \to B: & N_A \\ B \to A: & \langle N_B, [\![\langle B, \mathsf{pub}(B)\rangle]\!]_{\mathsf{sk}(S)}\rangle \\ A \to B: & \mathsf{aenc}(K_A, \mathsf{pub}(B)) \end{array}$$

The agent A sends B a fresh nonce N_A. Upon receiving this message, B generates a fresh nonce N_B and sends it to A together with a certificate $[\![B, \mathsf{pub}(B)]\!]_{\mathsf{sk}(S)}$ signed by a trusted third party S. The agent A checks that the certificate is valid. Then, it generates a fresh nonce K_A and uses the public key $\mathsf{pub}(B)$ to encrypt this nonce before sending it to B. The full protocol makes use of the nonces exchanged between the agents to generate a session key. However, we do not model the full protocol, as the part described here is enough to illustrate how protocols are modeled in the symbolic setting, and to explain the use of constraint systems.

A passive intruder eavesdropping on the communications between A and B would thus obtain the following sequence of messages at the end of the session:

$$N_A, \langle N_B, [\![\langle B, \mathsf{pub}(B)\rangle]\!]_{\mathsf{sk}(S)}\rangle, \mathsf{aenc}(K_A, \mathsf{pub}(B)).$$

We want to model an active intruder. In the Dolev-Yao setting [DY81] that we have adopted, such an intruder is given full control of the network: not only can he overhear messages, but he can also intercept and modify them. This also gives him the power to choose the interleaving of the messages.

To model that capacity, we consider that every agent involved in the protocol can only communicate with the intruder. For example, the first step of the TLS handshake protocol would intuitively be represented in the following way:

$$A \overset{N_A}{\to} I \overset{t_1}{\to} B$$

where t_1 is a term that the intruder I can build. During this step, the intruder overhears the terms N_A. Formally, if the initial knowledge of the intruder is represented by a set of terms T_0, t_1 is a term that can be deduced from $T_1 = T_0 \cup \{N_A\}$. The deducibility constraint associated is $T_1 \overset{?}{\vdash} x_1$ and will be formally defined later.

Now, take a look at the second step:

$$A \overset{t_2}{\leftarrow} I \overset{m_2}{\leftarrow} B$$

where $m_2 = \langle N_B, [\![B, \mathsf{pub}(B)]\!]_{\mathsf{sk}(S)}\rangle$. Notice that the knowledge of the intruder has grown, since he receives the message m_2. Thus the condition on t_2 is of the form $T_2 \vdash t_2$ where $T_2 = T_1 \cup \{m_2\}$. Furthermore, the agent A expects a message following the same pattern as $\langle N_B, [\![\langle B, \mathsf{pub}(B)\rangle]\!]_{\mathsf{sk}(S)}\rangle$ where S is a known in advance, trusted third party. Thus, the term t_2 has to be a term of the form $\langle x_2, [\![\langle y_2, z_2\rangle]\!]_{\mathsf{sk}(S)}\rangle$, where $\mathsf{pub}(y_2)$ is instantiated by the public key of B in the normal run of the protocol.

Finally, the third step

$$A \overset{m_3}{\to} I \overset{t_3}{\to} B$$

where $m_3 = \mathsf{aenc}(K_A, z_2)$, yields the following deducibility constraint: $T_3 \overset{?}{\vdash} \mathsf{aenc}(x_3, \mathsf{pub}(B))$ where $T_3 = T_2 \cup \{m_3\}$.

2.2 Constraint systems

In this section, we introduce formally constraint systems. We first give in §2.2.1 the formal definition of our constraint systems, and then we describe in §2.2.2 the simplifications rules that will allow us to always consider a simple form of constraint systems.

2.2.1 Defining constraint systems

To enforce the intruder capabilities, we assume that the intruder has at his disposal an infinite set of names that he might use at his will to mount attacks. In the general case, this ability is not used. However, our modeling of routing protocols, as exposed in Chapter 3, may yield disequality constraints (see Definition 3.2.1). In order to be able to satisfy these specific constraints, the intruder may need a potentially unbounded number of distinct names.

Definition 2.2.1 (constraint system)**.** *A* constraint system *is a pair $(\mathcal{C}, \mathcal{I})$ such that $\mathcal{I}$ is a non empty (and possibly infinite) set of names, and $\mathcal{C}$ is either $\perp$ or a finite conjunction*

$$T_1 \overset{?}{\vdash} u_1 \wedge \cdots \wedge T_n \overset{?}{\vdash} u_n$$

of expressions $T_i \overset{?}{\vdash} u_i$ called constraints*. Each T_i is a finite set of terms called the* left-hand side *of the constraint. Each u_i is a term, called the* right-hand side *of the constraint. The constraints are ordered such that they satisfy two conditions:*

- monotonicity: *$T_i \subseteq T_k$ for every i, k such that $1 \leq i < k \leq n$;*
- origination: *if $x \in vars(T_i)$ for some i then there exists $j < i$ such that $x \in vars(u_j)$.*

Moreover, $st(\mathcal{C}) \cap \mathcal{I} = \emptyset$.

The monotonicity condition states that the intruder knowledge is always increasing. The origination condition in Definition 2.2.1 states that each time a new variable is introduced, it first occurs in some right-hand side. The left-hand side of a constraint system usually represents the messages sent on the network, while the right-hand side represents the message expected by the party. The set $\mathcal{I}$ represents names that only the intruder knows, so they are not used in the messages exchanged on the network, as the condition $st(\mathcal{C}) \cap \mathcal{I} = \emptyset$ clearly states: the set of syntactic subterms of $\mathcal{C}$ has no term in common with $\mathcal{I}$.

Definition 2.2.2 (right-hand and left-hand sides)**.** *Let $(\mathcal{C}, \mathcal{I})$ be a constraint system. We denote by $\mathsf{rhs}(\mathcal{C})$ (respectively, $\mathsf{lhs}(\mathcal{C})$) the set of right-hand side terms (respectively, left-hand side sets of terms) of $\mathcal{C}$. Formally, $\mathsf{rhs}(\mathcal{C})$ and $\mathsf{lhs}(\mathcal{C})$ are defined recursively in the following way:*

$$\begin{array}{ll} \mathsf{rhs}(\perp) = \emptyset & \mathsf{lhs}(\perp) = \emptyset \\ \mathsf{rhs}(\mathcal{C} \wedge T \overset{?}{\vdash} u) = \mathsf{rhs}(\mathcal{C}) \cup \{u\} & \mathsf{lhs}(\mathcal{C} \wedge T \overset{?}{\vdash} u) = \mathsf{lhs}(\mathcal{C}) \cup \{T\} \end{array}$$

The origination property ensures that variables are always introduced in the right-hand side of a deduction constraint, which is always the case when modeling protocols. Formally, if $(\mathcal{C}, \mathcal{I})$ is a constraint system, then $var(\mathsf{rhs}(\mathcal{C})) = var(\mathcal{C})$.

Example 2.2.1. *An execution of the TLS handshake protocol as defined previously, in Example 2.1.4, can be represented by the constraint system* $(\mathcal{C}, \mathcal{I})$ *where*

$$\mathcal{C} = \begin{cases} T_1 \overset{def}{=} T_0 \cup \{N_A\} & \overset{?}{\vdash} & x_1 \\ T_2 \overset{def}{=} T_1 \cup \{\langle N_B, [\![\langle B, \mathsf{pub}(B)\rangle]\!]_{\mathsf{sk}(S)}\rangle\} & \overset{?}{\vdash} & \langle x_2, [\![\langle y_2, z_2\rangle]\!]_{\mathsf{sk}(S)}\rangle \\ T_3 \overset{def}{=} T_2 \cup \{\mathsf{aenc}(K_A, z_2)\} & \overset{?}{\vdash} & \mathsf{aenc}(x_3, \mathsf{pub}(B)) \end{cases}$$

where $T_0 = \{\mathsf{pub}(B), [\![\langle B, \mathsf{pub}(C)\rangle]\!]_{\mathsf{sk}(S)}\}$ *is a set of terms representing the initial knowledge of the intruder (here, we assume he is in possession of a false certificate), and* $\mathcal{I} = \{n_1, n_2, \dots\}$ *is a set of names disjoint from* $st(\mathcal{C})$. $\mathcal{I}$ *represents names that the intruder can generate and use at his will.*

Definition 2.2.3 ((non-confusing) solution). *Let* $(\mathcal{C}, \mathcal{I})$ *be a constraint system where* $\mathcal{C} = \bigwedge_{i=1}^{n} T_i \overset{?}{\vdash} u_i$. *A* solution *of* $(\mathcal{C}, \mathcal{I})$ *is a well-sorted ground substitution* θ *whose domain is* $vars(\mathcal{C})$ *such that* $T_i\theta \cup \mathcal{I} \vdash u_i\theta$ *for every* $i \in \{1, \dots, n\}$. *The empty constraint system is always satisfiable whereas* $(\bot, \mathcal{I})$ *denotes an unsatisfiable constraint system. Furthermore, we say that* θ *is* non-confusing *for* $(\mathcal{C}, \mathcal{I})$ *if* $t_1 = t_2$ *for any* $t_1, t_2 \in st(T_n)$ *such that* $t_1\theta = t_2\theta$.

In other words, non-confusing solutions do not map two distinct subterms of a left-hand side of the constraint system to the same term. We will show that we can restrict ourselves to consider this particular case of solutions when all possible equalities have already been guessed.

Example 2.2.2. *The substitution* $\theta = \{x_1 \mapsto N_A, x_2 \mapsto N_B, y_2 \mapsto B, z_2 \mapsto \mathsf{pub}(C), x_3 \mapsto n_1\}$ *is a solution of* $(\mathcal{C}, \mathcal{I})$. *But it is confusing since* $y_2, B \in st(T_3)$, $y_2 \neq B$ *and* $y_2\theta = B\theta (= B)$.

Notice that θ is a solution of the constraint system $((\mathcal{C} \wedge T \overset{?}{\vdash} u \wedge T \overset{?}{\vdash} u), \mathcal{I})$ if and only if θ is a solution of the constraint system $(\mathcal{C} \wedge T \overset{?}{\vdash} u, \mathcal{I})$. We will thus only consider constraint systems whose constraints are all distinct.

2.2.2 Simplifying constraint systems

We will use simplification rules in order to reduce solving constraint systems to solving simpler constraint systems that we call *solved*, as is done in [CLCZ10]. Our result follows this work closely but our simplification rules are slightly different in order to obtain a nice characterization of solutions by only considering non-confusing ones.

Definition 2.2.4 (solved form). *A constraint system is* solved *if it is* $(\bot, \mathcal{I})$ *or each of its constraints is of the form* $T \overset{?}{\vdash} x$ *where* x *is a variable.*

Solved constraints are especially easy to solve since variables can be instantiated by any term of the same sort.

The *simplification rules* for deducibility constraints we consider are defined in Figure 2.2.

$$\begin{array}{llll}
\mathsf{R_{ax}}: & (\mathcal{C} \wedge T \overset{?}{\vdash} u, \mathcal{I}) & \rightsquigarrow & (\mathcal{C}, \mathcal{I}) \qquad \text{if } T \cup \{x \mid T' \overset{?}{\vdash} x \in \mathcal{C}, T' \subsetneq T\} \vdash u \\
\mathsf{R_{unif}}: & (\mathcal{C} \wedge T \overset{?}{\vdash} u, \mathcal{I}) & \rightsquigarrow_\sigma & (\mathcal{C}\sigma \wedge T\sigma \overset{?}{\vdash} u\sigma, \mathcal{I}) \qquad \text{if } \sigma = mgu(t_1, t_2) \\
 & & & \text{where } t_1 \in st(T),\ t_2 \in st(T \cup \{u\}), \text{ and } t_1 \neq t_2 \\
\mathsf{R_{fail}}: & (\mathcal{C} \wedge T \overset{?}{\vdash} u, \mathcal{I}) & \rightsquigarrow & (\bot, \mathcal{I}) \qquad \text{if } vars(T \cup \{u\}) = \emptyset \text{ and } T \not\vdash u \\
\mathsf{R_f}: & (\mathcal{C} \wedge T \overset{?}{\vdash} \mathsf{f}(u,v), \mathcal{I}) & \rightsquigarrow & (\mathcal{C} \wedge T \overset{?}{\vdash} u \wedge T \overset{?}{\vdash} v, \mathcal{I}) \qquad \text{for } \mathsf{f} \in \mathcal{F} \smallsetminus \mathcal{F}_s
\end{array}$$

Figure 2.2: Simplification rules

The rule $\mathsf{R_{ax}}$ removes a redundant constraint, i.e a constraint which is a logical consequence of previous constraints. The rule $\mathsf{R_f}$ decomposes a term $\mathsf{f}(u, v)$. Intuitively, applying this rule means that the intruder has produced the term $\mathsf{f}(u, v)$ by applying function f to the terms u and v. The rule $\mathsf{R_{unif}}$ guesses some equality between two parts of the messages.

All the rules are indexed by a substitution. When there is no index the identity substitution is implicitly assumed. We write $(\mathcal{C}, \mathcal{I}) \rightsquigarrow^n_\sigma (\mathcal{C}', \mathcal{I})$ if there are $\mathcal{C}_1, \ldots, \mathcal{C}_{n-1}$ and $\sigma_1, \ldots, \sigma_n$ such that $(\mathcal{C}, \mathcal{I}) \rightsquigarrow_{\sigma_1} (\mathcal{C}_1, \mathcal{I}) \rightsquigarrow_{\sigma_2} \ldots \rightsquigarrow_{\sigma_n} (\mathcal{C}', \mathcal{I})$ and $\sigma = \sigma_n \circ \cdots \circ \sigma_2 \circ \sigma_1$. We write $(\mathcal{C}, \mathcal{I}) \rightsquigarrow^*_\sigma (\mathcal{C}', \mathcal{I})$ if there exists n such that $(\mathcal{C}, \mathcal{I}) \rightsquigarrow^n_\sigma (\mathcal{C}', \mathcal{I})$.

Our rules are similar to those in [CLCZ10] with a modification for rule $\mathsf{R_{unif}}$. More precisely, we authorize unification with a subterm of the right hand side u of the constraint and also with variables. This will allow us to obtain non-confusing solutions. We will also consider a particular strategy, defined in Figure 2.3, in order to ensure termination in polynomial time. Indeed, applying the simplification rules randomly could yield derivations of exponential length.

The strategy $\mathcal{S}$ is defined in the following way:

- apply $\mathsf{R_{fail}}$ as soon as possible
- apply $\mathsf{R_{unif}}$ up to a point arbitrarily decided, then stop applying it at all.
- Then, assuming that all the constraints are uncolored at the beginning:
  - Consider the uncolored constraint with the largest right-hand side. Either color it or apply $\mathsf{R_f}$ to it. Repeat.
  - When the system is entirely colored, apply $\mathsf{R_{ax}}$.

Figure 2.3: Strategy

Example 2.2.3. *Consider the constraint system $(\mathcal{C}, \mathcal{I})$ defined in Example 2.2.1, representing an execution of the TLS handshake protocol. We can simplify $(\mathcal{C}, \mathcal{I})$ following strategy $\mathcal{S}$:*

- *R_{unif}: $(\mathcal{C}, \mathcal{I}) \rightsquigarrow_\sigma (\mathcal{C}_1, \mathcal{I})$*
  with $\sigma = mgu([\![\langle B, \mathsf{pub}(C)\rangle]\!]_{\mathsf{sk}(S)}, [\![\langle y_2, z_2\rangle]\!]_{\mathsf{sk}(S)}) = \{y_2 \mapsto B, z_2 \mapsto \mathsf{pub}(C)\}$ and
  $\mathcal{C}_1 = \mathcal{C}\sigma = T_1\sigma \overset{?}{\vdash} x_1 \wedge T_2\sigma \overset{?}{\vdash} \langle x_2, [\![\langle B, \mathsf{pub}(C)\rangle]\!]_{\mathsf{sk}(S)}\rangle \wedge T_3\sigma \overset{?}{\vdash} \mathsf{aenc}(x_3, \mathsf{pub}(B))$.

- $\mathsf{R_f}$: $(\mathcal{C}_1, \mathcal{I}) \rightsquigarrow^2 (\mathcal{C}_2, \mathcal{I})$ *with*
  $\mathcal{C}_2 = T_1\sigma \overset{?}{\vdash} x_1 \;\wedge\; T_2\sigma \overset{?}{\vdash} x_2 \;\wedge\; T_2\sigma \overset{?}{\vdash} [\![\langle B, \mathsf{pub}(C)\rangle]\!]_{\mathsf{sk}(S)} \;\wedge\; T_3\sigma \overset{?}{\vdash} x_3 \;\wedge\; T_3\sigma \overset{?}{\vdash} \mathsf{pub}(B)$

- $\mathsf{R_{ax}}$: $(\mathcal{C}_2, \mathcal{I}) \rightsquigarrow^2 (\mathcal{C}', \mathcal{I}) \overset{def}{=} T_1\sigma \overset{?}{\vdash} x_1 \wedge T_2\sigma \overset{?}{\vdash} x_2 \wedge T_3\sigma \overset{?}{\vdash} x_3$

The constraint system $(\mathcal{C}', \mathcal{I})$ *is in solved form, and we have that* $\theta = \theta' \circ \sigma$ *where* $\theta' = \{x_1 \mapsto N_A, x_2 \mapsto N_B, x_3 \mapsto n_1\}$ *is a non-confusing solution of* $(\mathcal{C}', \mathcal{I})$*.*

In the next section, we will show that there is a solution to a constraint system $(\mathcal{C}, \mathcal{I})$ if and only if there is a sequence of simplification rules following strategy $\mathcal{S}$ leading from $(\mathcal{C}, \mathcal{I})$ to $(\mathcal{C}', \mathcal{I})$ such that $(\mathcal{C}', \mathcal{I})$ is a constraint system in solved form that has a non-confusing solution.

2.3 Properties

We show here that the simplification rules allow us to consider simpler constraint systems while preserving the exact same set of solutions. This is ensured by the following theorem.

Theorem 2.3.1. *Let* $(\mathcal{C}, \mathcal{I})$ *be a constraint system. We have that:*

- *Soundness: If* $(\mathcal{C}, \mathcal{I}) \rightsquigarrow^*_\sigma (\mathcal{C}', \mathcal{I})$ *for some constraint system* $(\mathcal{C}', \mathcal{I})$ *and some substitution* σ *and if* θ *is a solution of* $(\mathcal{C}', \mathcal{I})$ *then* $\theta \circ \sigma$ *is a solution of* $(\mathcal{C}, \mathcal{I})$*.*

- *Completeness: If* θ *is a solution of* $(\mathcal{C}, \mathcal{I})$*, then there exist a constraint system* $(\mathcal{C}', \mathcal{I})$ *in solved form and substitutions* σ*,* θ' *such that* $\theta = \theta' \circ \sigma$*,* $(\mathcal{C}, \mathcal{I}) \rightsquigarrow^*_\sigma (\mathcal{C}', \mathcal{I})$ *following the strategy* $\mathcal{S}$*, and* θ' *is a non-confusing solution of* $(\mathcal{C}', \mathcal{I})$*.*

- *Termination: If* $(\mathcal{C}, \mathcal{I}) \rightsquigarrow^n_\sigma (\mathcal{C}', \mathcal{I})$ *following the strategy* $\mathcal{S}$*, then* n *is polynomially bounded in the size of* $\mathcal{C}$*. Moreover, we have that* $st(\mathcal{C}') \subseteq st(\mathcal{C}\sigma) \subseteq st(\mathcal{C})\sigma$*.*

The theorem is a direct consequence of Propositions 2.3.5, 2.3.11, and 2.3.6, that are proven in the following sections.

2.3.1 Soundness

To show soundness, we give simple lemmas: first, we give simple properties of the deduction system, and secondly we show that the monotonicity and origination properties are invariant during simplification, i.e. our simplification rules transform a constraint system into a constraint system.

Lemma 2.3.2. *If* $T \vdash u$ *then* $vars(u) \subseteq vars(T)$*.*

Proof. The proof follows the proof of Lemma 4.4 in [CLCZ10], since no deduction rules introduce new variables. We proceed by induction on the depth of a proof of $T \vdash u$. Indeed, for deduction rules of the form

$$\frac{u_1 \quad \dots \quad u_n}{u}$$

with $n > 0$, we have that $vars(u) \subseteq \bigcup\limits_i vars(u_i)$.

□

The next lemma shows a cut elimination property for our deduction system.

Lemma 2.3.3. *If $T \vdash u$ and $T \cup \{u\} \vdash v$, then $T \vdash v$.*

Proof. Consider a proof π of $T \cup \{u\} \vdash v$ and a proof π' of $T \vdash u$. The tree obtained by replacing each leaf u of π by π' is a proof of $T \vdash v$. $\square$

As a consequence, if $T \cup \{u_1, \dots, u_k\} \vdash u$ and for every u_i, $T \vdash u_i$, then $T \vdash u$. This will be useful to show that a constraint eliminated by the simplification rule R_{ax} did not contain crucial information about possible solutions.

We can now show that monotonicity and origination are invariant by simplification.

Lemma 2.3.4. *Let $(\mathcal{C}, \mathcal{I})$ be a constraint system, and suppose that $(\mathcal{C}, \mathcal{I}) \rightsquigarrow_\sigma (\mathcal{C}', \mathcal{I})$. Then $(\mathcal{C}', \mathcal{I})$ is a constraint system.*

Proof. Let $(\mathcal{C}, \mathcal{I})$ be a constraint system such that $(\mathcal{C}, \mathcal{I}) \rightsquigarrow_\sigma (\mathcal{C}', \mathcal{I})$.

We write $\mathcal{C} = \bigwedge_{i=1}^{n} (T_i \overset{?}{\vdash} u_i)$ and $\mathcal{C}' = \bigwedge_{i=1}^{n'} (T'_i \overset{?}{\vdash} u'_i)$. We show that $(\mathcal{C}', \mathcal{I})$ satisfies the properties defining a constraint system, i.e. :

- monotonicity: $T'_i \subseteq T'_k$ for every i, k such that $1 \leq i < k \leq n'$;
- origination: if $x \in vars(T'_i)$ for some i then there exists $j < i$ such that $x \in vars(u'_j)$.

Since $T_i \subseteq T_k$ implies $T_i\sigma \subseteq T_k\sigma$, $(\mathcal{C}', \mathcal{I})$ satisfies monotonicity.

We show that it also satisfies origination. Consider $i \leq n'$ and $x \in vars(T'_i)$, we have to prove that there exists $j < i$ such that $x \in vars(u'_j)$. We distinguish cases, depending on which simplification rule is applied in the transition $(\mathcal{C}, \mathcal{I}) \rightsquigarrow_\sigma (\mathcal{C}', \mathcal{I})$:

- Case R_{ax}. Assume that it eliminates the constraint $T \overset{?}{\vdash} u$. Then $\mathcal{C}' = \mathcal{C} \smallsetminus \{T \overset{?}{\vdash} u\}$. Let $j = \min\{i \mid x \in var(u_i)\}$. As $(\mathcal{C}, \mathcal{I})$ is a constraint system, j exists and $j < i$. If $T_j \neq T$ then we can choose $j' = j$. We show that this was the only possible case.

  Suppose by contradiction that $T_j = T$. As j is minimal, it follows that $x \notin var(T_j)$ and $x \notin \{y \mid (T_k \overset{?}{\vdash} y) \in \mathcal{C}, k < j\}$. Furthermore, if $T_k \subsetneq T_j$ then $k < j$ since $(\mathcal{C}, \mathcal{I})$ is a constraint system, and thus $\{y \mid (T' \overset{?}{\vdash} y) \in \mathcal{C}, T' \subsetneq T_j\} \subseteq \{y \mid (T_k \overset{?}{\vdash} y) \in \mathcal{C}, k < j\}$. Since $x \in var(u)$, by Lemma 2.3.2, $T \cup \{y \mid (T' \overset{?}{\vdash} y) \in \mathcal{C}, T' \subsetneq T\} \not\vdash u$, and so rule R_{ax} can not be applied, which is in contradiction with our hypothesis. This allows us to conclude.

- Case $\mathsf{R}_{\mathsf{unif}}$. Then there exists a substitution σ such that $\mathcal{C}' = \mathcal{C}\sigma$.

  $(\mathcal{C}, \mathcal{I})$ is a constraint system, so it satisfies origination: if x is a variable and $x \in var(T_i)$ for some $(T_i \overset{?}{\vdash} u_i) \in \mathcal{C}$, then there exists $j < i$ such that $x \in var(u_j)$.

  Let x be a variable such that $x \in var(T_i\sigma)$ for $T_i \overset{?}{\vdash} u_i \in \mathcal{C}$. There exists y such that $x \in var(y\sigma)$ and $y \in var(T_i)$ (we can possibly have $x = y$). There exists $j < i$ such that $y \in vars(u_j)$. Consequently, $x \in vars(u_j\sigma)$.

  Thus, we have that $\mathcal{C}\sigma$ is a constraint system.

- If the rule $\mathsf{R}_{\mathsf{fail}}$ is applied then there is nothing to prove

- If some rule $\mathsf{R_f}$ is applied, then it is applied to some constraint $T_i \overset{?}{\vdash} u_i$. Hence, there exists $f \in \mathcal{F} \smallsetminus \mathcal{F}_s$ such that $u_i = f(u'_i, u'_{i+1})$ and $T'_i = T'_{i+1} = T_i$. We have that :
  - If $j < i$, $(T'_j \overset{?}{\vdash} u'_j) = (T_j \overset{?}{\vdash} u_j)$
  - If $j > i+1$, $(T'_j \overset{?}{\vdash} u'_j) = (T_{j-1} \overset{?}{\vdash} u_{j-1})$

  As $(\mathcal{C}, \mathcal{I})$ is a constraint system, it satisfies origination, and we easily conclude.

Thus, it only remains to show that $st(\mathcal{C}') \cap \mathcal{I} = \emptyset$. We distinguish several cases depending on the rule involved in the transition $(\mathcal{C}, \mathcal{I}) \rightsquigarrow_\sigma (\mathcal{C}', \mathcal{I})$.

- If $(\mathcal{C}', \mathcal{I})$ is obtained by applying $\mathsf{R_{ax}}$, $\mathsf{R_f}$, or $\mathsf{R_{fail}}$, then $st(\mathcal{C}') \subseteq st(\mathcal{C})$, hence $st(\mathcal{C}') \cap \mathcal{I} = \emptyset$.
- If $(\mathcal{C}', \mathcal{I})$ is obtained by applying $\mathsf{R_{unif}}$, it is sufficient to show that $st(\mathcal{C}\sigma) \cap \mathcal{I} = \emptyset$, where $\sigma = mgu(t, u)$, with $t, u \in st(\mathcal{C})$. As $st(\mathcal{C}) \cap \mathcal{I} = \emptyset$, for every $x \in dom(\sigma)$, $st(x\sigma) \cap \mathcal{I} = \emptyset$. So $st(\mathcal{C}\sigma) \cap \mathcal{I} = \emptyset$, i.e. $st(\mathcal{C}') \cap \mathcal{I} = \emptyset$.

This allows us to conclude. □

Hence, applying a simplification rule gives us a new constraint system, whose solutions allow us to build solutions of the first constraint system, as we show now.

Proposition 2.3.5 (soundness)**.** *Let $(\mathcal{C}, \mathcal{I})$ be a constraint system. If $(\mathcal{C}, \mathcal{I}) \rightsquigarrow^*_\sigma (\mathcal{C}', \mathcal{I})$, then $(\mathcal{C}', \mathcal{I})$ is a constraint system and for every solution τ of $(\mathcal{C}', \mathcal{I})$, $\tau \circ \sigma$ is a solution of $(\mathcal{C}, \mathcal{I})$.*

Proof. We show that, if $(\mathcal{C}, \mathcal{I}) \rightsquigarrow_\sigma (\mathcal{C}', \mathcal{I})$, then $(\mathcal{C}', \mathcal{I})$ is a constraint system and for every solution τ of $(\mathcal{C}', \mathcal{I})$, $\tau \circ \sigma$ is a solution of $(\mathcal{C}, \mathcal{I})$. The result of the proposition follows immediately by recursion on the length of the derivation.

Thanks to Lemma 2.3.4, we have that $(\mathcal{C}', \mathcal{I})$ is a constraint system. We reason by case study over the simplification rule used in $(\mathcal{C}, \mathcal{I}) \rightsquigarrow_\sigma (\mathcal{C}', \mathcal{I})$. Let τ be a solution of $(\mathcal{C}', \mathcal{I})$.

Case $\mathsf{R_{ax}}$. In such a case, we have that $\mathcal{C} = \mathcal{C}' \wedge T \overset{?}{\vdash} u$ and $T \cup \{x \mid (T' \overset{?}{\vdash} x) \in \mathcal{C}, T' \subsetneq T\} \vdash u$. It follows that:

$$T\tau \cup \{x\tau \mid (T' \overset{?}{\vdash} x) \in \mathcal{C}, T' \subsetneq T\} \vdash u\tau.$$

Each constraint $T' \overset{?}{\vdash} x$ in $\mathcal{C}$ with $T' \subsetneq T$ is also a constraint in $\mathcal{C}'$. Thus, for all such constraints, we have that $T'\tau \cup \mathcal{I} \vdash x\tau$, and hence $T\tau \cup \mathcal{I} \vdash x\tau$. Then, as $T\tau \cup \{x\tau \mid (T' \overset{?}{\vdash} x) \in \mathcal{C}, T' \subsetneq T\} \vdash u\tau$, we obtain through Lemma 2.3.3 that $T\tau \cup \mathcal{I} \vdash u\tau$, and we deduce that τ is a solution of $(\mathcal{C}, \mathcal{I})$.

Case $\mathsf{R_{unif}}$. In such a case, there exists a substitution σ such that $\mathcal{C}' = \mathcal{C}\sigma$. For every constraint $T \overset{?}{\vdash} u$ of $\mathcal{C}$, $T\sigma \overset{?}{\vdash} u\sigma$ is a constraint of $\mathcal{C}'$. As τ is a solution of $(\mathcal{C}', \mathcal{I})$, $(T\sigma)\tau \cup \mathcal{I} \vdash (u\sigma)\tau$, hence $\tau \circ \sigma$ is a solution of $(\mathcal{C}, \mathcal{I})$.

Case $\mathsf{R_f}$. In such a case, we have that $\mathcal{C} = \mathcal{C}_0 \wedge T \overset{?}{\vdash} \mathsf{f}(u, v)$, and $\mathcal{C}' = \mathcal{C}_0 \wedge T \overset{?}{\vdash} u \wedge T \overset{?}{\vdash} v$. We know that τ is a solution of $(\mathcal{C}', \mathcal{I})$, so in particular $T\tau \cup \mathcal{I} \vdash u\tau$ and $T\tau \cup \mathcal{I} \vdash v\tau$. By applying the corresponding inference rule, we obtain that $T\tau \cup \mathcal{I} \vdash \mathsf{f}(u, v)\tau$. Moreover, for every $T \overset{?}{\vdash} v \in \mathcal{C}_0, T\tau \cup \mathcal{I} \vdash v\tau$. And so, in that case, τ is a solution of $(\mathcal{C}, \mathcal{I})$.

Case $\mathsf{R_{fail}}$. This case is impossible since τ is a solution of $(\mathcal{C}', \mathcal{I})$. □

2.3.2 Termination

We show termination before we show completeness as the latter notion depends on termination. Applying the simplification rules terminates, whatever strategy is used, but we want a stronger result. There may be derivations of exponential length in the size of the constraints, but intuitively we can restrict ourselves to derivations of polynomial length by considering a suitable strategy. We consider the strategy defined in Figure 2.3.

Applying rules $\mathsf{R_{unif}}$ first allows us to guess all possible equalities and obtain a non-confusing solution. We show that we obtain non-confusing solutions and that this strategy is complete later on, in Section 2.3.3.

We show now that derivations following strategy $\mathcal{S}$ are polynomial in the size of $\mathcal{C}$. Intuitively, a derivation of exponential length may occur if a constraint $T \overset{?}{\vdash} u$ is considered several times. When a constraint is considered, it is eliminated and can be replaced by other ones (or lead to the empty constraint system). Notice that Rule $\mathsf{R_{ax}}$ does not generate new constraints. It is mainly when applying Rule $\mathsf{R_f}$ that we have to be careful, hence our strategy. As we may consider constraint systems which have distinct right-hand sides, choosing to consider the constraint with the largest right-hand side allows us to ensure that we will never consider it again for Rule $\mathsf{R_f}$.

Proposition 2.3.6 (complexity). *Let $(\mathcal{C},\mathcal{I})$ be a constraint system. If there is a derivation $(\mathcal{C},\mathcal{I}) \rightsquigarrow^n_\sigma (\mathcal{C}',\mathcal{I})$ following the strategy $\mathcal{S}$ for some constraint system $(\mathcal{C}',\mathcal{I})$ and some substitution σ, then n is polynomially bounded in the size of $\mathcal{C}$. Moreover, $st(\mathcal{C}') \subseteq st(\mathcal{C}\sigma) \subseteq st(\mathcal{C})\sigma$*

Proof. As a first step, we show that n is polynomially bounded in the size of $\mathcal{C}$. First, we prove that we cannot get twice the same constraint in the part of a derivation following strategy $\mathcal{S}$ using only rule $\mathsf{R_f}$. We denote by $\mathsf{rhs}(\mathcal{C})$ the right-hand side terms of $\mathcal{C}$. Consider a derivation sequence following the strategy $\mathcal{S}$.

$$(\mathcal{C}_0,\mathcal{I}) \overset{\mathsf{R_f}}{\rightsquigarrow} (\mathcal{C}_1,\mathcal{I}) \overset{\mathsf{R_f}}{\rightsquigarrow} \dots \overset{\mathsf{R_f}}{\rightsquigarrow} (\mathcal{C}_n,\mathcal{I}).$$

At each step i of this derivation, a constraint $T \overset{?}{\vdash} u$ is eliminated from $\mathcal{C}_i$, i.e. $T \overset{?}{\vdash} u \in \mathcal{C}_i \smallsetminus \mathcal{C}_{i+1}$ (this follows from the fact that the constraints of $\mathcal{C}_i$ are all distinct). If $T \overset{?}{\vdash} u \in \mathcal{C}_i \smallsetminus \mathcal{C}_{i+1}$ ($T \overset{?}{\vdash} u$ has been eliminated at this step), then, for any $j > i$, we show that $T \overset{?}{\vdash} u \notin \mathcal{C}_j$. Indeed, as the derivation follows strategy $\mathcal{S}$, it means that $T \overset{?}{\vdash} u$ is (one of) the uncolored constraint(s) with the largest right-hand side in $\mathcal{C}_i$. So we have that $\|u\| = \max_{t\in\mathsf{rhs}(\mathcal{C}'_i)} \|t\|$ (where $\mathcal{C}'_i$ is the set of uncolored constraints of $\mathcal{C}_i$). Suppose by contradiction that for some $j > i$, the constraint $T \overset{?}{\vdash} u$ was in $\mathcal{C}_{j+1}$ and not in $\mathcal{C}_j$. According to the strategy, rule $\mathsf{R_f}$ has been applied to the uncolored constraint with the largest right-hand side, and furthermore it has produced constraint $T \overset{?}{\vdash} u$. Hence, there exists v such that $u \in st(v)$, $\|u\| < \|v\|$ and $\|v\| = \max_{t\in\mathsf{rhs}(\mathcal{C}'_j)} \|t\|$. Thus the maximum of the sizes of the right-hand side terms of the uncolored constraints has strictly increased, which is impossible according to our strategy.

We want to show that derivation $(\mathcal{C},\mathcal{I}) \rightsquigarrow^n_\sigma (\mathcal{C}',\mathcal{I})$ following strategy $\mathcal{S}$ is of polynomial length. According to the definition of strategy $\mathcal{S}$, there exist $\mathcal{C}_1, \mathcal{C}_2$ such that

$$(\mathcal{C},\mathcal{I}) \overset{\mathsf{R_{unif}}}{\rightsquigarrow}_\sigma (\mathcal{C}_1,\mathcal{I}) \overset{\mathsf{R}_f}{\rightsquigarrow} (\mathcal{C}_2,\mathcal{I}) \overset{\mathsf{R_{ax}}}{\rightsquigarrow} (\mathcal{C}',\mathcal{I})$$

We assume a DAG representation of the terms and constraints, in such a way that the size of the constraints is proportional to the number of the distinct subterms occurring in it. Next, observe that

$$\#st(t\theta) \leq \#(st(t) \cup \bigcup_{x \in dom(\theta)} st(x\theta)).$$

Moreover, when unifying two subterms of t with mgu θ, $\#st(t\theta) \leq \#st(t)$ since, for every variable $x \in dom(\theta)$, $x\theta \in (st(t) \smallsetminus \{x\})\theta$, and so for every term $u \in st(t\theta)$, $u \in st(t)\theta$. It follows that, for any constraint system $\mathcal{C}'$ such that $(\mathcal{C}, \mathcal{I}) \rightsquigarrow_\sigma^* (\mathcal{C}', \mathcal{I})$ using only rule $\mathsf{R_{unif}}$, $st(\mathcal{C}') = st(\mathcal{C}\sigma) \subseteq st(\mathcal{C})\sigma$. Consequently, $\#st(\mathcal{C}') \leq \#st(\mathcal{C})$. In particular, $\#st(\mathcal{C}_1) \leq \#st(\mathcal{C})$.

Next, observe that the number of distinct left hand sides of the constraints, denoted by $\mathsf{lhs}(\mathcal{C}')$, is never increasing: $\#\mathsf{lhs}(\mathcal{C}') \leq \#\mathsf{lhs}(\mathcal{C})$ if $(\mathcal{C}, \mathcal{I}) \rightsquigarrow^* (\mathcal{C}', \mathcal{I})$. Furthermore, as long as we only apply the rules $\mathsf{R_{ax}}$ and $\mathsf{R_f}$, starting from $(\mathcal{C}_1, \mathcal{I})$, the left hand sides of the deduction constraint system are fixed: there are at most $\#\mathsf{lhs}(\mathcal{C}_1)$ of them. Now, since we cannot consider twice the same constraints, the number of consecutive applications of rules $\mathsf{R_{ax}}$ and $\mathsf{R_f}$ is bounded by

$$\#\mathsf{lhs}(\mathcal{C}_1) \times \#st(\mathsf{rhs}(\mathcal{C}_1)) \leq \#\mathsf{lhs}(\mathcal{C}) \times \#st(\mathcal{C})$$

Moreover, when applying rules $\mathsf{R_{ax}}$ and $\mathsf{R_f}$, it is clear that $st(\mathcal{C}') \subseteq st(\mathcal{C}_1) \subseteq st(\mathcal{C})\sigma$.

It follows that the length of a derivation sequence is bounded by $\#\mathsf{lhs}(\mathcal{C}) \times \#st(\mathcal{C})$ (for $\mathsf{R_{ax}}, \mathsf{R_f}$ steps) plus $\#var(\mathcal{C})$ (for $\mathsf{R_{unif}}$ steps) plus 1 (for a possible $\mathsf{R_{fail}}$ step).

$\square$

2.3.3 Completeness

We want to show that the strategy $\mathcal{S}$ defined in the previous subsection is complete, i.e. there exists a derivation following strategy $\mathcal{S}$ leading to a solved constraint system with a non-confusing solution, from which we can reconstruct our initial solution. Moreover, we will see that the solution of the solved constraint system is non-confusing.

The strategy $\mathcal{S}$ can be divided into two phases:

- First, apply only rules $\mathsf{R_{unif}}$ to obtain a constraint system with a non-confusing solution.
- Then, use rules $\mathsf{R_f}$ and $\mathsf{R_{ax}}$ to obtain a solved constraint system.

Consider a constraint system $(\mathcal{C}, \mathcal{I})$ and a substitution θ such that θ is a solution of $(\mathcal{C}, \mathcal{I})$. We want to build a derivation following strategy $\mathcal{S}$: $(\mathcal{C}, \mathcal{I}) \rightsquigarrow_\sigma^* (\mathcal{C}', \mathcal{I})$ such that $(\mathcal{C}', \mathcal{I})$ is in solved form, there exists a substitution θ' verifying $\theta = \theta' \circ \sigma$ and θ' is a non-confusing solution of $(\mathcal{C}', \mathcal{I})$. In fact, we use a stronger notion than non-confusing, that we define now.

Definition 2.3.1 (strongly non-confusing)**.** *Let $(\mathcal{C}, \mathcal{I})$ be a constraint system with $\mathcal{C} = \bigwedge_{i=1}^{n} T_i \overset{?}{\vdash} u_i$. A substitution θ is a* strongly non-confusing *solution of $(\mathcal{C}, \mathcal{I})$ if θ is a solution of $(\mathcal{C}, \mathcal{I})$ and, for every $1 \leq i \leq n$, for every terms $t_1 \in st(T_i)$ and $t_2 \in st(T_i \cup \{u_i\})$, we have that $t_1\theta = t_2\theta$ implies that $t_1 = t_2$.*

Intuitively, when a solution θ of a constraint system $(\mathcal{C}, \mathcal{I})$ is strongly non-confusing, rules $\mathsf{R_{unif}}$ cannot be applied to $(\mathcal{C}, \mathcal{I})$ while keeping this solution. Note that if θ is a strongly non-confusing solution of $(\mathcal{C}, \mathcal{I})$ then in particular θ is a non-confusing solution of $(\mathcal{C}, \mathcal{I})$. Note also that these two notions coincide on constraint systems in solved form.

Example 2.3.1. *θ' defined in Example 2.2.3 is not strongly non-confusing, as $x_1\theta' = N_A$, for instance, and $N_A, x_1 \in st(T_1\sigma, x_1)$. An example of strongly non-confusing solution is $\tau = \{x_1 \mapsto n_1, x_2 \mapsto n_2, x_3 \mapsto n_3\}$.*

If θ' is a strongly non-confusing solution of $(\mathcal{C}_1, \mathcal{I})$, and there is a derivation using only rules $\mathsf{R_f}$ and $\mathsf{R_{ax}}$ leading from $(\mathcal{C}_1, \mathcal{I})$ to $(\mathcal{C}_2, \mathcal{I})$, then θ' is a strongly non-confusing solution of $(\mathcal{C}_2, \mathcal{I})$. So it only remains for us to obtain a solved constraint system thanks to the second part of the strategy.

In this part of the strategy, we consider the largest uncolored constraint $T \overset{?}{\vdash} u$ and we have to either color it or apply some rule $\mathsf{R_f}$ to it. We decide between the two possibilities by considering a proof of $T \overset{?}{\vdash} u$ and whether it ends in a composition or not. We thus wish to consider proofs with the property that their last rule does not vary along a derivation. We define such proofs now.

Definition 2.3.2 (simple proof)**.** *Let $T_1 \subseteq T_2 \subseteq \cdots \subseteq T_n$. We say that a proof π of $T_i \vdash u$ is* left-minimal *with respect to $T_1, \ldots, T_n$ if, whenever there is a proof of $T_j \vdash u$ for some $j < i$, then π is a proof of $T_j \vdash u$.*

We say that a proof is simple *if all of its subproofs are left-minimal and there is no repeated label on any branch.*

Example 2.3.2. *Consider the following constraint system:*

$$T_1 = \{a\} \overset{?}{\vdash} x_1 \wedge T_2 = \{a, \langle a, b\rangle\} \overset{?}{\vdash} x_2.$$

$\dfrac{\langle a, b\rangle}{a}$ *is a proof of $T_2 \vdash a$, but it is not a simple proof. Indeed, it is not a proof of $T_1 \vdash a$, even though there is a proof of $T_1 \vdash a$.*

The definition of simple proofs we use is inherited from [CLCZ10], as is the next lemma, which shows that it is always possible to consider simple proofs. We can prove the lemma by following strictly the proof given in [CLCZ10] (Lemma 4.8), as it does not depend on the signature.

Lemma 2.3.7. *Let $T_1 \subseteq T_2 \subseteq \cdots \subseteq T_n$. If there is a proof of $T_i \vdash u$, then there is a simple proof of it.*

We show that if part of the constraint system is already in solved form and the next constraint is $T \overset{?}{\vdash} v$ with Δ a simple proof of $T\theta \cup \mathcal{I} \vdash u$, then there is a term $t \in T$ such that $t\theta = u$. This result will be useful for proving that we can apply R_{ax} on constraints whose simple proofs end with a decomposition.

Lemma 2.3.8. *Let $(\mathcal{C}, \mathcal{I})$ be a constraint system, θ be a solution of $(\mathcal{C}, \mathcal{I})$, $T_i \in \mathsf{lhs}(\mathcal{C})$ such that for any $(T \overset{?}{\vdash} v) \in \mathcal{C}$, if $T \subsetneq T_i$, then v is a variable. Let u be a term such that $u \notin \mathcal{I}$. If there is a simple proof of $T_i\theta \cup \mathcal{I} \vdash u$, that is reduced to a leaf or whose last inference rule is a decomposition, then there is $t \in st(T_i) \smallsetminus \mathcal{X}$ such that $t\theta = u$.*

Proof. Consider a simple proof π of $T_i\theta \cup \mathcal{I} \vdash u$ that is reduced to a leaf or whose last inference rule is a decomposition. We may assume, without loss of generality, that i is minimal. Otherwise, we have that $T_j\theta \cup \mathcal{I} \vdash u$ is derivable with $j < i$. In that case, as π is left-minimal, we still have a proof tree whose last inference rule is a decomposition. Such a $T_j \subseteq T_i$ also satisfies the hypotheses of the lemma.
We reason by induction on the depth of the proof π. We make a case distinction, depending on the last rule of π.

The last rule is an axiom. Then $u \in T_i\theta \cup \mathcal{I}$ and there is $t \in T_i \cup \mathcal{I}$ such that $t\theta = u$. By hypothesis, $u \notin \mathcal{I}$, so $t \notin \mathcal{I}$, *i.e.* $t \in T_i$ (and $t \in st(T_i)$). Suppose by contradiction that t is a variable. Then, by definition of a constraint system, there exists $T_j \overset{?}{\vdash} w \in \mathcal{C}$, with $t \in var(w)$, such that $T_j \subsetneq T_i$. Moreover, by hypothesis of the lemma, w must be a variable. Hence $t = w$. Then $T_j\theta \cup \mathcal{I} \vdash u$, which contradicts the minimality of i.

The last rule is a symmetric decryption. In such a case, we have that:

$$\frac{\mathsf{senc}(u, w) \qquad w}{u}$$

Let π_1 be the proof of $T_i\theta \cup \mathcal{I} \vdash \mathsf{senc}(u, w)$. As π is simple, the last rule of π_1 cannot be a composition, or else $T_i\theta \cup \mathcal{I} \vdash u$ would appear twice on the same path. Then, by induction hypothesis, there is $t \in st(T_i) \smallsetminus \mathcal{X}$ such that $t\theta = \mathsf{senc}(u, w)$. It follows that $t = \mathsf{senc}(t', t'')$ with $t'\theta = u$. If t' was a variable, then there would exist $T_j \overset{?}{\vdash} w \in \mathcal{C}$, with $T_j \subsetneq T_i$ such that $T_j\theta \cup \mathcal{I} \vdash t'\theta$. (because $t' \in var(w)$ and $w \in \mathcal{X}$). Hence we would have that $T_j\theta \cup \mathcal{I} \vdash u$, which contradicts the minimality of i. Hence t' is not variable.

For the other decomposition rules, the proof is similar to the previous case. □

The next two lemmas explain respectively how to apply rule $\mathsf{R_f}$ when the constraint considered has a simple proof ending with a composition, or how to apply rule $\mathsf{R_{ax}}$ when the constraint considered has a simple proof ending with a decomposition.

Lemma 2.3.9 (composition). *Let $(\mathcal{C}, \mathcal{I})$ be a constraint system, θ a strongly non-confusing solution of $(\mathcal{C}, \mathcal{I})$. Let $T \overset{?}{\vdash} u \in \mathcal{C}$ with u not a variable and π be a proof of $T\theta \cup \mathcal{I} \vdash u\theta$ that ends with a composition rule. We can apply the rule $\mathsf{R_f}$ on $T \overset{?}{\vdash} u$, yielding a constraint system $(\mathcal{C}', \mathcal{I})$ such that θ is a strongly non-confusing solution of $(\mathcal{C}', \mathcal{I})$.*

Proof. Since u is not a variable, we have that $u = \mathsf{f}(v_1, \ldots, v_p)$. The last rule of π ends with a composition, so $T\theta \cup \mathcal{I} \vdash v_j\theta$ for every $1 \leq j \leq p$. Then we can apply the simplification rule $\mathsf{R_f}$ to $(\mathcal{C}, \mathcal{I})$, yielding constraints $T \overset{?}{\vdash} v_j$ in $\mathcal{C}'$ for every $1 \leq j \leq p$. Clearly, we have that θ is a solution of the constraint system $(\mathcal{C}', \mathcal{I})$. It remains to show that θ is a strongly non-confusing solution of $(\mathcal{C}', \mathcal{I})$. Assume by contradiction that this is not the case. This means that there exist $t_1 \in st(T)$ and $t_2 \in st(T \cup \{v_j\})$ with $j \in \{1, \ldots p\}$ such that $t_1\theta = t_2\theta$ and $t_1 \neq t_2$. This would imply that θ is not a strongly non-confusing solution of $(\mathcal{C}, \mathcal{I})$. Hence contradiction. □

Lemma 2.3.10 (decomposition). *Let $(\mathcal{C}, \mathcal{I})$ be a constraint system not in solved form and θ be a strongly non-confusing solution of $(\mathcal{C}, \mathcal{I})$. Suppose that for every constraint $T \overset{?}{\vdash} u \in \mathcal{C}$*

such that $u \notin \mathcal{X}$, there exists a simple proof π of $T\theta \cup \mathcal{I} \vdash u\theta$ which ends with a decomposition. Then there exists a constraint system $(\mathcal{C}', \mathcal{I})$ such that $(\mathcal{C}, \mathcal{I}) \rightsquigarrow (\mathcal{C}', \mathcal{I})$ using $\mathsf{R_{ax}}$ and θ is a strongly non-confusing solution of $(\mathcal{C}', \mathcal{I})$. Furthermore, for every constraint $T \overset{?}{\vdash} u \in \mathcal{C}'$ such that $u \notin \mathcal{X}$, there exists a simple proof π of $T\theta \cup \mathcal{I} \vdash u\theta$ which ends with a decomposition.

Proof. Let $\mathcal{C} = \bigwedge_{i=1}^{n} T_i \overset{?}{\vdash} u_i$. Consider a constraint $T_i \overset{?}{\vdash} u_i$ such that $u_i \notin \mathcal{X}$ and for all $j < i$, $u_j \in \mathcal{X}$. We show that for every $u \in st(T_i) \smallsetminus \mathcal{X}$, if $T_i\theta \cup \mathcal{I} \vdash u\theta$, then $T_i' \vdash u$, where $T_i' = T_i \cup \{x \mid (T \overset{?}{\vdash} x) \in \mathcal{C} \text{ and } T \subsetneq T_i\}$. Consider a simple proof π of $T_i\theta \cup \mathcal{I} \vdash u\theta$. We show this result by induction on $|\pi|$ where $|\pi|$ is the size, *i.e.* number of nodes, of π.

Base case: $|\pi| = 1$. In such a case, we have that there is $t \in T_i \cup \mathcal{I}$ such that $t\theta = u\theta$. Actually, since $u \notin \mathcal{X}$, we have that $t \in T_i$ and thus, using the fact that θ is strongly non-confusing, we deduce that $t = u$. Hence $u \in T_i$, so $T_i \vdash u$, and as $T_i \subseteq T_i'$, we have $T_i' \vdash u$.

Induction step: $|\pi| > 1$. We distinguish several cases depending on the last rule of π.

The last rule is a symmetric decryption rule. In such a case, we have that:

$$\frac{\mathsf{senc}(u\theta, w) \qquad w}{u\theta}$$

As π is simple, the last rule of the proof of $T_i\theta \cup \mathcal{I} \vdash \mathsf{senc}(u\theta, w)$ is a decomposition. Furthermore, $\mathsf{senc}(u\theta, w) \notin \mathcal{I}$. Consequently, thanks to Lemma 2.3.8, there is $t \in st(T_i) \smallsetminus \mathcal{X}$ such that $t\theta = \mathsf{senc}(u\theta, w)$. Let $t = \mathsf{senc}(t_1, t_2)$ and $t_1\theta = u\theta$, $t_2\theta = w$. By induction hypothesis, we have that $T_i' \vdash t$.

As θ is strongly non-confusing and $t_1\theta = u\theta$ with $t_1 \in st(T_i)$, we get that $t_1 = u$. If t_2 is a variable, then $t_2 \in var(T_i)$, and by definition of a constraint system there exists $j < i$ such that $T_j \overset{?}{\vdash} u_j \in \mathcal{C}$ and $t_2 \in var(u_j)$. By hypothesis, $u_j \in \mathcal{X}$, so $t_2 = u_j$, so $t_2 \in T_i'$. If t_2 is not a variable, we can apply the induction hypothesis, and we deduce that $T_i' \vdash t_2$. So, in any case, $T_i' \vdash t_2$.

Now, we have both that $T_i' \vdash \mathsf{senc}(u, t_2)$ and $T_i' \vdash t_2$, from which we conclude that $T_i' \vdash u$ by applying the symmetric decryption rule.

The last rule is an asymmetric decryption rule. In such a case, we have that:

$$\frac{\mathsf{aenc}(u\theta, \mathsf{pub}(v)) \qquad \mathsf{priv}(v)}{u\theta}$$

As π is simple, the last rule of the proof of $T_j\theta \cup \mathcal{I} \vdash \mathsf{aenc}(u\theta, \mathsf{pub}(v))$ is a decomposition. Furthermore, $\mathsf{aenc}(u\theta, \mathsf{pub}(v)) \notin \mathcal{I}$. Consequently, thanks to Lemma 2.3.8, there is $t \in st(T_i) \smallsetminus \mathcal{X}$ such that $t\theta = \mathsf{aenc}(u\theta, \mathsf{pub}(v))$. Let $t = \mathsf{aenc}(t_1, t_2)$ with $t_1\theta = u\theta$, $t_2\theta = \mathsf{pub}(v)$. By induction hypothesis, we have that $T_i' \vdash t$.

As θ is strongly non-confusing and $t_1\theta = u\theta$ with $t_1 \in st(T_i)$, we get that $t_1 = u$. On the other hand, the last rule in the proof of $T_j\theta \cup \mathcal{I} \vdash \mathsf{sk}(v)$ is a decomposition (no composition rule can yield a term headed with $\mathsf{priv}()$). Then, by Lemma 2.3.8, there is $w \in st(T_i) \smallsetminus \mathcal{X}$ such that $w\theta = \mathsf{priv}(v)$. Let $w = \mathsf{priv}(w')$. By induction hypothesis, $T_i' \vdash \mathsf{priv}(w')$.

$$\frac{\begin{array}{c}(\mathsf{aenc}(t_1, t_2))\theta \\ \| \\ \mathsf{aenc}(u\theta, \mathsf{pub}(v))\end{array} \qquad \begin{array}{c}\mathsf{priv}(w')\theta \\ \| \\ \mathsf{priv}(v)\end{array}}{u\theta}$$

Now, $t_2 \in st(t)$, and $t \in st(T_i)$, so $t_2 \in st(T_i)$. If t_2 is a variable, then $t_2 \in var(T_i)$, and by definition of a constraint system there exists $j < i$ such that $T_j \overset{?}{\vdash} u_j \in \mathcal{C}$ and $t_2 \in var(u_j)$. By hypothesis, $u_j \in \mathcal{X}$, so $t_2 = u_j$. Hence, we have a simple proof of $T_j\theta \cup \mathcal{I} \vdash t_2\theta$. Note that this proof is either reduced to a leaf or ends with a decomposition rule since there is no composition rule yielding to a term headed with $\mathsf{pub}()$. Hence, we apply Lemma 2.3.8, and we deduce that there exists $t_3 \in st(T_j) \smallsetminus \mathcal{X}$ such that $t_3\theta = t_2\theta$. Hence, we have that $t_3 = \mathsf{pub}(t_4)$ with $t_4\theta = v$ with $t_4 \in st(T_i)$.

Similarly, $w' \in st(w)$, and $w \in st(T_i)$, so $w' \in st(T_i)$. Moreover, $t_4\theta = v = w'\theta$, and θ is non-confusing, so $t_4 = w'$. Now, we have both that $T'_i \vdash \mathsf{aenc}(u, \mathsf{pub}(w'))$ and $T'_i \vdash \mathsf{priv}(w')$, from which we conclude that $T'_i \vdash u$ by applying the asymmetric decryption rule.

Similarly, we can conclude for the optional rule.

The last rule is a projection rule. By symmetry, we can assume that

$$\frac{\mathsf{f}(u\theta, v)}{u\theta}$$

with $\mathsf{f} \in \{\langle,\rangle, ::\}$.

As π is simple, the last rule of the proof of $T_i\theta \cup \mathcal{I} \vdash \mathsf{f}(u\theta, v)$ is a decomposition, and so, thanks to Lemma 2.3.8, there is $t \in st(T_i) \smallsetminus \mathcal{X}$ such that $t\theta = \mathsf{f}(u\theta, v)$. Let $t = \mathsf{f}(t_1, t_2)$. By induction hypothesis, $T'_i \vdash t$.

Now, as θ is strongly non-confusing and $t_1\theta = u\theta$, we have that $t_1 = u$. From $T'_i \vdash \mathsf{f}(u, t_2)$, we deduce $T'_i \vdash u$ by projection.

The last rule is a composition In such a case, we have that:

$$\frac{v_1 \quad \ldots \quad v_n}{\mathsf{f}(v_1, \ldots, v_n)}$$

with $u\theta = \mathsf{f}(v_1, \ldots, v_n)$. Since u is not a variable, $u = \mathsf{f}(w_1, \ldots, w_n)$, with $w_k\theta = v_k$ for all k. If w_k is a variable, then $w_k \in var(T_i)$, and by definition of a constraint system there exists $j < i$ such that $T_j \overset{?}{\vdash} u_j \in \mathcal{C}$ and $w_k \in var(u_j)$. By hypothesis, $u_j \in \mathcal{X}$, so $w_k = u_j$, so $w_k \in T'_i$. If w_k is not a variable, we can apply the induction hypothesis, and we deduce that $T'_i \vdash w_k$. So, for every k, we get that in all possible cases $T'_i \vdash w_k$. Thus $T'_i \vdash u$ by applying the composition rule.

We have shown that for every $u \in st(T_i) \smallsetminus \mathcal{X}$, if $T_i\theta \cup \mathcal{I} \vdash u\theta$, then $T'_i \vdash u$, where $T'_i = T_i \cup \{x \mid (T \overset{?}{\vdash} x) \in \mathcal{C}, T \subsetneq T_i\}$. Consider the term u_i. We know that there is a simple proof of $T_i\theta \cup \mathcal{I} \vdash u_i\theta$ which ends with a decomposition, so by applying Lemma 2.3.8 (since $u_i \in \mathcal{C}$ and $st(\mathcal{C}) \cap \mathcal{I} = \emptyset$, $u_i \notin \mathcal{I}$), there is $t \in st(T_i) \smallsetminus \mathcal{X}$ such that $t\theta = u_i\theta$. As θ is strongly non-confusing, we deduce that $t = u_i$, and $u_i \in st(T_i) \smallsetminus \mathcal{X}$ such that $T_i\theta \cup \mathcal{I} \vdash u_i\theta$. Hence, $T'_i \vdash u_i$. Consequently, we can apply $\mathsf{R_{ax}}$ to the constraint $T_i \overset{?}{\vdash} u_i$, and we get a constraint system $\mathcal{C}'$.

Furthermore, consider $T_j \overset{?}{\vdash} u \in \mathcal{C}'$ with u not a variable. We know that $T_j \overset{?}{\vdash} u \in \mathcal{C}$, and there exists a simple proof π of $T_j\theta \cup \mathcal{I} \vdash u\theta$ with respect to the left-hand sides of $\mathcal{C}$ which ends with a decomposition. As π is simple w.r.t. $\mathsf{lhs}(\mathcal{C})$, and $\mathsf{lhs}(\mathcal{C}') \subseteq \mathsf{lhs}(\mathcal{C})$, π is simple w.r.t. $\mathsf{lhs}(\mathcal{C}')$. □

We can now prove formally the completeness of our strategy $\mathcal{S}$.

Proposition 2.3.11 (completeness). *Let $(\mathcal{C},\mathcal{I})$ be a constraint system and θ be a solution of $(\mathcal{C},\mathcal{I})$. There exist a constraint system $(\mathcal{C}',\mathcal{I})$ in solved form, substitutions σ and θ' such that $(\mathcal{C},\mathcal{I}) \rightsquigarrow^*_\sigma (\mathcal{C}',\mathcal{I})$ following the strategy $\mathcal{S}$, $\theta = \theta' \circ \sigma$ and θ' is a (strongly) non-confusing solution of $(\mathcal{C}',\mathcal{I})$.*

Proof. Let $\mathcal{C} = \bigwedge_{i=1}^{n} T_i \overset{?}{\vdash} u_i$.

Step 1. First, we show that there exist substitutions θ', σ and a constraint system $(\mathcal{C}_1,\mathcal{I})$ such that $(\mathcal{C},\mathcal{I}) \rightsquigarrow^*_\sigma (\mathcal{C}_1,\mathcal{I})$ using only rules $\mathsf{R}_{\mathsf{unif}}$ and θ' is a strongly non-confusing solution of $(\mathcal{C}_1,\mathcal{I})$ with $\theta = \theta' \circ \sigma$. We show this result by induction on $\#vars(\mathcal{C})$.

Base case: $\#vars(\mathcal{C}) = 0$. In such a case, we have that $dom(\theta) = \emptyset$ and thus θ is a strongly non-confusing solution of $(\mathcal{C},\mathcal{I})$. We easily conclude.

Induction step: $\#vars(\mathcal{C}) > 0$. In such a case, either θ is already a non-confusing solution of $(\mathcal{C},\mathcal{I})$ and we easily conclude. Otherwise, we have that there exist $1 \leq i \leq n$, $t_1 \in st(T_i)$, and $t_2 \in st(T_i \cup \{u_i\})$ such that $t_1\theta = t_2\theta$ with $t_1 \neq t_2$. In such a case, we apply $\mathsf{R}_{\mathsf{unif}}$ obtaining a constraint system on which we can apply our induction hypothesis. This allows us to conclude for this first step.

Step 2. Now, θ' is a strongly non-confusing solution of $(\mathcal{C}_1,\mathcal{I})$. We still need to derive $\mathcal{C}'$ in solved form from $\mathcal{C}_1$. To do that, we follow the strategy $\mathcal{S}$. Intuitively, we consider each constraint and either decompose it or color it to remember not to consider it anymore.

We derive a colored constraint system from $\mathcal{C}_1$ in the following way: Select a constraint $T \overset{?}{\vdash} u$ among the uncolored constraints with the largest right-hand sides. According to the strategy $\mathcal{S}$, we must either color it or decompose it. We describe how we choose between these two possibilities.

- If u is a variable, color the constraint.
- Else, consider a simple proof Δ of $T\theta' \cup \mathcal{I} \vdash u\theta'$:
  - If Δ ends with a decomposition, we color the constraint $T \overset{?}{\vdash} u$.
  - If Δ ends with a composition, we apply rule R_{f} to the constraint $T \overset{?}{\vdash} u$.

We show that this procedure terminates and produces a derivation:

$$(\mathcal{C}_1,\mathcal{I}) \overset{\mathsf{R}_{\mathsf{f}}}{\rightsquigarrow} (\mathcal{C}_2,\mathcal{I}) \overset{\mathsf{R}_{\mathsf{f}}}{\rightsquigarrow} (\mathcal{C}_3,\mathcal{I}) \ldots \overset{\mathsf{R}_{\mathsf{f}}}{\rightsquigarrow} (\mathcal{C}_\ell,\mathcal{I})$$

where $\mathcal{C}_\ell$ is totally colored, and for each $(T \overset{?}{\vdash} u) \in \mathcal{C}_\ell$, either $u \in \mathcal{X}$ or there is a simple proof of $T\theta' \cup \mathcal{I} \vdash u\theta'$ whose last rule is a decomposition. Indeed, consider a constraint $T \overset{?}{\vdash} u$, uncolored in $\mathcal{C}_{i-1}$ and colored in $\mathcal{C}_i$. Then, either u is a variable, or there is a simple proof Δ of $T\theta' \cup \mathcal{I} \vdash u\theta'$ w.r.t. $\mathsf{lhs}(\mathcal{C}_{i-1}\theta')$ ending with a decomposition. Furthermore, if Δ is a simple proof of a colored constraint and we apply R_{f} to another constraint, Δ is still a simple proof in the constraint system obtained after simplification.

Regarding termination, note that the total size of the right-hand sides of the uncolored constraints strictly decreases, either because we color a constraint or we apply R_{f} to an uncolored constraint.

Applying Lemma 2.3.9 recursively on the derivation

$$(\mathcal{C}_1,\mathcal{I}) \overset{\mathsf{R}_{\mathsf{f}}}{\rightsquigarrow} (\mathcal{C}_2,\mathcal{I}) \overset{\mathsf{R}_{\mathsf{f}}}{\rightsquigarrow} (\mathcal{C}_3,\mathcal{I}) \ldots \overset{\mathsf{R}_{\mathsf{f}}}{\rightsquigarrow} (\mathcal{C}_\ell,\mathcal{I})$$

we obtain that for every constraint $T \overset{?}{\vdash} u \in \mathcal{C}_\ell$ with u not a variable, there exists a simple proof Δ of $T\theta' \cup \mathcal{I} \vdash u\theta'$ which ends with a decomposition. Furthermore, θ' is a strongly non-confusing solution of $(\mathcal{C}_\ell, \mathcal{I})$. Then, by applying Lemma 2.3.10 recursively, we have a derivation

$$(\mathcal{C}_\ell, \mathcal{I}) \overset{\mathsf{R_{ax}}}{\rightsquigarrow} \dots \overset{\mathsf{R_{ax}}}{\rightsquigarrow} (\mathcal{C}', \mathcal{I})$$

such that $(\mathcal{C}', \mathcal{I})$ is in solved form and θ' is a strongly non-confusing solution of $(\mathcal{C}', \mathcal{I})$.

To sum up, there is a derivation $(\mathcal{C}, \mathcal{I}) \rightsquigarrow_\sigma (\mathcal{C}', \mathcal{I})$ following strategy $\mathcal{S}$ such that $\theta = \theta' \circ \sigma$ and θ' is a solution of both $(\mathcal{C}', \mathcal{I})$. □

2.4 Characterization of solutions

Showing that solving constraint systems can be reduced to solving solved constraint systems has been done in [CLCZ10]. Our result enables us to furthermore reduce the search for solutions to non-confusing solutions. This is interesting because, for any non-confusing solution (which represents an execution trace), any term of the attacker knowledge may be obtained by composition only.

Definition 2.4.1. *We associate to each set of terms T the set of subterms of T that may be deduced from $T \cup vars(T)$: $Sat_v(T) = \{u \in st(T) \mid T \cup vars(T) \vdash u\}$*

Notice that in the case of solved constraint systems, the variables occurring in T are deducible.

Proposition 2.4.1 states that it is possible to compute from a solved constraint system, a "basis" $Sat_v(T)$ from which all deducible terms can be obtained applying only composition rules. This follows the spirit of [AC06] but now in the active case.

Proposition 2.4.1. *Let $(\mathcal{C}, \mathcal{I})$ be a constraint system in solved form, θ be a non-confusing solution of $(\mathcal{C}, \mathcal{I})$, T be a left-hand side of a constraint in $\mathcal{C}$ and u be a term such that $T\theta \cup \mathcal{I} \vdash u$. We have that $Sat_v(T)\theta \cup \mathcal{I} \vdash u$ by using composition rules only.*

Proof. Consider a simple proof Δ of $T_i\theta \cup \mathcal{I} \vdash u$. We show by induction on $(i, |\Delta|)$ that there exists a proof Δ' of $Sat_v(T_i)\theta \cup \mathcal{I} \vdash u$ that uses composition rules only. We distinguish several cases depending on the last rule of Δ:

The last rule is an axiom. Then $u \in T_i\theta \cup \mathcal{I}$ and there is $t \in T_i \cup \mathcal{I}$ such that $u = t\theta$. The property immediately follows.

The last rule is a symmetric decryption rule:

$$\frac{\mathsf{senc}(u, v) \quad v}{u}$$

Let Δ_1 be the subproof of Δ whose root is labelled with $\mathsf{senc}(u, v)$. By induction hypothesis, there exists a proof Δ_1' of $Sat_v(T_i)\theta \cup \mathcal{I} \vdash \mathsf{senc}(u, v)$ that uses composition rules only. Let j be the minimal index such that $Sat_v(T_j)\theta \cup \mathcal{I} \vdash \mathsf{senc}(u, v)$ with a proof that uses composition rules only. We distinguish two cases.

Either Δ_1' ends with a symmetric encryption rule. Let Δ' be the direct subproof of Δ_1' whose root is labelled with u. We have that Δ' is a proof of $Sat_v(T_j)\theta \cup \mathcal{I} \vdash u$ that uses composition rules only.

Otherwise, Δ'_1 is reduced to an axiom. In such a case, there exists $u_1 \in Sat_v(T_j) \cup \mathcal{I}$ such that $u_1\theta = \mathsf{senc}(u,v)$. If u_1 was a variable, there would exist $k < j$ such that $(T_k \overset{?}{\vdash} u_1) \in \mathcal{C}$, and so $T_k\theta \cup \mathcal{I} \vdash \mathsf{senc}(u,v)$. Since Δ_1 is a simple proof of $T_i\theta \cup \mathcal{I} \vdash \mathsf{senc}(u,v)$, it is also a simple proof of $T_k\theta \cup \mathcal{I} \vdash \mathsf{senc}(u,v)$. By applying our induction hypothesis, we deduce that there exists a proof of $Sat_v(T_k)\theta \cup \mathcal{I} \vdash \mathsf{senc}(u,v)$ that uses composition rules only. This contradicts the minimality of j. Hence, u_1 is not a variable. Consequently, there exist t_1, t_2 such that $u_1 = \mathsf{senc}(t_1,t_2)$, with $t_1\theta = u$ and $t_2\theta = v$. We are thus in the case where $\mathsf{senc}(t_1,t_2) \in Sat_v(T_i) \cup \mathcal{I}$.

We want to show that $t_2 \in Sat_v(T_i) \cup \mathcal{I}$ and we already know that $t_2 \in st(T_i) \cup \mathcal{I}$. Thus, it only remains to show that $T_i \cup var(T_i) \cup \mathcal{I} \vdash t_2$. By induction hypothesis, we know that there exists a proof Δ'_2 of $Sat_v(T_i)\theta \cup \mathcal{I} \vdash v$ that uses composition rules only. Furthermore, $v = t_2\theta$ and θ is non-confusing. We show by induction on t_2 that, if there exists a proof Δ'_2 of $Sat_v(T_i)\theta \cup \mathcal{I} \vdash t_2\theta$ that uses composition rules only, and $t_2 \in st(T_i) \cup \mathcal{I}$, then $t_2 \in Sat_v(T_i) \cup \mathcal{I}$.

- if $t_2 = x \in \mathcal{X}$, then $x \in var(T_i)$, and by definition of $Sat_v(T_i) \cup \mathcal{I}$, $t_2 \in Sat_v(T_i) \cup \mathcal{I}$.
- if $t_2 = \mathsf{f}(t'_1, \dots, t'_n)$ then we make a case distinction depending on Δ'_2:
  - Δ'_2 *is reduced to an axiom.* In such a case, there exists $v_1 \in Sat_v(T_i) \cup \mathcal{I}$ such that $t_2\theta = v_1\theta$. Since $t_2\theta$ is headed with f, we deduce that $v_1 \in st(T_i)$. As θ is non-confusing, $t_2\theta = v_1\theta$ implies that $t_2 = v_1 \in Sat_v(T_i)$.
  - Δ'_2 *ends with a composition rule.* Let $\pi_1, \dots, \pi_n$ be the direct subproofs of Δ'_2. We know that each π_j is a proof of $Sat_v(T_i) \cup \mathcal{I} \vdash t'_j\theta$ that uses composition rules only. By induction hypothesis, we deduce that $t'_j \in Sat_v(T_i) \cup \mathcal{I}$. Hence, $T_i \cup var(T_i) \cup \mathcal{I} \vdash t_2$ by applying the composition rule associated with function symbol f. As $t_2 \in st(T_i) \cup \mathcal{I}$, it follows that $t_2 \in Sat_v(T_i) \cup \mathcal{I}$.

We have shown that $\mathsf{senc}(t_1,t_2)$ and t_2 are in $Sat_v(T_i) \cup \mathcal{I}$. Hence, we easily conclude that $t_1 \in Sat_v(T_i) \cup \mathcal{I}$. Since $t_1\theta = u$, this allows us to conclude by considering a simple proof Δ' reduced to an axiom rule.

The last rule is an asymmetric decryption rule:

$$\frac{\mathsf{aenc}(u, \mathsf{pub}(v)) \quad \mathsf{priv}(v)}{u}$$

Let Δ_1 be the subproof of Δ whose root is labelled with $\mathsf{aenc}(u, \mathsf{pub}(v))$. By induction hypothesis, there exists a proof Δ'_1 of $Sat_v(T_i)\theta \cup \mathcal{I} \vdash \mathsf{aenc}(u, \mathsf{pub}(v))$ that uses composition rules only. Let j be the minimal indice such that $Sat_v(T_j)\theta \cup \mathcal{I} \vdash \mathsf{aenc}(u, \mathsf{pub}(v))$ with a proof that uses composition rules only. We distinguish two cases.

Either Δ'_1 ends with an asymmetric encryption rule. Let Δ' be the direct subproof of Δ'_1 whose root is labelled with u. We have that Δ' is a proof of $Sat_v(T_j)\theta \cup \mathcal{I} \vdash u$ that uses composition rules only.

Otherwise, Δ'_1 is reduced to an axiom. In such a case, there exists $u_1 \in Sat_v(T_j) \cup \mathcal{I}$ such that $u_1\theta = \mathsf{aenc}(u, \mathsf{pub}(v))$. If u_1 was a variable, there would exist $k < j$ such that $(T_k \overset{?}{\vdash} u_1) \in \mathcal{C}$, and so $T_k\theta \cup \mathcal{I} \vdash \mathsf{aenc}(u, \mathsf{pub}(v))$. Since Δ_1 is a simple proof of $T_i\theta \cup \mathcal{I} \vdash \mathsf{aenc}(u, \mathsf{pub}(v))$, it is also a simple proof of $T_k\theta \cup \mathcal{I} \vdash \mathsf{aenc}(u, \mathsf{pub}(v))$. By applying our induction hypothesis, we deduce that there exists a proof of $Sat_v(T_k)\theta \cup \mathcal{I} \vdash \mathsf{aenc}(u, \mathsf{pub}(v))$ that uses composition rules only. This contradicts the minimality of j. Hence u_1 is not a variable. Consequently, there exist t_1, t_2 such that $u_1 = \mathsf{aenc}(t_1, t_2)$ with $t_1\theta = u$ and $t_2\theta = \mathsf{pub}(v)$. We are thus in the case where $\mathsf{aenc}(t_1, t_2) \in Sat_v(T_i) \cup \mathcal{I}$.

Now, we show that t_2 is not a variable. By contradiction, assume that t_2 is a variable. In such a case, there exists $k < i$ such that $(T_k \overset{?}{\vdash} t_2) \in \mathcal{C}$, and so $T_k\theta \cup \mathcal{I} \vdash t_2\theta = \mathsf{pub}(v)$, and there is a simple proof witnessing this fact. This proof is either reduced to a leaf or ends with a decomposition. Thus, thanks to Lemma 2.3.8, there exists $t'_2 \in st(T_k) \smallsetminus \mathcal{X}$ such that $t'_2\theta = \mathsf{pub}(v)$. Hence, we have that $t'_2 = \mathsf{pub}(t'_3)$. We have also that $t'_2\theta = t_2\theta$. Since θ is non-confusing, we deduce that $t_2 = t'_2$.

Now, we want to show that $\mathsf{priv}(t'_3) \in Sat_v(T_i) \cup \mathcal{I}$. Let Δ_2 be the subproof of Δ whose root is labelled with $\mathsf{priv}(v)$. By applying our induction hypothesis, we deduce that there exists Δ'_2 of $Sat_v(T_i)\theta \cup \mathcal{I} \vdash \mathsf{priv}(v)$ that uses composition rules only. Actually, we necessarily have that $\mathsf{priv}(v) \in Sat_v(T_i)\theta$, *i.e.* there exists $w \in Sat_v(T_i)$ such that $\mathsf{priv}(v) = w\theta$. Moreover, we know that Δ_2 is a simple proof that is either reduced to a leaf or that ends with a decomposition. Hence, we can apply Lemma 2.3.8. We deduce that there exists $t'_4 \in st(T_i) \smallsetminus \mathcal{X}$ such that $t'_4\theta = \mathsf{priv}(v)$. Hence, we have that $t'_4 = \mathsf{priv}(t'_5)$. Moreover, we have that $t'_3\theta = t'_5\theta$. Since θ is non-confusing, we deduce that $t'_3 = t'_5$. Lastly, we have that $w\theta = t'_4\theta$. Since θ is non-confusing, we deduce that $w = t'_4$. Hence, we have that $w = t'_4 = \mathsf{priv}(t'_5) = \mathsf{priv}(t'_3)$, and thus $\mathsf{priv}(t'_3) \in Sat_v(T_i) \cup \mathcal{I}$.

Hence, we have that $\mathsf{aenc}(t_1, \mathsf{pub}(t'_3)) \in Sat_v(T_i) \cup \mathcal{I}$, and $\mathsf{priv}(t'_3) \in Sat_v(T_i) \cup \mathcal{I}$. Hence, we easily conclude that $t_1 \in Sat_v(T_i) \cup \mathcal{I}$. Since $t_1\theta = u$, this allows us to conclude by considering a simple proof Δ' reduced to an axiom rule.

A similar reasoning holds for our optional rule.

The last rule is another decomposition rule: We can apply a similar reasoning as in the case of the decryption rule. By symmetry, consider a rule of the form

$$\frac{\mathsf{f}(u, v)}{u} \qquad \text{with } \mathsf{f} \in \{\langle\rangle, ::\}$$

Let Δ_1 be the subproof of Δ whose root is labelled with $\mathsf{f}(u, v)$. By induction hypothesis, there exists a proof Δ'_1 of $Sat_v(T_i)\theta \cup \mathcal{I} \vdash \mathsf{f}(u, v)$ that uses composition rules only. Let j be the minimal indice such that $Sat_v(T_j)\theta \cup \mathcal{I} \vdash \mathsf{f}(u, v)$ with a proof that uses composition rules only. We distinguish two cases:

Either Δ'_1 ends with a composition rule. Let Δ' be the direct subproof of Δ'_1 whose root is labelled with u. We have that Δ' is a proof of $Sat_v(T_j)\theta \cup \mathcal{I} \vdash u$ that uses composition rules only.

Otherwise, Δ'_1 is reduced to an axiom. In such a case, there exists $u_1 \in Sat_v(T_j) \cup \mathcal{I}$ such that $u_1\theta = \mathsf{f}(u, v)$. If u_1 was a variable, there would exist $k < j$ such that $(T_k \overset{?}{\vdash} u_1) \in \mathcal{C}$, and so $T_k\theta \cup \mathcal{I} \vdash \mathsf{f}(u, v)$. Since Δ_1 is a simple proof of $T_i\theta \cup \mathcal{I} \vdash \mathsf{f}(u, v)$. By applying our induction hypothesis, we deduce that there exists a proof of $Sat_v(T_k)\theta \cup \mathcal{I} \vdash \mathsf{f}(u, v)$ that uses composition rules only. This contradicts the minimality of j. Hence, u_1 is not a variable. Consequently, there exist t_1, t_2 such that $u_1 = \mathsf{f}(t_1, t_2)$, with $t_1\theta = u$ and $t_2\theta = v$. We are thus in the case where $\mathsf{f}(t_1, t_2) \in Sat_v(T_i) \cup \mathcal{I}$. Hence, we easily conclude that $t_1 \in Sat_v(T_i) \cup \mathcal{I}$.

The last rule is a composition rule:

$$\frac{v_1 \quad \dots \quad v_n}{\mathsf{f}(v_1, \dots, v_n)}$$

By induction hypothesis, we know that for $1 \leq k \leq n$, there exist Δ'_k a proof of $Sat_v(T_i)\theta \cup \mathcal{I} \vdash v_k$ that uses composition rules only. Let Δ' be the proof consisting of applying the composition rule associated to the symbol f on $\Delta'_1, \dots, \Delta'_k$. It immediately follows that Δ' is a proof of $Sat_v(T_i)\theta \cup \mathcal{I} \vdash f(v_1, \dots, v_n)$ that uses composition rules only. Hence the result.

□

2.5 Conclusion and future prospects

We use a symbolic model with a large signature, encompassing symmetric and asymmetric encryption, signature, hashes, and lists. Moreover, we provide the intruder with an infinite set of names that he can use however he wants. We have defined in this setting the notion of constraint systems to model the execution of the protocol. Using simplification rules, we have obtained a complete symbolic representation of the knowledge of the attacker: when considering non-confusing solutions, any term of the intruder knowledge may be obtained by composition only from a clearly defined set of terms.

In order to model more protocols, we could extend this characterization result to a wider signature. For instance, for local inference systems, an algorithm in [BDC09] gives a representation of solutions. Trying to pinpoint the properties of the deduction system representing the intruder capabilities that are needed in order to keep our characterization result would allow to add new primitives to the signature automatically, without having to prove everything from scratch each time a new primitive is needed. We would like to discover conditions on the signature or on the deduction system that allow to generalize the result so that it holds for a family of signatures. Intuitively, these conditions would be linked to the deduction system, and the relations between composition and decomposition rules.

Chapter 3

Modeling and analyzing routing protocols

A standard method to model protocols is to use a process calculus [AG97]. Process calculi are particularly well adapted to model interactions between independent agents. But ad hoc networks have particularities that can not be modeled in the standard way, most notably the communication medium. Indeed, in such a network, nodes can only send messages to nodes that are situated within a certain distance of them. So, in order to properly model ad hoc routing protocol with a process calculus, the communication model has to be adapted.

Furthermore, the standard intruder model used in protocol verification, the so-called Dolev-Yao intruder [DY81], is not well suited for the study of such protocols. The intruder in the Dolev Yao model controls the network: he can overhear every communication, modify or delete any message. This is far too strong an assumption in a wireless setting. A more reasonable assumption regarding his ability is to limit the scope of his actions: for instance, he can only overhear messages sent by nodes that are situated near enough, and he can not prevent the reception of a specific message.

As our goal is to model ad hoc routing protocols, we have to take the particularities of these protocols into account. Few other formal approaches have been proposed to achieve this goal. Nanz and Hankin [NH06] propose a process calculus to model the network topology and broadcast communications. They also provide a decision procedure for an intruder that is already specified by the user. This allows to check security against fixed, known in advance scenarios. The model we propose here is inspired from their work. We add a logic for specifying the tests performed at each step by the nodes on the current route and to specify the security properties.

In this chapter, we will first propose in Section 3.1 a way to model protocols in ad hoc networks. It is a process calculus with an underlying graph that represents the links in the network. Then we will explain in Section 3.2 how to abstract some parts in the execution of the protocol in order to get a finite number of representations of the possible runs. Finally, we will show in Section 3.3 how to bound the lists appearing in messages, in order to bound in turn the size of messages exchanged. This allows us to provide two NP decision procedures in Section 3.4 for analyzing routing protocols for a bounded number of sessions.

3.1 Modeling

Recall that we consider an infinite set of *names* $\mathcal{N}$. We state here some of the names we will use in the course of the chapter: $\mathcal{N} = \{\mathsf{rep}, \mathsf{req}, N, K, K_a, A, S, D, \ldots\}$. Furthermore, we will consider a special set of names $\mathcal{N}_{\mathsf{loc}}$, which represents the nodes of the network. The lists we use in this chapter are lists of names in $\mathcal{N}_{\mathsf{loc}}$, they typically represent a route in the network. We assume that the intruder has access to $\mathcal{N}_{\mathsf{loc}}$.

3.1.1 Process calculus

Security protocols are typically defined by the roles of the agents participating in the protocol. A role is a sequence of actions that the agent must accomplish to execute the protocol. Process algebras are well suited to model security protocols specified by roles, with each role represented by a distinct process.

Several calculi already exist to model security protocols (e.g. [AG97, AF01]). However, for our purpose, a node, i.e. a process, has to perform some specific actions that can not be easily modeled in such calculi. For instance, a node stores some information, e.g. the content of its routing table. We also need to take into account the network topology and to model broadcast communication. Such features can not be easily modeled in these calculi.

Actually, our calculus is inspired from CBS#, a process calculus introduced in [NH06], which allows mobile wireless networks and their security properties to be formally described and analyzed. However, we extend this calculus to allow nodes to perform some sanity checks on the routes they receive, such as checking neighborhood properties.

The intended behavior of each node of the network can be modeled by a *process* defined by the grammar given in Figure 3.1. Our calculus is parametrized by a set $\mathcal{L}$ of formulas.

$P, Q ::=$	Processes
0	null process
$\mathsf{out}(u).P$	emission
$\mathsf{in}\ u[\Phi].P$	reception, $\Phi \in \mathcal{L}$
$\mathsf{store}(u).P$	storage
$\mathsf{read}\ u\ \mathsf{then}\ P\ \mathsf{else}\ Q$	reading
$\mathsf{if}\ \Phi\ \mathsf{then}\ P\ \mathsf{else}\ Q$	conditional, $\Phi \in \mathcal{L}$
$P \mid Q$	parallel composition
$!P$	replication
$\mathsf{new}\ m.P$	fresh name generation

Figure 3.1: Process grammar.

First, we describe the constructions specific to our calculus. The process $\mathsf{out}(u).P$ emits u and then behaves like P. The process $\mathsf{in}\ u[\Phi].P$ expects a message m matching the pattern u and such that Φ is true. It then behaves like $P\sigma$ where $\sigma = mgu(m, u)$. If Φ is the true formula, we simply write $\mathsf{in}\ u.P$. The process $\mathsf{store}(u).P$ stores u in the storage list of the node executing the process and then behaves like P. The process $\mathsf{read}\ u\ \mathsf{then}\ P\ \mathsf{else}\ Q$ looks for a message matching the pattern u in the storage list of the node executing the process. Then, if such an element m is found, it behaves like $P\sigma$ where $\sigma = mgu(m, u)$. If no element of the form u is found, it behaves like Q. For the sake of clarity, we will omit the else part when $Q = 0$.

The other entries in the grammar are the usual ones. The process $\mathsf{if}\ \Phi\ \mathsf{then}\ P\ \mathsf{else}\ Q$ tests whether Φ is true. If Φ is true, it then behaves like P. Else, it behaves like Q. For the sake of clarity, we will again omit the else part when $Q = 0$. The process $P \mid Q$ allows computation in process P and Q to happen independently. The process $!P$ allows to use process P as many times as we want. (Note that in the general case, allowing replication leads to undecidability) The process $\mathsf{new}\ m.P$ creates a fresh nonce m and then behaves like P.

Sometimes, for the sake of clarity, we will omit the null process.

We write $fv(P)$ (respectively, $bv(P)$) for the set of free (respectively, bound) variables of P. A process P is *ground* when $fv(P) = \emptyset$.

The store and read primitives are particularly important when modeling routing protocols, in order to avoid multiple answers to a single request or to allow nodes to store and retrieve already known routes. These primitives can also be used to represent other classes of protocols, where a global state is assumed for each agent, in order to store some information (black list, already used keys. *etc.*) throughout the sessions.

Secured routing protocols require the agents participating in the protocol to perform some checks on the part of the messages they receive that is supposed to represent a route or part of a route. For instance, they may check that the list they receive begins with the name of the neighbor who sent them the message, to test whether the message was processed correctly. These verifications rely partly on *neighborhood discovery*, which is a protocol run by the nodes before executing the routing protocol. The aim of a node running such a protocol is to discover which nodes are within his reach, and are thus neighbors in the underlying graph representing the network. In order to get a secure routing protocol, the neighborhood discovery protocol needs to be correct [PPS+08, PPH08]. We assume that a secure neighborhood discovery protocol has been used, consequently, each node can check whether a given node is one of his neighbors. We express these checks thanks to a logic. Next is an example of such a logic.

Example 3.1.1. *We will typically consider the logic $\mathcal{L}_{\mathsf{route}}$ defined by the following grammar:*

$\Phi ::=$	*Formula*
$\mathsf{check}(a, b)$	*neighborhood of two nodes*
$\mathsf{checkl}(c, l)$	*local neighborhood of a node in a list*
$\mathsf{route}(l)$	*validity of a route*
$\mathsf{loop}(l)$	*existence of a loop in a list*
$\Phi_1 \wedge \Phi_2$	*conjunction*
$\Phi_1 \vee \Phi_2$	*disjunction*
$\neg\Phi$	*negation*

Given an undirected graph $G = (V, E)$ with $V \subseteq \mathcal{N}_{\mathsf{loc}}$, the semantics $[\![\Phi]\!]_G$ of a formula $\Phi \in \mathcal{L}_{\mathsf{route}}$ is recursively defined by:

- $[\![\mathsf{check}(a, b)]\!]_G = 1$ *iff* $(a, b) \in E$,
- $[\![\mathsf{checkl}(c, l)]\!]_G = 1$ *iff l is of sort* lists, *c appears exactly once in l, and for any sub-list l' of l,*
  - *if* $l' = a :: c :: l_1$, *then* $(a, c) \in E$.
  - *if* $l' = c :: b :: l_1$, *then* $(b, c) \in E$.

- $[\![\mathsf{route}(l)]\!]_G = 1$ *iff* l *is of sort* lists, $l = a_1 :: \ldots :: a_n$, *for every* $1 \leq i < n$, $(a_i, a_{i+1}) \in E$, *and for every* $1 \leq i, j \leq n$, $i \neq j$ *implies that* $a_i \neq a_j$.
- $[\![\mathsf{loop}(l)]\!]_G = 1$ *iff* l *is of sort* lists *and there exists an element appearing at least twice in* l,
- $[\![\Phi_1 \wedge \Phi_2]\!]_G = [\![\Phi_1]\!]_G \wedge [\![\Phi_2]\!]_G$, $[\![\neg\Phi]\!]_G = \neg[\![\Phi]\!]_G$, *and* $[\![\Phi_1 \vee \Phi_2]\!]_G = [\![\Phi_1]\!]_G \vee [\![\Phi_2]\!]_G$.

Intuitively, $\mathsf{check}(a, b)$ *is true if the agents* a *and* b *are neighbors in the network.* $\mathsf{checkl}(c, l)$ *is true if from the point of view of* c*, the list* l *could be a valid route that goes through* c*, i.e. if the list* l *contains one occurrence of* c *between two neighbours of* c*.* $\mathsf{route}(l)$ *is true if the list* l *represents a valid path in the graph that does not go through the same node twice.* $\mathsf{loop}(l)$ *is true if the list* l *contains twice the same element. (We usually want to test for loop-free lists). The other entries are the usual ones:* $\Phi_1 \wedge \Phi_2$ *is true if* Φ_1 *and* Φ_2 *are true,* $\Phi_1 \vee \Phi_2$ *is true if* Φ_1 *or* Φ_2 *is true, and* $\neg\Phi$ *is true if* Φ *is false.*

3.1.2 Example: modeling the SRP protocol

We consider SRP introduced in [PH02], assuming that each node already knows his neighbors (running e.g. some neighbor discovery protocol). We model here its application to the DSR protocol [JMB01].

Consider the signature given in Example 2.1.1 and let $S, D, \mathsf{req}, \mathsf{rep}, Id, K_{SD}$ be names ($S, D \in \mathcal{N}_{\mathsf{loc}}$) and x_L be a variable of sort lists. The process executed by a node S initiating the search for a route towards a node D is:

$$P_{init}(S, D) = \mathsf{new}\ Id.\mathsf{out}(u_1).\mathsf{in}\ u_2[\Phi_S].0$$

where:

$$\begin{array}{l} u_1 = \langle \mathsf{req}, S, D, Id, S :: \bot, \mathsf{hmac}(\langle \mathsf{req}, S, D, id\rangle, K_{SD})\rangle \\ u_2 = \langle \mathsf{rep}, D, S, Id, x_L, \mathsf{hmac}(\langle \mathsf{rep}, D, S, id, x_L\rangle, K_{SD})\rangle \\ \Phi_S = \mathsf{checkl}(S, x_L) \wedge \neg\mathsf{loop}(x_L). \end{array}$$

The names of the intermediate nodes are accumulated in the route request packet. Intermediate nodes relay the request over the network, except if they have already seen it. An intermediate node also checks that the received request is locally correct by verifying whether the head of the list in the request is one of its neighbors. Below, $V \in \mathcal{N}_{\mathsf{loc}}$, x_S, x_D and x_a are variables of sort loc whereas x_r is a variable of sort lists and x_{Id}, x_m are variables of sort terms. The process executed by an intermediate node V when forwarding a request is as follows:

$$P_{\mathsf{req}}(V) = \mathsf{in}\ w_1[\Phi_V].\mathsf{read}\ t\ \mathsf{then}\ 0\ \mathsf{else}\ (\mathsf{store}(t).\mathsf{out}(w_2))$$

$$\text{where } \left\{ \begin{array}{l} w_1 = \langle \mathsf{req}, x_S, x_D, x_{Id}, x_a :: x_r, x_m\rangle \\ \Phi_V = \mathsf{check}(V, x_a) \\ t = \langle x_S, x_D, x_{Id}\rangle \\ w_2 = \langle \mathsf{req}, x_S, x_D, x_{Id}, V :: (x_a :: x_r), x_m\rangle \end{array} \right.$$

When the request reaches the destination D, it checks that the request has a correct hmac and that the first node in the route is one of his neighbors. Then, the destination D constructs a route reply, in particular it computes a new hmac over the route accumulated in the request packet with K_{SD}, and sends the answer back over the network.

The process executed by the destination node D is the following:

$$P_{dest}(D,S) = \mathsf{in}\ v_1[\Phi_D].\mathsf{out}(v_2).0$$

$$\text{where: } \begin{cases} v_1 = \langle \mathsf{req}, S, D, x_{Id}, x_a :: x_l, \mathsf{hmac}(\langle \mathsf{req}, S, D, x_{Id}\rangle, K_{SD})\rangle \\ \Phi_D = \mathsf{check}(D, x_a) \\ v_2 = \langle \mathsf{rep}, D, S, x_{Id}, x_a :: x_l, \mathsf{hmac}(\langle \mathsf{rep}, D, S, x_{Id}, x_a :: x_l\rangle, K_{SD})\rangle \end{cases}$$

Then, the reply travels along the route back to S. The intermediate nodes check that the route in the reply packet is locally correct, which means that their name appears once in the list and that the names appearing just before and just after in the list are the names of some of their neighbors. If this test is passed successfully, they forward the reply. The process executed by an intermediary node V when forwarding a reply is the following:

$$P_{\mathsf{rep}}(V) = \mathsf{in}\ w'[\Phi'_V].\mathsf{out}(w')$$

$$\text{where } \begin{cases} w' = \langle \mathsf{rep}, x_D, x_S, x_{Id}, x_r, x_m\rangle \\ \Phi'_V = \mathsf{checkl}(V, x_r) \end{cases}$$

3.1.3 Execution model

Each process is located at a specified node of the network. Unlike classical Dolev-Yao model, the intruder does not control the entire network but can only interact with its neighbors. More specifically, we assume that the topology of the network is represented by giving an undirected graph $G = (V, E)$ with $V \subseteq \mathcal{N}_{\mathsf{loc}}$, where an edge in the graph models the fact that two nodes are neighbors. We also assume that we have a set of nodes $\mathcal{M}$ that are controlled by the attacker. These nodes are then called *malicious*. Our model is not restricted to a single malicious node. Our results allow us to consider the case of several compromised nodes that collaborate by sharing their knowledge. However, it is well-known that the presence of several colluding malicious nodes often yields straightforward attacks [HPJ06, LPM+05].

Definition 3.1.1 (configuration). *A (ground)* concrete configuration *of the network is a triplet $(\mathcal{P}; \mathcal{S}; \mathcal{I})$ where:*

- *$\mathcal{P}$ is a multiset of expressions of the form $\lfloor P \rfloor_n$, which represents the (ground) process P located at node $n \in V$.*
- *$\mathcal{S}$ is a set of expressions of the form $\lfloor t \rfloor_n$ with $n \in V$ and t a ground term. $\lfloor t \rfloor_n$ represents the fact that the node n has stored the term t.*
- *$\mathcal{I}$ is a set of ground terms representing the messages seen by the intruder.*

In the expressions of the form $\lfloor P \rfloor_n$, we consider for the sake of clarity that null processes, i.e. expressions of the form $\lfloor 0 \rfloor_n$, are removed. Moreover, we will write $\lfloor P \rfloor_n \cup \mathcal{P}$ instead of $\{\lfloor P \rfloor_n\} \cup \mathcal{P}$.

Example 3.1.2. *Continuing our modeling of* SRP*, a typical initial configuration for the* SRP *protocol is*

$$K_0 = (\lfloor P_{init}(S,D) \rfloor_S \mid \lfloor P_{dest}(D,S) \rfloor_D; \emptyset; \mathcal{I}_0)$$

where both the source node S and the destination node D wish to communicate. We assume that each node has an empty storage list and that the initial knowledge of the intruder is given *by an infinite set of terms $\mathcal{I}_0$. A possible network configuration is modeled by the graph G_0 in Figure 3.2. We assume that there is a single malicious node, i.e. $\mathcal{M}_0 = \{n_I\}$. The nodes W and X are two extra (honest) nodes. We do not assume that the intermediary nodes n_I, W, and X execute the routing protocol. Actually, this is not needed to show that the protocol is flawed, and we want to keep this example as simple as possible.*

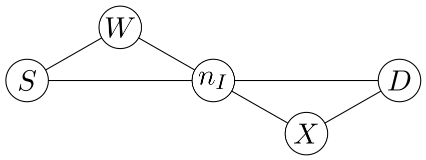


Figure 3.2: Example of network topology (where n_I is the malicious node).

Each honest node broadcasts its messages to all his neighbors. To capture more malicious behaviors, we allow the nodes controlled by the intruder to send messages only to some specific neighbor. The communication system is formally defined by the rules of Figure 3.3. They are parametrized by the underlying graph G and the set of malicious nodes $\mathcal{M}$.

The relation $\rightarrow^*_{G,\mathcal{M}}$ is the reflexive and transitive closure of $\rightarrow_{G,\mathcal{M}}$. We may write $\rightarrow$, $\rightarrow_G$, $\rightarrow_{\mathcal{M}}$ instead of $\rightarrow_{G,\mathcal{M}}$ when the underlying network topology G or the underlying set $\mathcal{M}$ is clear from the context.

Note that in the case where we assume that there is a single malicious node with each honest node connected to it, we retrieve the model where the attacker is assumed to control all the communications.

Example 3.1.3. *Continuing the example developed in Section 3.1.2, the following sequence of transitions is enabled from the initial configuration K_0:*

$$K_0 \rightarrow^*_{G_0,\mathcal{M}_0} (\lfloor \mathsf{in}\ u_2[\Phi_S].0 \rfloor_S \cup \lfloor P_{dest}(D,S) \rfloor_D ; \emptyset ; \mathcal{I}_0 \cup \{u_1\})$$

where:

$$\begin{array}{l} u_1 = \langle \mathsf{req}, S, D, Id, S :: \bot, \mathsf{hmac}(\langle \mathsf{req}, S, D, Id \rangle, K_{SD}) \rangle \\ u_2 = \langle \mathsf{rep}, D, S, Id, x_L, \mathsf{hmac}(\langle \mathsf{rep}, D, S, Id, x_L \rangle, K_{SD}) \rangle \\ \Phi_S = \mathsf{checkl}(S, x_L) \wedge \neg \mathsf{loop}(x_L) \end{array}$$

During this transition, S broadcasts to its neighbors a request to find a route to D. The intruder n_I is a neighbor of S in G_0, so he learns the request message. Assuming that the intruder knows the names of its neighbors, i.e. $W, X \in \mathcal{I}_0$, he can then build the following fake message request:

$$m = \langle \mathsf{req}, S, D, Id, [X; W; S], \mathsf{hmac}(\langle \mathsf{req}, S, D, Id \rangle, K_{SD}) \rangle$$

and broadcasts it. Since $(X, D) \in E$, D accepts this message and the resulting configuration of the transition is

$$(\lfloor \mathsf{in}\ u_2[\Phi_S].0 \rfloor_S \cup \lfloor \mathsf{out}(v_2\sigma).0 \rfloor_D ; \emptyset ; \mathcal{I}_0 \cup \{u_1\})$$

$$\textit{where} \quad \left\{ \begin{array}{l} v_2 = \langle \mathsf{rep}, D, S, x_{Id}, x_a :: x_l, \mathsf{hmac}(\langle D, S, x_{Id}, x_a :: \rangle, K_{SD}) \rangle \\ \sigma = \{x_{Id} \mapsto Id, x_a \mapsto X, \mapsto [W; S]\} \end{array} \right.$$

Comm $(\{\lfloor \mathsf{in}\ u_j[\Phi_j].P_j\rfloor_{n_j} \mid mgu(t,u_j)\neq\perp, [\![\Phi_j\sigma_j]\!]_G = 1, (n,n_j)\in E\}$
$\cup \lfloor \mathsf{out}(t).P\rfloor_n \cup \mathcal{P};\mathcal{S};\mathcal{I})$
$\rightarrow_{G,\mathcal{M}} \quad (\{\lfloor P_j\sigma_j\rfloor_{n_j}\}\cup\lfloor P\rfloor_n\cup\mathcal{P};\mathcal{S};\mathcal{I}')$

where $\sigma_j = mgu(t,u_j)$, $\mathcal{I}' = \mathcal{I}\cup\{t\}$ if $(n,n_I)\in E$ for some $n_I\in\mathcal{M}$ and $\mathcal{I}'=\mathcal{I}$ otherwise. Moreover, $\lfloor P'\rfloor_{n'}\in\mathcal{P}$ implies that:

- $(n,n')\notin E$, or
- P' is not of the form $\mathsf{in}\ u'[\Phi'].Q'$, or
- $P' = \mathsf{in}\ u'[\Phi'].Q'$ and $(mgu(t,u')=\perp$ or $[\![\Phi' mgu(t,u')]\!]_G = 0)$.

In $(\lfloor \mathsf{in}\ u[\Phi].P\rfloor_n\cup\mathcal{P};\mathcal{S};\mathcal{I}) \rightarrow_{G,\mathcal{M}} (\lfloor P\sigma\rfloor_n\cup\mathcal{P};\mathcal{S};\mathcal{I})$
if $(n_I,n)\in E$ for some $n_I\in\mathcal{M}$, $\mathcal{I}\vdash t$, $\sigma = mgu(t,u)$ and $[\![\Phi\sigma]\!]_G = 1$

Store $(\lfloor \mathsf{store}(t).P\rfloor_n\cup\mathcal{P};\mathcal{S};\mathcal{I}) \rightarrow_{G,\mathcal{M}} (\lfloor P\rfloor_n\cup\mathcal{P};\lfloor t\rfloor_n\cup\mathcal{S};\mathcal{I})$

Read-Then $(\lfloor \mathsf{read}\ u\ \mathsf{then}\ P\ \mathsf{else}\ Q\rfloor_n\cup\mathcal{P};\lfloor t\rfloor_n\cup\mathcal{S};\mathcal{I})$
$\rightarrow_{G,\mathcal{M}} (\lfloor P\sigma\rfloor_n\cup\mathcal{P};\lfloor t\rfloor_n\cup\mathcal{S};\mathcal{I})$
where $\sigma = mgu(t,u)$

Read-Else $(\lfloor \mathsf{read}\ u\ \mathsf{then}\ P\ \mathsf{else}\ Q\rfloor_n\cup\mathcal{P};\mathcal{S};\mathcal{I})$
$\rightarrow_{G,\mathcal{M}} (\lfloor Q\rfloor_n\cup\mathcal{P};\mathcal{S};\mathcal{I})$
if for all t such that $\lfloor t\rfloor_n\in\mathcal{S}$, $mgu(t,u)=\perp$

If-Then $(\lfloor \mathsf{if}\ \Phi\ \mathsf{then}\ P\ \mathsf{else}\ Q\rfloor_n\cup\mathcal{P};\mathcal{S};\mathcal{I})$
$\rightarrow_{G,\mathcal{M}} (\lfloor P\rfloor_n\cup\mathcal{P};\mathcal{S};\mathcal{I})$ if $[\![\Phi]\!]_G = 1$

If-Else $(\lfloor \mathsf{if}\ \Phi\ \mathsf{then}\ P\ \mathsf{else}\ Q\rfloor_n\cup\mathcal{P};\mathcal{S};\mathcal{I})$
$\rightarrow_{G,\mathcal{M}} (\lfloor Q\rfloor_n\cup\mathcal{P};\mathcal{S};\mathcal{I})$ if $[\![\Phi]\!]_G = 0$

Par $(\lfloor P_1 \mid P_2\rfloor_n\cup\mathcal{P};\mathcal{S};\mathcal{I}) \rightarrow_{G,\mathcal{M}} (\lfloor P_1\rfloor_n\cup\lfloor P_2\rfloor_n\cup\mathcal{P};\mathcal{S};\mathcal{I})$

Repl $(\lfloor !P\rfloor_n\cup\mathcal{P};\mathcal{S};\mathcal{I}) \rightarrow_{G,\mathcal{M}} (\lfloor P\alpha\rfloor_n\cup\lfloor !P\rfloor_n\cup\mathcal{P};\mathcal{S};\mathcal{I})$
where α is a renaming of the bound variables of P

New $(\lfloor \mathsf{new}\ m.P\rfloor_n\cup\mathcal{P};\mathcal{S};\mathcal{I}) \rightarrow_{G,\mathcal{M}} (\lfloor P\{m\mapsto m'\}\rfloor_n\cup\mathcal{P};\mathcal{S};\mathcal{I})$
where m' is a fresh name

Figure 3.3: Concrete transition system.

As usual, an attack is defined as a reachability property.

Definition 3.1.2. *Let G be a graph and $\mathcal{M}$ be a set of nodes. There is an $\mathcal{M}$-*attack *on a configuration with a hole $(\mathcal{P}[\_];\mathcal{S};\mathcal{I})$ for the network topology G and the formula Φ if there exist $n, \mathcal{P}', \mathcal{S}', \mathcal{I}'$ such that:*

$$(\mathcal{P}[\mathsf{if}\ \Phi\ \mathsf{then}\ \mathsf{out}(\mathsf{error})];\mathcal{S};\mathcal{I}) \rightarrow^*_{G,\mathcal{M}} \ (\lfloor \mathsf{out}(\mathsf{error}) \rfloor_n \cup \mathcal{P}', \mathcal{S}', \mathcal{I}')$$

where error *is a special symbol not occurring in the configuration $(\mathcal{P}[\_];\mathcal{S};\mathcal{I})$.*

The usual secrecy property can be typically encoded by adding a witness process in parallel. For example, the process $W = \mathsf{in}\ s.\_$ can only evolve if it receives the secret s. Thus the secrecy preservation of s on a configuration $(\mathcal{P};\mathcal{S};\mathcal{I})$ for a graph $G = (V, E)$ can be defined by the (non) existence of an $\{n_I\}$-attack on the configuration $(\mathcal{P} \cup \lfloor W \rfloor_n;\mathcal{S};\mathcal{I})$ and the formula true for the graph $G' = (V \cup \{n\}, E \cup \{(n, n_I)\})$.

Example 3.1.4. *For the SRP protocol, the property we want to check is that the list of nodes obtained by the source through the protocol represents a path in the graph. We can easily encode this property by replacing the null process in $P_{init}(S, D)$ by a hole, and checking whether the formula $\neg\mathsf{route}(x_L)$ holds. Let $P'_{init}(S, D)$ be the resulting process.*

$$P'_{init}(S, D) = \mathsf{new}\ Id.\mathsf{out}(u_1).\mathsf{in}\ u_2[\Phi_S].P$$

where $P = \mathsf{if}\ \neg\mathsf{route}(x_L)\ \mathsf{then}\ \mathsf{out}(\mathsf{error})$. Then, we recover the attack mentioned in [BV04] with the topology G_0 given in Example 3.1.2, and from the initial configuration:

$$K'_0 = (\lfloor P'_{init}(S, D) \rfloor_S \mid \lfloor P_{dest}(D, S) \rfloor_D;\emptyset;\mathcal{I}_0).$$

Indeed, we have that:

$$\begin{array}{rl} K_0 & \rightarrow^* (\lfloor \mathsf{in}\ u_2[\Phi_S].P \rfloor_S \cup \lfloor \mathsf{out}(m').0 \rfloor_D;\emptyset;\mathcal{I}) \\ & \rightarrow (\lfloor \mathsf{in}\ u_2[\Phi_S].P \rfloor_S \cup \lfloor 0 \rfloor_D;\emptyset;\mathcal{I}') \\ & \rightarrow (\lfloor \mathsf{if} \neg \mathsf{route}([X;W;S])\ \mathsf{then}\ \mathsf{out}(\mathsf{error}) \rfloor_S;\emptyset;\mathcal{I}') \\ & \rightarrow (\lfloor \mathsf{out}(\mathsf{error}).0 \rfloor_S;\emptyset;\mathcal{I}') \end{array}$$

$$\textit{where} \begin{cases} m' = \langle \mathsf{rep}, D, S, Id, [X;W;S], \mathsf{hmac}(\langle D, S, Id, [X;W;S] \rangle, K_{SD}) \rangle \\ \mathcal{I} = \mathcal{I}_0 \cup \{u_1\}, \textit{and} \\ \mathcal{I}' = \mathcal{I}_0 \cup \{u_1\} \cup \{m'\}. \end{cases}$$

3.2 Symbolic model

It is difficult to directly reason with the transition system defined in Figure 3.3 since it is infinitely branching. Indeed, a potentially infinite number of distinct messages can be sent at each step by the intruder node. In fact, the messages that the intruder can send encompass any message that he is able to forge from his knowledge.

That is why it is often interesting to introduce a *symbolic* transition system where each intruder step is captured by a single rule (as in e.g. [ALV02]). This transition system will have to maintain some sort of control over the messages through the use of the constraint systems defined in Chapter 2. Furthermore, we will also have to consider formulas and disequality constraints to account for some of the requirements inherited from the concrete transition system.

As in [MS01, CLCZ10, RT01], groups of executions can be represented using constraint systems. However, compared to previous work, we have to add constraints in order to cope with the formulas that are checked upon the reception of a message and also in order to cope with generalized disequality tests for reflecting cases where agents reject messages of the wrong form. Indeed, since messages can be broadcasted to all neighbors, we need to determine for each message which agents will accept the message and which agents will not accept it.

Definition 3.2.1 (Disequality constraint)**.** *A* disequality constraint *is an expression of the form $\forall X. v \neq u$ where v, u are terms and X is a set of variables.*

Our disequality constraints are rather general: they do not simply allow to check that two terms are different ($u \neq v$), but they also allow to ensure that no unification was possible at a certain point of the execution. It is necessary to check this due to our broadcast primitive: the disequality constraint represents the fact that a message was not treated because it did not match the expected pattern.

To model the execution of a protocol in a symbolic way, we will use the constraint systems defined in Chapter 2. Moreover, we will also maintain a set of other constraints Ψ consisting in disequality constraints and a formula Φ of $\mathcal{L}$.

Definition 3.2.2. *Let $(\mathcal{C}, \mathcal{I})$ be a constraint system and $\Psi = \Phi_1 \wedge \Phi_2$ where $\Phi_1 \in \mathcal{L}$ and Φ_2 is a conjunction of disequality constraints, such that $fv(\Psi) \subseteq rvar(\mathcal{C})$ and $names(\Psi) \cap \mathcal{I} = \emptyset$. A* solution *to $(\mathcal{C}, \mathcal{I})$ and Ψ for a graph G is a ground substitution θ such that $dom(\theta) = rvar(\mathcal{C})$ and:*

- *$\mathcal{I}\theta \cup T\theta \vdash u\theta$ for all $T \overset{?}{\vdash} u \in \mathcal{C}$;*
- *for all $(\forall X. v \neq u) \in \Phi_2$, the terms $v\theta$ and $u\theta$ are not unifiable (even renaming the variables of X with fresh variables); and*
- *$[\![\Phi_1\theta]\!]_G = 1$.*

Example 3.2.1. *Consider the conjunction of constraints $\mathcal{C} = \mathcal{I}_0 \cup \{u_1\} \overset{?}{\vdash} v_1 \ \wedge\ \mathcal{I}_0 \cup \{u_1, v_2\} \overset{?}{\vdash} u_2$ and the formula $\Phi = \Phi_D \ \wedge\ \Phi_S \ \wedge\ \neg\mathsf{route}(x_L)$.*

with:

$$\begin{aligned}
u_1 &= \langle \mathsf{req}, S, D, Id, S :: \bot, \mathsf{hmac}(\langle \mathsf{req}, S, D, id\rangle, K_{SD})\rangle\\
u_2 &= \langle \mathsf{rep}, D, S, Id, x_L, \mathsf{hmac}(\langle \mathsf{rep}, D, S, id, x_L\rangle, K_{SD})\rangle\\
\Phi_D &= \mathsf{check}(D, x_a)\\
\Phi_S &= \mathsf{checkl}(S, x_L) \wedge \neg\mathsf{loop}(x_L)\\
v_1 &= \langle \mathsf{req}, S, D, x_{Id}, x_a :: x_l, \mathsf{hmac}(\langle \mathsf{req}, S, D, x_{id}\rangle, K_{SD})\rangle\\
v_2 &= \langle \mathsf{rep}, D, S, x_{Id}, x_a :: x_l, \mathsf{hmac}(\langle \mathsf{rep}, D, S, x_{Id}, x_a :: x_l\rangle, K_{SD})\rangle
\end{aligned}$$

Let $\mathcal{I}_0$ be a set of names such that $names(\mathcal{C}, \phi) \cap \mathcal{I}_0 = \emptyset$. We have that $(\mathcal{C}, \mathcal{I}_0)$ is a constraint system, and the substitution

$$\theta = \{x_{Id} \mapsto Id, x_a \mapsto X, x_l \mapsto [W; S], x_L \mapsto [X; W; S]\}$$

is a solution of the constraint system $(\mathcal{C}, \mathcal{I}_0)$ and of the formula Φ for the graph G_0 defined in Example 3.1.2.

3.2.1 Transition system

Concrete executions can be finitely represented by executing the transitions *symbolically*. A *symbolic configuration* is a quintuplet $(\mathcal{P};\mathcal{S};\mathcal{I};\mathcal{C};\Psi)$ where

- $\mathcal{P}$ is a multiset of expressions of the form $\lfloor P \rfloor_n$ where null processes are removed. $\lfloor P \rfloor_n$ represents the process P located at node $n \in \mathcal{N}_{\mathsf{loc}}$.
- $\mathcal{S}$ is a set of expressions of the form $\lfloor t \rfloor_n$ with $n \in \mathcal{N}_{\mathsf{loc}}$ and t a term (not necessarily ground).
- $\mathcal{I} = \mathcal{I}_{names} \uplus \mathcal{I}_{\mathsf{terms}}$ where $\mathcal{I}_{\mathsf{terms}}$ is a set of terms (not necessarily ground) representing the messages seen by the intruder, and $\mathcal{I}_{names}$ is a set of names that the intruder has at his disposal .
- $(\mathcal{C}, \mathcal{I}_{names})$ is a constraint system such that $T \subseteq \mathcal{I}_{\mathsf{terms}}$ for every constraint $T \overset{?}{\vdash} u \in \mathcal{C}$.
- $\Psi = \Phi_1 \wedge \Phi_2$ where $\Phi_1 \in \mathcal{L}$ and Φ_2 is a conjunction of disequality constraints.

Such a configuration is *ground* when:

$$fv(\mathcal{P}) \cup vars(\mathcal{S}) \cup vars(\mathcal{I}) \cup fv(\Psi) \subseteq rvar(\mathcal{C})$$

Compared to concrete configurations, terms exchanged by processes in symbolic configurations are not necessarily ground anymore but have to satisfy some (deduction or disequality) constraints. We define the associated symbolic transitions in Figure 3.4. They mimic concrete ones. In particular, for the communication rule, the set I of processes ready to input a message is split into three sets J, K and L. The message being transmitted is a term t, and the processes ready to input are of the form $\mathsf{in}\ u_i[\Phi_i]$. J is the set of processes that accept the message t, K is the set of processes that reject the message t because t does not unify with the expected pattern u_k, and L is the set of processes that reject the message t because the condition Φ_l is not fulfilled.

Whenever $(\mathcal{P};\mathcal{S};\mathcal{I};\mathcal{C};\Psi) \rightarrow^s_{G,\mathcal{M}} (\mathcal{P}';\mathcal{S}';\mathcal{I}';\mathcal{C}';\Psi')$ where $(\mathcal{P};\mathcal{S};\mathcal{I};\mathcal{C};\Psi)$ is a (ground) symbolic configuration then $(\mathcal{P}';\mathcal{S}';\mathcal{I}';\mathcal{C}';\Psi')$ is still a (ground) symbolic configuration.

More precisely, we show in Lemma 3.2.1 that the result of a transition from a ground symbolic configuration is also a ground symbolic configuration, in particular the set of constraints obtained is a constraint system. This lemma will be useful later, to show that our transition system is complete (Proposition 3.2.2) and sound (Proposition 3.2.3) when considering ground configurations.

Lemma 3.2.1. *Let $G = (\mathcal{N}_{\mathsf{loc}}, E)$ be a graph, $\mathcal{M} \subseteq \mathcal{N}_{\mathsf{loc}}$, and $K_s = (\mathcal{P};\mathcal{S};\mathcal{I};\mathcal{C};\Psi)$ be a ground symbolic configuration. If K'_s is a quintuplet such that $K_s \rightarrow^s_{G,\mathcal{M}} K'_s$, then K'_s is a ground symbolic configuration.*

Proof. Since K_s is a symbolic configuration, there exist $\mathcal{I}_{names}, \mathcal{I}_{\mathsf{terms}}$ such that $\mathcal{I} = \mathcal{I}_{\mathsf{terms}} \uplus \mathcal{I}_{names}$, $(\mathcal{C}, \mathcal{I}_{names})$ is a constraint system and $T \subseteq \mathcal{I}_{\mathsf{terms}}$ for every $T \overset{?}{\vdash} u \in \mathcal{C}$. We also have that $\Psi = \Phi_1 \wedge \Phi_2$ with $\Phi_1 \in \mathcal{L}$ and Φ_2 is a conjunction of disequality constraints. Moreover, since K_s is ground, we have that $var(\mathcal{I}) \cup fv(\mathcal{P}) \cup var(\mathcal{S}) \cup fv(\Psi) \subseteq rvar(\mathcal{C})$. Let us write $K'_s = (\mathcal{P}';\mathcal{S}';\mathcal{I}';\mathcal{C}';\Psi')$ and $G = (V, E)$. To prove the result, we do a case analysis on

$$\textsc{Comm}_s \quad (\lfloor \mathsf{out}(t).P\rfloor_n \cup \{\lfloor \mathsf{in}\ u_i[\Phi_i].P'_i\rfloor_{n_i} \mid i \in I\} \cup \mathcal{P}; \mathcal{S}; \mathcal{I}; \mathcal{C}; \Psi) \to^s_{G,\mathcal{M}} (\{\lfloor P'_j\sigma\rfloor_{n_j} \mid j \in J\} \cup \mathcal{P}\sigma; \mathcal{S}\sigma; \mathcal{I}'; \mathcal{C}'; \Psi')$$

where:

- $\lfloor P'\rfloor_{n'} \in \mathcal{P}$ implies that $(n, n') \notin E$ or P' is not of the form $\mathsf{in}\ u'[\Phi'].Q'$,
- $I = J \uplus K \uplus L$ and $(n_i, n) \in E$ for every $i \in I$,
- for every $l \in L, \alpha_l$ is a renaming of $vars(u_l) \smallsetminus rvar(\mathcal{C})$ by fresh variables
- $\sigma = mgu(\{u_j = t \mid j \in J\} \cup \{u_l\alpha_l = t \mid l \in L\})$
- $\Psi_J = \{\Phi_j \mid j \in J\}$, $\Psi_K = \{\forall Y_k\,.\,t \neq u_k \mid k \in K\}$ and $\Psi_L = \{\neg\Phi_l\alpha_l \mid l \in L\}$ where $Y_k = (vars(u_k) \smallsetminus rvar(\mathcal{C}))$
- $\mathcal{I}' = (\mathcal{I} \cup \{t\})\sigma$ when $(n, n_I) \in E$ for some $n_I \in \mathcal{M}$, and $\mathcal{I}' = \mathcal{I}\sigma$ otherwise.
- $\mathcal{C}' = \mathcal{C}\sigma$ and $\Psi' = (\Psi \cup \Psi_J \cup \Psi_K \cup \Psi_L)\sigma$

$$\textsc{In}_s \quad (\lfloor \mathsf{in}\ u[\Phi].P\rfloor_n \cup \mathcal{P}; \mathcal{S}; \mathcal{I}; \mathcal{C}; \Psi) \to^s_{G,\mathcal{M}} (\lfloor P\rfloor_n \cup \mathcal{P}; \mathcal{S}; \mathcal{I}; \mathcal{C} \wedge \mathcal{I} \overset{?}{\vdash} u; \Psi \wedge \Phi)$$
if $(n_I, n) \in E$ for some $n_I \in \mathcal{M}$

$$\textsc{Store}_s \quad (\lfloor \mathsf{store}(t).P\rfloor_n \cup \mathcal{P}; \mathcal{S}; \mathcal{I}; \mathcal{C}; \Psi) \to^s_{G,\mathcal{M}} (\lfloor P\rfloor_n \cup \mathcal{P}; \lfloor t\rfloor_n \cup \mathcal{S}; \mathcal{I}; \mathcal{C}; \Psi)$$

$$\textsc{Read-Then}_s \quad (\lfloor \mathsf{read}\ u\ \mathsf{then}\ P\ \mathsf{else}\ Q\rfloor_n \cup \mathcal{P}; \mathcal{S}; \mathcal{I}; \mathcal{C}; \Psi) \to^s_{G,\mathcal{M}} (\lfloor P\sigma\rfloor_n \cup \mathcal{P}\sigma; \mathcal{S}\sigma; \mathcal{I}\sigma; \mathcal{C}\sigma; \Psi\sigma)$$
where $\lfloor t\rfloor_n \in \mathcal{S}$ and $\sigma = mgu(t, u)$

$$\textsc{Read-Else}_s \quad (\lfloor \mathsf{read}\ u\ \mathsf{then}\ P\ \mathsf{else}\ Q\rfloor_n \cup \mathcal{P}; \mathcal{S}; \mathcal{I}; \mathcal{C}; \Psi) \to^s_{G,\mathcal{M}} (\lfloor Q\rfloor_n \cup \mathcal{P}; \mathcal{S}; \mathcal{I}; \mathcal{C}; \Psi \wedge \{\forall X\,.\,t \neq u \mid \lfloor t\rfloor_n \in \mathcal{S}\})$$
where $X = vars(u) \smallsetminus rvar(\mathcal{C})$

$$\textsc{If-Then}_s \quad (\lfloor \mathsf{if}\ \Phi\ \mathsf{then}\ P\ \mathsf{else}\ Q\rfloor_n \cup \mathcal{P}; \mathcal{S}; \mathcal{I}; \mathcal{C}; \Psi) \to^s_{G,\mathcal{M}} (\lfloor P\rfloor_n \cup \mathcal{P}; \mathcal{S}; \mathcal{I}; \mathcal{C}; \Psi \wedge \Phi)$$

$$\textsc{If-Else}_s \quad (\lfloor \mathsf{if}\ \Phi\ \mathsf{then}\ P\ \mathsf{else}\ Q\rfloor_n \cup \mathcal{P}; \mathcal{S}; \mathcal{I}; \mathcal{C}; \Psi) \to^s_{G,\mathcal{M}} (\lfloor Q\rfloor_n \cup \mathcal{P}; \mathcal{S}; \mathcal{I}; \mathcal{C}; \Psi \wedge \neg\Phi)$$

$$\textsc{Par}_s \quad (\lfloor P_1 \mid P_2\rfloor_n \cup \mathcal{P}; \mathcal{S}; \mathcal{I}; \mathcal{C}; \Psi) \to^s_{G,\mathcal{M}} (\lfloor P_1\rfloor_n \cup \lfloor P_2\rfloor_n \cup \mathcal{P}; \mathcal{S}; \mathcal{I}; \mathcal{C}; \Psi)$$

$$\textsc{Repl}_s \quad (\lfloor !P\rfloor_n \cup \mathcal{P}; \mathcal{S}; \mathcal{I}; \mathcal{C}; \Psi) \to^s_{G,\mathcal{M}} (\lfloor P\alpha\rfloor_n \cup \lfloor !P\rfloor_n \cup \mathcal{P}; \mathcal{S}; \mathcal{I}; \mathcal{C}; \Psi)$$
where α is a renaming of the bound variables of P that are not in $rvar(\mathcal{C})$.

$$\textsc{New}_s \quad (\lfloor \mathsf{new}\ m.P\rfloor_n \cup \mathcal{P}; \mathcal{S}; \mathcal{I}; \mathcal{C}; \Psi) \to^s_{G,\mathcal{M}} (\lfloor P\{m \mapsto m'\}\rfloor_n \cup \mathcal{P}; \mathcal{S}; \mathcal{I}; \mathcal{C}; \Psi)$$
where m' is a fresh name

Figure 3.4: Symbolic transition system.

the transition rule involved in $K_s \rightarrow^s_{G,\mathcal{M}} K'_s$. Note that the result is straightforward for the rules STORE$_s$, PAR$_s$, REPL$_s$, and NEW$_s$. Indeed, in these cases, we have that $\mathcal{C}' = \mathcal{C}$, $\Psi' = \Psi$, $\mathcal{I}' = \mathcal{I}$, and $fv(\mathcal{P}) \cup var(\mathcal{S}) = fv(\mathcal{P}') \cup var(\mathcal{S}')$. Now, we consider the remaining rules in turn.

- Rule READ-THEN$_s$. We have that:

  $$(\lfloor \mathsf{read}\ u\ \mathsf{then}\ P\ \mathsf{else}\ Q \rfloor_n \cup \mathcal{Q}; \mathcal{S}; \mathcal{I}; \mathcal{C}; \Psi) \rightarrow^s_{G,\mathcal{M}} (\lfloor P\sigma \rfloor_n \cup \mathcal{Q}\sigma; \mathcal{S}\sigma; \mathcal{I}\sigma; \mathcal{C}\sigma; \Psi\sigma)$$

  where $\lfloor t \rfloor_n \in \mathcal{S}$ and $\sigma = mgu(u, t)$.

  First, we have that $(\mathcal{C}', \mathcal{I}_{names})$ is a constraint system. Indeed, monotonicity is straightforwardly satisfied by $\mathcal{C}'$. Moreover, application of a substitution preserves origination. Since $\mathcal{I}' = \mathcal{I}\sigma = \mathcal{I}_{\mathsf{terms}}\sigma \uplus \mathcal{I}_{names}$ and $\mathcal{C}' = \mathcal{C}\sigma$, we also have that $T' \subseteq \mathcal{I}_{\mathsf{terms}}\sigma$ for every $T' \overset{?}{\vdash} u' \in \mathcal{C}'$. Lastly, since K_s is ground, we have that:

  $$fv(P) \cup fv(\mathcal{Q}) \cup vars(S) \cup vars(\mathcal{I}) \cup fv(\Psi) \subseteq rvar(\mathcal{C})$$

  Thus, we deduce that

  $$fv(P\sigma) \cup fv(\mathcal{Q}\sigma) \cup vars(S\sigma) \cup vars(\mathcal{I}\sigma) \cup fv(\Psi\sigma) \subseteq rvar(\mathcal{C}\sigma)$$

  We conclude that the resulting symbolic configuration K'_s is ground.

- Rule READ-ELSE$_s$. We have that:

  $$(\lfloor \mathsf{read}\ u\ \mathsf{then}\ P\ \mathsf{else}\ Q \rfloor_n \cup \mathcal{Q}; \mathcal{S}; \mathcal{I}; \mathcal{C}; \Psi) \rightarrow^s_{G,\mathcal{M}} (\lfloor Q \rfloor_n \cup \mathcal{Q}; \mathcal{S}; \mathcal{I}; \mathcal{C}; \Psi \wedge \mathsf{Eq})$$

  where $\mathsf{Eq} = \{\forall var(u) \smallsetminus rvar(\mathcal{C}) \,.\, t \neq u \mid \lfloor t \rfloor_n \in \mathcal{S}\}$.

  First, we know that $\mathcal{I} = \mathcal{I}_{\mathsf{terms}} \uplus \mathcal{I}_{names}$, $(\mathcal{C}', \mathcal{I}_{names})$ is a constraint system, and we have that $T \subseteq \mathcal{I}_{\mathsf{terms}}$ for every $T \overset{?}{\vdash} u \in \mathcal{C}$. Furthermore, since K_s is ground, we already have that $fv(Q) \cup fv(\mathcal{Q}) \cup var(\mathcal{S}) \cup var(\mathcal{I}) \subseteq rvar(\mathcal{C})$. Since $fv(\mathsf{Eq}) \subseteq rvar(\mathcal{C})$, we obtain that $fv(\Psi') \subseteq rvar(\mathcal{C})$. In conclusion, the resulting configuration K'_s is ground.

- Rule IF-THEN$_s$. We have that:

  $$(\lfloor \mathsf{if}\ \Phi\ \mathsf{then}\ P\ \mathsf{else}\ Q \rfloor_n \cup \mathcal{Q}; \mathcal{S}; \mathcal{I}; \mathcal{C}; \Psi) \rightarrow^s_{G,\mathcal{M}} (\lfloor P \rfloor_n \cup \mathcal{Q}; \mathcal{S}; \mathcal{I}; \mathcal{C}; \Psi \wedge \Phi)$$

  We still have that $\mathcal{I} = \mathcal{I}_{\mathsf{terms}} \uplus \mathcal{I}_{names}$, $(\mathcal{C}', \mathcal{I}_{names})$ is a constraint system, and $T \subseteq \mathcal{I}_{\mathsf{terms}}$ for every $T \overset{?}{\vdash} u \in \mathcal{C}$. Since K_s is ground, we have that

  $$fv(P) \cup fv(\Phi) \cup fv(\mathcal{Q}) \cup var(\mathcal{S}) \cup var(\mathcal{I}) \cup fv(\Psi) \subseteq rvar(\mathcal{C})$$

  Thus, the configuration K'_s is ground.

- Rule IF-ELSE$_s$. Similar to the previous case.

- Rule IN$_s$. We have that:

  $$(\lfloor \mathsf{in}\ u[\Phi].P \rfloor_n \cup \mathcal{Q}; \mathcal{S}; \mathcal{I}; \mathcal{C}; \Psi) \rightarrow^s_{G,\mathcal{M}} (\lfloor P \rfloor_n \cup \mathcal{Q}; \mathcal{S}; \mathcal{I}; \mathcal{C} \wedge \mathcal{I} \overset{?}{\vdash} u; \Psi \wedge \Phi)$$

  where $(n_I, n) \in E$ for some $n_I \in \mathcal{M}$.

  We have that $\mathcal{I} = \mathcal{I}_{\mathsf{terms}} \uplus \mathcal{I}_{names}$, $(\mathcal{C}', \mathcal{I}_{names})$ is a constraint system, and $T \subseteq \mathcal{I}_{\mathsf{terms}}$ for every $T \overset{?}{\vdash} u \in \mathcal{C}$. Consequently, we deduce that $(\mathcal{C}', \mathcal{I}_{names})$ satisfies the monotonicity property. Since $var(\mathcal{I}) \subseteq rvar(\mathcal{C})$ (because K_s is ground), $(\mathcal{C}', \mathcal{I}_{names})$ furthermore satisfies the origination property. Clearly, we have that $T \subseteq \mathcal{I}_{\mathsf{terms}}$ for any $T \overset{?}{\vdash} u \in \mathcal{C}'$. Lastly, since K_s is ground and $fv(\mathsf{in}\ u[\Phi]P) = (fv(P) \cup fv(\Phi)) \smallsetminus var(u)$, we have that:

$$fv(P) \cup fv(\Phi) \cup fv(\mathcal{Q}) \cup var(\mathcal{S}) \cup var(\mathcal{I}) \cup fv(\Psi) \subseteq rvar(\mathcal{C}) \cup var(u).$$

Since $rvar(\mathcal{C}') = rvar(\mathcal{C}) \cup var(u)$, we easily deduce that the symbolic configuration K'_s is ground.

- Rule $\textsc{Comm}_s$. We have that:

$$(\lfloor \mathsf{out}(t).P \rfloor_n \cup \mathcal{P}_I \cup \mathcal{Q}; \mathcal{S}; \mathcal{I}; \mathcal{C}; \Psi) \ \rightarrow^s_{G,\mathcal{M}} \\ (\lfloor P\sigma \rfloor_n \cup \mathcal{P}_J\sigma \cup \mathcal{P}_{K,L}\sigma \cup \mathcal{Q}\sigma; \mathcal{S}\sigma; \mathcal{I}'\sigma; \mathcal{C}\sigma; \Psi')$$

where:

  - $\mathcal{P}_I = \{\lfloor \mathsf{in}\ u_i[\Phi_i].P_i \rfloor_{n_i} \mid i \in I\}$,
  - $\mathcal{P}_J = \{\lfloor P_j \rfloor_{n_j} \mid j \in J\}$
  - $\mathcal{P}_{K,L} = \{\lfloor \mathsf{in}\ u_k[\Phi_k].P_k \rfloor_{n_k} \mid k \in K \cup L\}$
  - $\Psi_J = \{\Phi_j \mid j \in J\}$ and $\Psi_L = \{\neg \Phi_l \alpha_l \mid l \in L\}$
  - $\Psi_K = \{\forall var(u_k) \smallsetminus rvar(\mathcal{C}) \,.\, t \neq u_k \mid k \in K\}$,
  - $\sigma = mgu(\{u_j = t \mid j \in J\} \cup \{u_l \alpha_l = t \mid l \in L\})$
  - $\Psi' = (\Psi \wedge \Psi_J \wedge \Psi_K \wedge \Psi_L)\sigma$

$\lfloor P' \rfloor_{n'} \in \mathcal{Q}$ implies that $(n, n') \notin E$ or P' is not of the form $\mathsf{in}\ u'[\Phi'].Q'$, $I = J \uplus K \uplus L$, $(n_i, n) \in E$ for any $i \in I$, α_l is a renaming of $var(u_l) \smallsetminus rvar(\mathcal{C})$ by fresh variables, and if $(n, n_I) \in E$ for some $n_I \in \mathcal{M}$ then $\mathcal{I}' = \mathcal{I} \cup \{t\})$ else $\mathcal{I}' = \mathcal{I}$.

Clearly, $T\sigma \subseteq \mathcal{I}_{\mathsf{terms}}\sigma$ for any $T \overset{?}{\vdash} u \in \mathcal{C}$. Moreover, $(\mathcal{C}', \mathcal{I}_{names})$ straightforwardly satisfies the monotonicity property. As substitution preserves origination, $(\mathcal{C}', \mathcal{I}_{names})$ satisfies the origination property. Consequently, K'_s is a symbolic configuration. Lastly, we have to show that K'_s is ground. Since K_s is ground, we have that:

$$fv(t, P, \mathcal{P}_I, \mathcal{Q}, \Psi) \cup var(\mathcal{S}, \mathcal{I}) \subseteq rvar(\mathcal{C})$$

We immediately deduce that

$$fv(P\sigma, \mathcal{P}_J\sigma, \mathcal{P}_{K,L}\sigma, \mathcal{Q}\sigma, \Psi\sigma) \cup var(\mathcal{S}\sigma, \mathcal{I}\sigma, t\sigma) \subseteq rvar(\mathcal{C}\sigma)$$

It remains to show that $fv(\Psi_J \wedge \Psi_K \wedge \Psi_L)\sigma \subseteq rvar(\mathcal{C}\sigma)$. Now, $fv(\Psi_J) \subseteq rvar(\mathcal{C})$ as K_s is ground. Furthermore, $fv(\Psi_K) \subseteq rvar(\mathcal{C})$ by definition of Ψ_K. So we only have to prove that $fv(\Psi_L\sigma) \subseteq rvar(\mathcal{C}\sigma)$. Let $l \in L$, and $X_l = var(u_l) \smallsetminus rvar(\mathcal{C})$. Necessarily, $var(\Phi_l) \subseteq var(u_l) \cup rvar(\mathcal{C}) = X_l \uplus rvar(\mathcal{C})$. Consequently, $var(\Phi_l\alpha_l) \subseteq X_l\alpha_l \uplus rvar(\mathcal{C})$, as α_l is a renaming of X_l by fresh variables. Moreover, $(u_l\alpha_l)\sigma = t\sigma$, so $var((u_l\alpha_l)\sigma) = var(t\sigma) \subseteq rvar(\mathcal{C}\sigma)$. Consequently, $var((\Phi_l\alpha_l)\sigma) \subseteq rvar(\mathcal{C}\sigma)$. We easily deduce that $fv(\Psi_L\sigma) \subseteq rvar(\mathcal{C}\sigma)$.

We conclude that K'_s is a ground symbolic configuration.

□

Example 3.2.2. *Executing the same transitions as in Example 3.1.4 symbolically, we reach the following configuration:*

$$K_s = (\lfloor \mathsf{out}(\mathsf{error}).0 \rfloor_S; \emptyset; \mathcal{I}_0 \cup \{u_1, v_2\}; \mathcal{C}; \Phi)$$

where $\mathcal{C}, \Phi, u_1, v_2$ are defined as in Example 3.2.1.

3.2.2 Soundness and completeness

We show that our symbolic transition system reflects exactly the concrete transition system, i.e. each concrete execution of a process is captured by one of the symbolic executions. More precisely, a concrete configuration is represented by a symbolic configuration if it is one of its instances, called *concretization*.

Definition 3.2.3 (θ-concretization). *Let $K_s = (\mathcal{P}_s; \mathcal{S}_s; \mathcal{I}_s; \mathcal{C}; \Psi)$ be a symbolic configuration such that $\mathcal{I} = \mathcal{I}_{names} \uplus \mathcal{I}_{\mathsf{terms}}$, $(\mathcal{C}, \mathcal{I}_{names})$ is a constraint system. A concretization of K_s is a concrete configuration $K_c = (\mathcal{P}; \mathcal{S}; \mathcal{I})$ such that there exists θ a solution of $(\mathcal{C}, \mathcal{I}_{names})$ and Ψ and, furthermore, $\mathcal{P}_s\theta = \mathcal{P}$, $\mathcal{S}_s\theta = \mathcal{S}$, $\mathcal{I}_s\theta = \mathcal{I}$. We say that K_c is a θ-concretization of K_s.*

Note that the θ-concretization of a ground symbolic configuration is a ground concrete configuration. Now, we show that each concrete transition can be matched by a symbolic one. The proof is performed by studying each rule of the concrete transition system, showing that the corresponding symbolic rule covers all possible cases. In particular, disequality constraints allow to faithfully model cases where nodes reject a message because the message does not match the expected pattern.

Proposition 3.2.2 (completeness). *Let $G = (\mathcal{N}_{\mathsf{loc}}, E)$ be a graph and $\mathcal{M} \subseteq \mathcal{N}_{\mathsf{loc}}$. Let $K_s = (\mathcal{P}_s; \mathcal{S}_s; \mathcal{I}_s; \mathcal{C}; \Psi)$ be a ground symbolic configuration with $\mathcal{I} = \mathcal{I}_{names} \uplus \mathcal{I}_{\mathsf{terms}}$ and θ be a solution of the constraint system $(\mathcal{C}, \mathcal{I}_{names})$ and Ψ. Let K_c be the θ-concretization of K_s. Let K'_c be a concrete configuration such that $K_c \rightarrow_{G,\mathcal{M}} K'_c$. Then there exists a ground symbolic configuration K'_s and a substitution θ' such that:*

- *K'_c is the θ'-concretization of K'_s, and*
- *$K_s \rightarrow^s_{G,\mathcal{M}} K'_s$.*

Proof. Let $K_c = (\mathcal{P}; \mathcal{S}; \mathcal{I})$. We distinguish cases depending on which transition is applied to K_c. We write $K'_c = (\mathcal{P}'; \mathcal{S}'; \mathcal{I}')$. We show that there exists a symbolic configuration K'_s such that K'_c is the θ'-concretization of K'_s and $K_s \rightarrow^s_{G,\mathcal{M}} K'_s$. Thanks to Lemma 3.2.1, we easily deduce that K'_s is ground.

- Rule Par. We have that:
$$(\lfloor P_1 | P_2 \rfloor_n \cup \mathcal{Q}; \mathcal{S}; \mathcal{I}) \rightarrow_{G,\mathcal{M}} (\lfloor P_1 \rfloor_n \cup \lfloor P_2 \rfloor_n \cup \mathcal{Q}; \mathcal{S}; \mathcal{I})$$
By hypothesis, K_s is a symbolic configuration whose θ-concretization is K_c. Consequently, we have that $K_s = (\lfloor P^s_1 | P^s_2 \rfloor_n \cup \mathcal{Q}_s; \mathcal{S}_s; \mathcal{I}_s; \mathcal{C}; \Psi)$ with $\mathcal{Q}_s\theta = \mathcal{Q}$, $\mathcal{S}_s\theta = \mathcal{S}$, $\mathcal{I}_s\theta = \mathcal{I}$, $P^s_1\theta = P_1$, and $P^s_2\theta = P_2$. Let $K'_s = (\lfloor P^s_1 \rfloor_n \cup \lfloor P^s_2 \rfloor_n \cup \mathcal{Q}_s; \mathcal{S}_s; \mathcal{I}_s; \mathcal{C}; \Psi)$. We have that $K_s \rightarrow^s_{G,\mathcal{M}} K'_s$ (with the Par$_s$ rule), θ is a solution of $(\mathcal{C}, \mathcal{I}_{names})$ and Ψ and K'_c is the θ-concretization of K'_s.

- Rule Repl. We have that:
$$(\lfloor !P\rfloor_n \cup \mathcal{Q}; \mathcal{S}; \mathcal{I}) \rightarrow_{G,\mathcal{M}} (\lfloor P\alpha\rfloor_n \cup \lfloor !P\rfloor_n \cup \mathcal{Q}; \mathcal{S}; \mathcal{I})$$
where α is a fresh renaming of the bound variables in P. Since K_s is a symbolic configuration whose θ-concretization is K_c, we have that $K_s = (\lfloor !P_s\rfloor_n \cup \mathcal{Q}_s; \mathcal{S}_s; \mathcal{I}_s; \mathcal{C}; \Psi)$ with $\mathcal{Q}_s\theta = \mathcal{Q}$, $\mathcal{S}_s\theta = \mathcal{S}$, $\mathcal{I}_s\theta = \mathcal{I}$ and $P_s\theta = P$. Note that α is also a renaming of the variables in $bv(P_s) \smallsetminus rvar(\mathcal{C})$. Let $K'_s = (\lfloor P_s\alpha\rfloor_n \cup \lfloor !P_s\rfloor_n \cup \mathcal{Q}_s; \mathcal{S}_s; \mathcal{I}_s; \mathcal{C}; \Phi)$. We have that $K_s \rightarrow^s_{G,\mathcal{M}} K'_s$ (with the Repl$_s$ rule) and θ is a solution of $(\mathcal{C}, \mathcal{I}_{names})$. It remains to show that K'_c is the θ-concretization of K'_s. Since the variables introduced by α are fresh, we have that $img(\alpha) \cap dom(\theta) = \emptyset$, and since $P_s\theta = P$, we have that $dom(\alpha) \cap dom(\theta) = \emptyset$. Hence we have that $(P_s\alpha)\theta = (P_s\theta)\alpha = P\alpha$. This allows us to conclude.

- Rule New. We have that:
$$(\lfloor \mathsf{new}\ m.P\rfloor_n \cup \mathcal{Q}; \mathcal{S}; \mathcal{I}) \rightarrow_{G,\mathcal{M}} (\lfloor P\{m \mapsto m'\}\rfloor_n \cup \mathcal{Q}; \mathcal{S}; \mathcal{I})$$
where m' is a fresh name.

  We know that K_s is a symbolic configuration whose θ-concretization is K_c. Thus, we have that $K_s = (\lfloor \mathsf{new}\ m.P_s\rfloor_n \cup \mathcal{Q}_s; \mathcal{S}_s; \mathcal{I}_s; \mathcal{C}; \Psi)$ with $\mathcal{Q}_s\theta = \mathcal{Q}$, $\mathcal{S}_s\theta = \mathcal{S}$, $\mathcal{I}_s\theta = \mathcal{I}$, and $P_s\theta = P$. Let $K'_s = (\lfloor P_s\{m \mapsto m'\}\rfloor_n \cup \mathcal{Q}_s; \mathcal{S}_s; \mathcal{I}_s; \mathcal{C}; \Psi)$. We have that $K_s \rightarrow^s_{G,\mathcal{M}} K'_s$ (with the New$_s$ rule), θ is a solution of $(\mathcal{C}, \mathcal{I}_{names})$ and K'_c is the θ-concretization of K'_s.

- Rule Store. We have that:
$$(\lfloor \mathsf{store}(t).P\rfloor_n \cup \mathcal{Q}; \mathcal{S}; \mathcal{I}) \rightarrow_{G,\mathcal{M}} (\lfloor P\rfloor_n \cup \mathcal{Q}; \lfloor t\rfloor_n \cup \mathcal{S}; \mathcal{I})$$
We know that K_s is a symbolic configuration whose θ-concretization is K_c. Thus, we have that $K_s = (\lfloor \mathsf{store}(t_s).P_s\rfloor_n \cup \mathcal{Q}_s; \mathcal{S}_s; \mathcal{I}_s; \mathcal{C}; \Psi)$ with $\mathcal{Q}_s\theta = \mathcal{Q}$, $\mathcal{S}_s\theta = \mathcal{S}$, $\mathcal{I}_s\theta = \mathcal{I}$, $P_s\theta = P$ and $t_s\theta = t$. Let $K'_s = (\lfloor P_s\rfloor_n \cup \mathcal{Q}_s; \lfloor t_s\rfloor_n \cup \mathcal{S}_s; \mathcal{I}_s; \mathcal{C}; \Psi)$. We have that $K_s \rightarrow^s_{G,\mathcal{M}} K'_s$ (with the Store$_s$ rule), θ is a solution of $(\mathcal{C}, \mathcal{I}_{names})$ and K'_c is the θ-concretization of K'_s.

- Rule Read-Then . We have that:
$$(\lfloor \mathsf{read}\ u\ \mathsf{then}\ P\ \mathsf{else}\ Q\rfloor_n \cup \mathcal{Q}; \lfloor t\rfloor_n \cup \mathcal{S}; \mathcal{I}) \rightarrow_{G,\mathcal{M}} (\lfloor P\sigma\rfloor_n \cup \mathcal{Q}; \lfloor t\rfloor_n \cup \mathcal{S}; \mathcal{I})$$
where $\sigma = mgu(t, u)$.

  We know that K_s is a symbolic configuration whose θ-concretization is K_c. Thus, we have that $K_s = (\lfloor \mathsf{read}\ u_s\ \mathsf{then}\ P_s\ \mathsf{else}\ Q_s\rfloor_n \cup \mathcal{Q}_s; \lfloor t_s\rfloor_n \cup \mathcal{S}_s; \mathcal{I}_s; \mathcal{C}; \Psi)$ with $\mathcal{Q}_s\theta = \mathcal{Q}$, $u_s\theta = u$, $t_s\theta = t$, $P_s\theta = P$, $Q_s\theta = Q$, $\mathcal{S}_s\theta = \mathcal{S}$ and $\mathcal{I}_s\theta = \mathcal{I}$. By hypothesis, we have that $(u_s\theta)\sigma = u\sigma = t\sigma = (t_s\theta)\sigma$, so $\sigma' = mgu(u_s, t_s)$ exists and there exists θ' a substitution such that $\sigma \circ \theta = \theta' \circ \sigma'$. Let us define $K'_s = (\lfloor P_s\sigma'\rfloor_n \cup \mathcal{Q}_s\sigma'; \lfloor t_s\sigma'\rfloor_n \cup \mathcal{S}_s\sigma'; \mathcal{I}_s\sigma'; \mathcal{C}\sigma'; \Psi\sigma')$. We have that $K_s \rightarrow^s_{G,\mathcal{M}} K'_s$ (with the Read-Then$_s$ rule).

  It remains to prove that θ' is a solution for $(\mathcal{C}\sigma', \mathcal{I}_{names})$ and $\Psi\sigma'$. As θ is a solution of $(\mathcal{C}, \mathcal{I}_{names})$, for every $T \overset{?}{\vdash} u \in \mathcal{C}$, $T\theta \cup \mathcal{I}_{names} \vdash u\theta$, and so $(T\sigma')\theta' \cup \mathcal{I}_{names} \vdash (u\sigma')\theta'$. With a similar reasoning, we can show that θ' is also a solution for $\Psi\sigma'$. Furthermore, we have that $\mathcal{Q}_s\sigma'\theta' = \mathcal{Q}_s\theta\sigma = \mathcal{Q}\sigma$, $P_s\sigma'\theta' = P_s\theta\sigma = P\sigma$, $\mathcal{S}_s\sigma'\theta' = \mathcal{S}_s\theta\sigma = \mathcal{S}\sigma$ and $\mathcal{I}_s\sigma'\theta' = \mathcal{I}_s\theta\sigma = \mathcal{I}\sigma$. As $\sigma = mgu(u, t)$ where t is ground and the variables of u are bound in ($\mathsf{read}\ u\ \mathsf{then}\ P\ \mathsf{else}\ Q$), we deduce that $dom(\sigma) \cap vars(\mathcal{Q}, \mathcal{S}, \mathcal{I}) = \emptyset$, and so $\mathcal{Q}\sigma = \mathcal{Q}$, $\mathcal{S}\sigma = \mathcal{S}$ and $\mathcal{I}\sigma = \mathcal{I}$. Hence, we have that K'_c is the θ'-concretization of K'_s.

- Rule Read-Else. We have that:
$$(\lfloor \mathsf{read}\ u\ \mathsf{then}\ P\ \mathsf{else}\ Q\rfloor_n \cup \mathcal{Q}; \mathcal{S}; \mathcal{I}) \rightarrow_{G,\mathcal{M}} (\lfloor Q\rfloor_n \cup \mathcal{Q}; \mathcal{S}; \mathcal{I})$$
and for all $\lfloor t\rfloor_n \in \mathcal{S}$ we have that $mgu(t,u) = \bot$.

  We know that K_s is a symbolic configuration whose θ-concretization is K_c. Thus, we have that $K_s = (\lfloor \mathsf{read}\ u_s\ \mathsf{then}\ P_s\ \mathsf{else}\ Q_s\rfloor_n \cup \mathcal{Q}_s; \mathcal{S}_s; \mathcal{I}_s; \mathcal{C}; \Psi)$ with $u_s\theta = u$, $P_s\theta = P$, $Q_s\theta = Q$, $\mathcal{Q}_s\theta = \mathcal{Q}$, $\mathcal{S}_s\theta = \mathcal{S}$, and $\mathcal{I}_s\theta = \mathcal{I}$.

  Let $K'_s = (\lfloor Q_s\rfloor_n \cup \mathcal{Q}_s; \mathcal{S}_s; \mathcal{I}_s; \mathcal{C}; \Psi')$ where $\Psi' = \Psi \wedge \{\forall Y. t_s \neq u_s \mid \lfloor t_s\rfloor_n \in \mathcal{S}\}$ and $Y = var(u_s) \smallsetminus rvar(\mathcal{C})$. We have that $K_s \rightarrow^s_{G,\mathcal{M}} K'_s$ (with the Read-Else$_s$ rule). Now, let us show that θ is a solution of Ψ'. Let $\forall Y. t_s \neq u_s$ be a disequation in $\Psi' \smallsetminus \Psi$. We have that $u_s\theta = u$, $t_s\theta = t$ for some term t such that $\lfloor t\rfloor_n \in \mathcal{S}$, and $mgu(t,u) = \bot$. Thus, θ is also a solution of this constraint, and more generally θ is a solution of Ψ'. Now, it is easy to see that K'_c is the θ-concretization of K'_s.

- Rule If-Then. We have that:
$$(\lfloor \mathsf{if}\ \Phi\ \mathsf{then}\ P\ \mathsf{else}\ Q\rfloor_n \cup \mathcal{Q}; \mathcal{S}; \mathcal{I}) \rightarrow_{G,\mathcal{M}} (\lfloor P\rfloor_n \cup \mathcal{Q}; \mathcal{S}; \mathcal{I}) \qquad \text{and } [\![\Phi]\!]_G = 1.$$

  We know that K_s is a symbolic configuration whose θ-concretization is K_c. Thus, we have that $K_s = (\lfloor \mathsf{if}\ \Phi_s\ \mathsf{then}\ P_s\ \mathsf{else}\ Q_s\rfloor_n \cup \mathcal{Q}_s; \mathcal{S}_s; \mathcal{I}_s; \mathcal{C}; \Psi)$ with $\Phi_s\theta = \Phi$, $P_s\theta = P$, $Q_s\theta = Q$, $\mathcal{Q}_s\theta = \mathcal{Q}$, $\mathcal{S}_s\theta = \mathcal{S}$, and $\mathcal{I}_s\theta = \mathcal{I}$.

  Let $K'_s = (\lfloor P_s\rfloor_n \cup \mathcal{Q}_s; \mathcal{S}_s; \mathcal{I}_s; \mathcal{C}; \Psi \wedge \Phi_s)$. We have that $K_s \rightarrow^s_{G,\mathcal{M}} K'_s$ (with the If-Then$_s$ rule). By hypothesis, we have that θ is a solution of Ψ, and as $[\![\Phi_s\theta]\!]_G = [\![\Phi]\!]_G$ is true, we easily deduce that θ is a solution of $\Psi' = \Psi \wedge \Phi_s$. Lastly, it is easy to see that K'_c is the θ-concretization of K'_s.

- Rule If-Else. Similar to the previous case.

- Rule In. We have that:
$$(\lfloor \mathsf{in}\ u[\Phi].P\rfloor_n \cup \mathcal{Q}; \mathcal{S}; \mathcal{I}) \rightarrow_{G,\mathcal{M}} (\lfloor P\sigma\rfloor_n \cup \mathcal{Q}; \mathcal{S}; \mathcal{I})$$
with $(n_I, n) \in E$ for some $n_I \in \mathcal{M}$, $\sigma = mgu(t,u)$, $\mathcal{I} \vdash t$ and $[\![\Phi\sigma]\!]_G = 1$.

  We know that K_s is a symbolic configuration whose θ-concretization is K_c. Thus, we have that $K_s = (\lfloor \mathsf{in}\ u_s[\Phi_s].P_s\rfloor_n \cup \mathcal{Q}_s; \mathcal{S}_s; \mathcal{I}_s; \mathcal{C}; \Psi)$ with $u_s\theta = u$, $\Phi_s\theta = \Phi$, $P_s\theta = P$, $\mathcal{Q}_s\theta = \mathcal{Q}$, $\mathcal{S}_s\theta = \mathcal{S}$, and $\mathcal{I}_s\theta = \mathcal{I}$.

  Let $K'_s = (\lfloor P_s\rfloor_n \cup \mathcal{Q}_s; \mathcal{S}_s; \mathcal{I}_s; \mathcal{C}'; \Psi')$ where $\mathcal{C}' = \mathcal{C} \wedge \mathcal{I}_s \overset{?}{\vdash} u_s$ and $\Psi' = \Psi \wedge \Phi_s$. We have that $K_s \rightarrow^s_{G,\mathcal{M}} K'_s$ (with the In$_s$ rule). Let $\theta' = \theta \circ \sigma$. By hypothesis, we have that θ is a solution of $(\mathcal{C}, \mathcal{I}_{names})$ and Ψ. To show that θ' is a solution of $(\mathcal{C}', \mathcal{I}_{names})$ and Ψ', it remains to establish that:

  - $(\mathcal{I}_{\mathsf{terms}}\theta)\sigma \cup \mathcal{I}_{names} \vdash (u_s\theta)\sigma$: We have that $(\mathcal{I}_{\mathsf{terms}}\theta)\sigma \cup \mathcal{I}_{names} = \mathcal{I}$ since $var(\mathcal{I}) = \emptyset$, and $(u_s\theta)\sigma = u\sigma = t$. Since by hypothesis, we have that $\mathcal{I} \vdash t$, we easily conclude.
  - $[\![(\Phi_s\theta)\sigma]\!]_G = 1$. Actually, we have that $(\Phi_s\theta)\sigma = \Phi\sigma$. Since, by hypothesis, we have that $[\![\Phi\sigma]\!]_G = 1$, we easily conclude.

  Hence, we have that θ' is a solution of $\mathcal{C}'$. It is easy to see that K'_c is the θ'-concretization of K'_s.

- Rule COMM. We have that

$$K_c = (\lfloor \mathsf{out}(t).P \rfloor_n \cup \{\lfloor \mathsf{in}\ u_j[\Phi_j].P_j \rfloor_{n_j} \mid j \in J\} \cup \mathcal{Q}; \mathcal{S}; \mathcal{I}) \\ \rightarrow_{G,\mathcal{M}} (\lfloor P \rfloor_n \cup \lfloor P_j \sigma_j \rfloor_{n_j}\} \cup \mathcal{Q}; \mathcal{S}; \mathcal{I}') = K'_c$$

where:

  - $\sigma_j = mgu(t, u_j)$, $(n, n_j) \in E$, and $[\![\Phi_j \sigma_j]\!]_G = 1$ for any $j \in J$,
  - if $(n, n_I) \in E$ for some $n_I \in \mathcal{M}$ then $\mathcal{I}' = \mathcal{I} \cup \{t\}$ else $\mathcal{I}' = \mathcal{I}$.

Moreover, we know that $\lfloor P' \rfloor_{n'} \in \mathcal{Q}$ implies that:

  - $(n, n') \notin E$, or
  - P' is not of the form $\mathsf{in}\ u'[\Phi'].Q'$, or
  - $P' = \mathsf{in}\ u'[\Phi'].Q'$ and $(mgu(t, u') = \bot$ or $[\![\Phi' mgu(t, u')]\!]_G = 0)$.

We know that K_s is a symbolic configuration whose θ-concretization is K_c. Thus, we have that $K_s = (\lfloor \mathsf{out}(t_s).P_s \rfloor_n \cup \{\lfloor \mathsf{in}\ u^s_j[\Phi^s_j].P^s_j \rfloor_{n_j} | j \in J\} \cup \mathcal{Q}_s; \mathcal{S}_s; \mathcal{I}_s; \mathcal{C}; \Psi)$ with $t_s\theta = t$, $P_s\theta = P$, $\mathcal{I}_s\theta = \mathcal{I}$, $\mathcal{S}_s\theta = \mathcal{S}$, $\mathcal{Q}_s\theta = \mathcal{Q}$ and for any $j \in J$, we have that $u^s_j\theta = u_j$, $\Phi^s_j\theta = \Phi_j$, and $P^s_j\theta = P_j$. Let us define

  - $\mathcal{P}^s_{K,L} = \{\lfloor \mathsf{in}\ u^s_k[\Phi^s_k].P^s_k \rfloor_{n_k} \in \mathcal{Q}_s \mid (n_k, n) \in E\}$,
  - $\mathcal{Q}'_s$ be such that $\mathcal{Q}_s = \mathcal{P}^s_{K,L} \uplus \mathcal{Q}'_s$,
  - $K = \{k \mid \lfloor \mathsf{in}\ u^s_k[\Phi^s_k].P^s_k \rfloor_{n_k} \in \mathcal{P}^s_{K,L}$ and $mgu(t_s\theta, u^s_k\theta) = \bot\}$,
  - $L = \{l \mid \lfloor \mathsf{in}\ u^s_l[\Phi^s_l].P^s_l \rfloor_{n_l} \in \mathcal{P}^s_{K,L},\ \sigma_l = mgu(t_s\theta, u^s_l\theta)$ and $\neg[\![(\Phi^s_l\theta)\sigma'_l]\!]_G = 1\}$.

We have that $\mathcal{P}^s_{K,L} = \{\lfloor \mathsf{in}\ u^s_k[\Phi^s_k].P^s_k \rfloor_{n_k} \in \mathcal{P}^s_{K,L} \mid k \in K \uplus L\}$.

Let α be a renaming of $\{var(u_l) \mid l \in L\} \smallsetminus rvar(\mathcal{C})$. Let $\sigma = \bigcup_{j \in J} \sigma_j \cup \bigcup_{l \in L} \sigma_l$. We show that there exists a substitution $\sigma' = mgu(\{u^s_j = t_s \mid j \in J\} \cup \{u^s_l\alpha = t_s \mid l \in L\})$. To achieve this result, we show that $\sigma \circ \alpha^{-1} \circ \theta$ is a unifier of $\{u^s_j = t_s \mid j \in J\} \cup \{u^s_l\alpha = t_s \mid l \in L\}$:

  - $\forall j \in J, \sigma_j = mgu(u_j, t)$. We have that $u^s_j\theta = u_j$ and $t_s\theta = t$. As $dom(\alpha^{-1})$ includes only fresh variables, $u_j\alpha^{-1} = u_j$ and $t\alpha^{-1} = t$. Consequently, $((u^s_j\theta)\alpha^{-1})\sigma = ((t_s\theta)\alpha^{-1})\sigma$
  - $\forall l \in L, \sigma_l = mgu(t_s\theta, u^s_l\theta)$ exists. We have that $dom(\theta) = rvar(\mathcal{C})$ and $img(\theta) \cap \mathcal{X} = \emptyset$. Moreover, $dom(\alpha) = \{var(u_l) \mid l \in L\} \smallsetminus rvar(\mathcal{C})$ and $img(\alpha)$ is a set of fresh variables. Hence, $\theta \circ \alpha = \alpha \circ \theta$. We deduce that $(((u^s_l\alpha)\theta)\alpha^{-1})\sigma = (u^s_l\theta)\sigma = (t_s\theta)\sigma = (((t_s\alpha)\theta)\alpha^{-1})\sigma$

We have proven that $\sigma \circ \alpha^{-1} \circ \theta$ is a unifier of $\{u^s_j = t_s \mid j \in J\} \cup \{u^s_l\alpha = t_s \mid l \in L\}$. Consequently, there exists σ', θ' such that $\sigma' = mgu(\{u^s_j = t_s \mid j \in J\} \cup \{u^s_l\alpha = t_s \mid l \in L\})$ and $\theta' \circ \sigma' = \sigma \circ \alpha^{-1} \circ \theta$.

Let $K'_s = (\lfloor P_s\sigma' \rfloor_n \cup \mathcal{P}^s_J\sigma' \cup \mathcal{P}^s_{K,L}\sigma' \cup \mathcal{Q}'_s\sigma'; \mathcal{S}_s\sigma'; \mathcal{I}'_s\sigma'; \mathcal{C}\sigma'; \Psi'\sigma')$ where:

  - $\mathcal{P}^s_J = \{\lfloor P^s_j \rfloor_{n_j} \mid j \in J\}$ and $\Psi_J = \{\Phi^s_j \mid j \in J\}$,
  - $\Psi_K = \{\forall Y_k \,.\, t_s \neq u^s_k \mid k \in K\}$ with $Y_k = var(u_k) \smallsetminus rvar(\mathcal{C})$,

- $\Psi_L = \{\neg\Phi_l^s\alpha \mid l \in L\}$
- $\mathcal{I}_s' = \mathcal{I}_s \cup \{t\}$ if $(n, n_I) \in E$ and $\mathcal{I}_s' = \mathcal{I}_s$ otherwise.
- $\Psi' = (\Psi \wedge \Psi_J \wedge \Psi_K \wedge \Psi_L)$,

Clearly, we have that $K_s \rightarrow^s_{G,\mathcal{M}} K_s'$. To conclude, it remains to show that K_c' is the θ'-concretization of K_s'.

First, as θ is a solution of $\mathcal{C}$ and $\theta' \circ \sigma' = \sigma \circ \alpha^{-1} \circ \theta$, it is straightforward to see that θ' is a solution of $\mathcal{C}' = \mathcal{C}\sigma'$. Similarly, θ' is a solution of $\Psi\sigma'$.

It remains to establish that:

- θ' is a solution of $\Psi_J\sigma'$. For every $j \in J$, $[\![\Phi_j\sigma_j]\!]_G = 1$, and $\Phi_j = \Phi_j^s\theta = (\Phi_j^s\theta)\alpha^{-1}$, so $[\![(\Phi_j\sigma')\theta']\!]_G = 1$.
- θ' is a solution of $\Psi_K\sigma'$, i.e. θ' satisfies $\forall var(u_k) \smallsetminus rvar(\mathcal{C})\,.\, t_s\sigma' \neq u_k^s\sigma'$ for any $k \in K$. This is true since $\theta' \circ \sigma' = \sigma \circ \alpha^{-1} \circ \theta$ and $mgu(t_s\theta, u_k^s\theta) = \perp$ for any $k \in K$.
- θ' is a solution of $\Psi_L\sigma'$, i.e. $[\![((\Phi_l^s\alpha)\sigma')\theta']\!]_G = 0$ for any $l \in L$. By definition of L, $[\![(\Phi_l^s\theta)\sigma]\!]_G = 0$. We have that $((\Phi_l^s\alpha)\sigma')\theta' = (((\Phi_l^s\alpha)\theta)\alpha^{-1})\sigma = (\Phi_l^s\theta)\sigma$. Hence, we have that $[\![((\Phi_l^s\alpha)\sigma')\theta']\!]_G = 0$ for any $l \in L$.

Lastly, it remains to verify that K_c' is the θ'-concretization of K_s'. Indeed, we have that:

- $(P_s\sigma')\theta' = ((P_s\theta)\alpha^{-1})\sigma = P_s\theta = P$,
- $(P_j^s\sigma')\theta' = ((P_j^s\theta)\alpha^{-1})\sigma = (P_j^s\theta)\sigma_j = P_j\sigma_j$ for any $j \in J$,
- $((\mathcal{P}_{K,L}^s \cup \mathcal{Q}_s')\sigma')\theta' = (\mathcal{Q}_s\sigma')\theta' = ((\mathcal{Q}_s\theta)\alpha^{-1})\sigma = \mathcal{Q}_s\theta = \mathcal{Q}$,
- $(\mathcal{S}_s\sigma')\theta' = ((\mathcal{S}_s\theta)\alpha^{-1})\sigma = \mathcal{S}_s\theta = \mathcal{S}$,
- $(\mathcal{I}_s'\sigma')\theta' = ((\mathcal{I}_s'\theta)\alpha^{-1})\sigma = \mathcal{I}_s'\theta = \mathcal{I}'$.

This allows us to conclude.

□

Conversely, we have that each symbolic transition can be instantiated in a concrete one. The proof is again obtained by inspection of the rules. We deduce from these two propositions that checking for a concrete attack can be reduced to checking for a symbolic one.

Proposition 3.2.3 (soundness). *Let $G = (\mathcal{N}_{\mathsf{loc}}, E)$ be a graph and $\mathcal{M} \subseteq \mathcal{N}_{\mathsf{loc}}$. Let $K_s = (\mathcal{P}_s; \mathcal{S}_s; \mathcal{I}_s; \mathcal{C}; \Psi)$ and $K_s' = (\mathcal{P}_s'; \mathcal{S}_s'; \mathcal{I}_s'; \mathcal{C}'; \Psi')$ be two ground symbolic configurations, such that $K_s \rightarrow^s_{G,\mathcal{M}} K_s'$. Let θ' be a substitution and let K_c' be the θ'-concretization of K_s'. There exists a substitution θ and a ground configuration K_c such that*

- *K_c is the θ-concretization of K_s.*
- *$K_c \rightarrow_{G,\mathcal{M}} K_c'$, and*

Proof. There exists $\mathcal{I}_{names}, \mathcal{I}_{\mathsf{terms}}$ such that $\mathcal{I}_s = \mathcal{I}_{\mathsf{terms}} \uplus \mathcal{I}_{names}$ and $(\mathcal{C}, \mathcal{I}_{names})$ is a constraint system. As K_c' is the θ'-concretization of K_s', θ' is a solution of $(\mathcal{C}', \mathcal{I}_{names})$ and Ψ'. To prove the proposition, we define first a substitution θ, solution of $(\mathcal{C}, \mathcal{I}_{names})$ and Ψ, and we consider K_c the θ-concretization of K_s. We distinguish several cases, depending on the rule involved in the transition $K_s \rightarrow^s_{G,\mathcal{M}} K_s'$.

- Rule Par$_s$. We have that:
$$(\lfloor P_1^s | P_2^s \rfloor_n \cup \mathcal{Q}_s; \mathcal{S}_s; \mathcal{I}_s; \mathcal{C}; \Psi) \rightarrow^s_{G,\mathcal{M}} (\lfloor P_1^s \rfloor_n \cup \lfloor P_2^s \rfloor_n \cup \mathcal{Q}_s; \mathcal{S}_s; \mathcal{I}_s; \mathcal{C}; \Psi)$$
Since K'_c is the θ'-concretization of K'_s, we have that:
$$K'_c = (\lfloor P_1^s \theta' \rfloor_n \cup \lfloor P_2^s \theta' \rfloor_n \cup \mathcal{Q}_s\theta'; \mathcal{S}_s\theta'; \mathcal{I}_s\theta')$$
Since $\mathcal{C}' = \mathcal{C}$ and $\Psi' = \Psi$, we can choose $\theta = \theta'$ solution of $(\mathcal{C}, \mathcal{I}_{names})$ and Ψ. Let K_c be the θ-concretization of K_s.
$$K_c = (\lfloor P_1^s \theta' | P_2^s \theta' \rfloor_n \cup \mathcal{Q}_s\theta'; \mathcal{S}_s\theta'; \mathcal{I}_s\theta')$$
We have $K_c \rightarrow_{G,\mathcal{M}} K'_c$ (by the Par rule).

- Rule Repl$_s$. We have that:
$$(\lfloor !P_s \rfloor_n \cup \mathcal{Q}_s; \mathcal{S}_s; \mathcal{I}_s; \mathcal{C}; \Psi) \rightarrow^s_{G,\mathcal{M}} (\lfloor P_s\alpha_s \rfloor_n \cup \lfloor !P_s \rfloor_n \cup \mathcal{Q}_s; \mathcal{S}_s; \mathcal{I}_s; \mathcal{C}; \Psi)$$
where α_s is a renaming of the bound variables of P_s that are not in $rvar(\mathcal{C})$.

  Since K'_c is the θ'-concretization of K'_s, we have that:
$$K'_c = (\lfloor (P_s\alpha_s)\theta' \rfloor_n \cup \lfloor !P_s\theta' \rfloor_n \cup \mathcal{Q}_s\theta'; \mathcal{S}_s\theta'; \mathcal{I}_s\theta')$$
Since $\mathcal{C}' = \mathcal{C}$ and $\Psi' = \Psi$, we can choose $\theta = \theta'$ solution of $(\mathcal{C}, \mathcal{I}_{names})$ and Ψ. Let K_c be the θ-concretization of K_s.
$$K_c = (\lfloor !P_s\theta \rfloor_n \cup \mathcal{Q}_s\theta; \mathcal{S}_s\theta; \mathcal{I}_s\theta).$$
To show that $K_c \rightarrow_{G,\mathcal{M}} K'_c$ (by the Repl rule), it remains to prove that:
  - $(P_s\theta)\alpha_s = (P_s\alpha_s)\theta$. This equality comes from the fact $dom(\theta) \cap dom(\alpha_s) = \emptyset$.
  - α_s is a renaming of $bv(P_s\theta)$. This is due to the fact that α_s is renaming of the bound variables of P_s that are not in $rvar(\mathcal{C})$ and $dom(\theta) = rvar(\mathcal{C})$.

- Rule New$_s$. We have that:
$$(\lfloor \mathsf{new}\ m.P_s \rfloor_n \cup \mathcal{Q}_s; \mathcal{S}_s; \mathcal{I}_s; \mathcal{C}; \Psi) \rightarrow^s_{G,\mathcal{M}} (\lfloor P_s\{m \mapsto m'\} \rfloor_n \cup \mathcal{Q}_s; \mathcal{S}_s; \mathcal{I}_s; \mathcal{C}; \Psi)$$
where m' is a fresh name.

  As in the previous cases, we have that K'_c is the θ'-concretization of K'_s. Moreover, since $\mathcal{C}' = \mathcal{C}$ and $\Psi' = \Psi$, we can choose $\theta = \theta'$. Let K_c be the θ-concretization of K_s. Hence, we have that:
  - $K'_c = (\lfloor ((P_s\{m \mapsto m'\})\theta) \rfloor_n \cup \mathcal{Q}_s\theta; \mathcal{S}_s\theta; \mathcal{I}_s\theta)$,
  - $K_c = (\lfloor \mathsf{new}\ m.P_s\theta \rfloor_n \cup \mathcal{Q}_s\theta; \mathcal{S}_s\theta; \mathcal{I}_s\theta)$.

  As in the previous case, since m' is a fresh name and $(P_s\theta)\{m \mapsto m'\} = (P_s\{m \mapsto m'\})\theta$, we have that $K_c \rightarrow_{G,\mathcal{M}} K'_c$ (by the New rule).

- Rule Store$_s$. We have that:
$$(\lfloor \mathsf{store}(t_s).P_s \rfloor_n \cup \mathcal{Q}_s; \mathcal{S}_s; \mathcal{I}_s; \mathcal{C}; \Psi) \rightarrow^s_{G,\mathcal{M}} (\lfloor P_s \rfloor_n \cup \mathcal{Q}_s; \lfloor t_s \rfloor_n \cup \mathcal{S}_s; \mathcal{I}_s; \mathcal{C}; \Psi)$$
As in the previous cases, we have that K'_c is the θ'-concretization of K'_s. Moreover, since $\mathcal{C}' = \mathcal{C}$ and $\Psi' = \Psi$, we can choose $\theta = \theta'$. Let K_c be the θ-concretization of K_s. Hence, we have that:
  - $K_c = (\lfloor \mathsf{store}(t_s\theta).(P_s\theta) \rfloor_n \cup \mathcal{Q}_s\theta; \mathcal{S}_s\theta; \mathcal{I}_s\theta)$,
  - $K'_c = (\lfloor P_s\theta' \rfloor_n \cup \mathcal{Q}_s\theta'; \lfloor t_s\theta' \rfloor_n \cup \mathcal{S}_s\theta'; \mathcal{I}_s\theta')$.

  We have that $K_c \rightarrow_{G,\mathcal{M}} K'_c$ (by the Store rule).

- Rule READ-THEN$_s$. We have that:

$$\begin{aligned}&(\lfloor \mathsf{read}\ u_s\ \mathsf{then}\ P_s\ \mathsf{else}\ Q_s\rfloor_n \cup \mathcal{Q}_s;\lfloor t_s\rfloor_n \cup \mathcal{S}_s;\mathcal{I}_s;\mathcal{C};\Psi)\\ &\qquad\to^s_{G,\mathcal{M}} (\lfloor P_s\sigma'\rfloor_n \cup \mathcal{Q}_s\sigma';\lfloor t_s\sigma'\rfloor_n \cup \mathcal{S}_s\sigma';\mathcal{I}_s\sigma';\mathcal{C}\sigma';\Psi\sigma')\end{aligned}$$

  where $\sigma' = mgu(u_s, t_s)$

  Since K'_c is the θ'-concretization of K'_s, we have that:

$$K'_c = (\lfloor (P_s\sigma')\theta'\rfloor_n \cup (\mathcal{Q}_s\sigma')\theta';\lfloor (t_s\sigma')\theta'\rfloor_n \cup (\mathcal{S}_s\sigma')\theta';(\mathcal{I}_s\sigma')\theta')$$

  Let $\theta = \theta' \circ \sigma'$. θ is a solution of $(\mathcal{C}, \mathcal{I}_{names})$ and Ψ. Let K_c be the θ-concretization of K_s.

$$K_c = (\lfloor \mathsf{read}\ u_s\theta\ \mathsf{then}\ P_s\theta\ \mathsf{else}\ Q_s\theta\rfloor_n \cup \mathcal{Q}_s\theta;\lfloor t_s\theta\rfloor_n \cup \mathcal{S}_s\theta;\mathcal{I}_s\theta).$$

  We have that $u_s\theta = t_s\theta$. Hence, we have that:

$$K_c \to_{G,\mathcal{M}} (\lfloor (P_s\theta)\sigma\rfloor_n \cup \mathcal{Q}_s\theta;\lfloor t_s\theta\rfloor_n \cup \mathcal{S}_s\theta;\mathcal{I}_s\theta)$$

  by the READ-THEN rule with $\sigma = Id$. Since K_s is a ground symbolic configuration, we know that $var(\mathcal{I}_s) \cup fv(\mathcal{Q}_s) \cup var(\lfloor t_s\rfloor_n \cup \mathcal{S}_s) \subseteq dom(\theta)$, and thus, $K_c \to_{G,\mathcal{M}} K'_c$ (by the READ-THEN rule).

- Rule READ-ELSE$_s$. We have that:

$$\begin{aligned}K_s = &(\lfloor \mathsf{read}\ u_s\ \mathsf{then}\ P_s\ \mathsf{else}\ Q_s\rfloor_n \cup \mathcal{Q}_s;\mathcal{S}_s;\mathcal{I}_s;\mathcal{C};\Psi)\\ &\qquad\to^s_{G,\mathcal{M}} (\lfloor Q_s\rfloor_n \cup \mathcal{Q}_s;\mathcal{S}_s;\mathcal{I}_s;\mathcal{C};\Psi \wedge \mathsf{Eq}) = K'_s\end{aligned}$$

  where $\mathsf{Eq} = \{\forall var(u_s) \setminus rvar(\mathcal{C})\ .\ t_s \neq u_s \mid \lfloor t_s\rfloor_n \in \mathcal{S}_s\}$.

  We can choose $\theta = \theta'$. θ is a solution of $(\mathcal{C}', \mathcal{I}_{names})$ and Ψ'. In particular, θ is a solution of $(\mathcal{C}, \mathcal{I}_{names})$ and Ψ.

  As in the previous cases, we have that K'_c is the θ'-concretization of K'_s. Let K_c be the θ-concretization of K_s. Hence, we have that:

  - $K_c = (\lfloor \mathsf{read}\ u_s\theta\ \mathsf{then}\ P_s\theta\ \mathsf{else}\ Q_s\theta\rfloor_n \cup \mathcal{Q}_s\theta;\mathcal{S}_s\theta;\mathcal{I}_s\theta)$,
  - $K'_c = (\lfloor Q_s\theta'\rfloor_n \cup \mathcal{Q}_s\theta';\mathcal{S}_s\theta';\mathcal{I}_s\theta')$.

  Furthermore, we have that $u_s\theta$ is not unifiable with $t_s\theta$ for any $\lfloor t_s\rfloor_n \in \mathcal{S}_s$. In other words, $mgu(u_s\theta, t) = \perp$ for any t such that $\lfloor t\rfloor_n \in \mathcal{S}_s\theta$. Hence, we have that $K_c \to_{G,\mathcal{M}} K'_c$ by the READ-ELSE rule.

- Rule IF-THEN$_s$. We have that:

$$\begin{aligned}K_s = &(\lfloor \mathsf{if}\ \Phi_s\ \mathsf{then}\ P_s\ \mathsf{else}\ Q_s\rfloor_n \cup \mathcal{Q}_s;\mathcal{S}_s;\mathcal{I}_s;\mathcal{C};\Psi)\\ &\qquad\to^s_{G,\mathcal{M}} (\lfloor P_s\rfloor_n \cup \mathcal{Q}_s;\mathcal{S}_s;\mathcal{I}_s;\mathcal{C};\Psi \wedge \Phi_s) = K'_s\end{aligned}$$

  We can choose $\theta = \theta'$. θ is a solution of $(\mathcal{C}', \mathcal{I}_{names})$ and Ψ'. In particular, θ is a solution of $(\mathcal{C}, \mathcal{I}_{names})$ and Ψ. As in the previous cases, we have that K'_c is the θ'-concretization of K'_s. Let K_c be the θ-concretization of K_s. Hence, we have that:

  - $K_c = (\lfloor \mathsf{if}\ \Phi_s\theta\ \mathsf{then}\ P_s\theta\ \mathsf{else}\ Q_s\theta\rfloor_n \cup \mathcal{Q}_s\theta;\mathcal{S}_s\theta;\mathcal{I}_s\theta)$,
  - $K'_c = (\lfloor P_s\theta\rfloor_n \cup \mathcal{Q}_s\theta;\mathcal{S}_s\theta;\mathcal{I}_s\theta)$.

  Moreover, since θ is a solution of Φ_s, we have that $[\![\Phi_s\theta]\!] = 1$. Hence, we have that $K_c \to_{G,\mathcal{M}} K'_c$ by the IF-THEN rule.

- Rule IF-ELSE$_s$. This case is similar to the previous one.
- Rule IN$_s$. We have that:

$$(\lfloor \mathsf{in}\ u_s[\Phi_s].P_s\rfloor_n \cup \mathcal{Q}_s; \mathcal{S}_s; \mathcal{I}_s; \mathcal{C}; \Psi) \rightarrow^s_{G,\mathcal{M}} (\lfloor P_s\rfloor_n \cup \mathcal{Q}_s; \mathcal{S}_s; \mathcal{I}_s; \mathcal{C}'; \Psi')$$

  where $\mathcal{C}' = \mathcal{C} \wedge \mathcal{I}_s \overset{?}{\vdash} u_s$ and $\Psi' = \Psi \wedge \Phi_s$ and $(n_I, n) \in E$ for some $n_I \in \mathcal{M}$. We choose $\theta = \theta'$, solution of $(\mathcal{C}, \mathcal{I}_{names})$ and Ψ.

  As in the previous cases, we have that K'_c is the θ'-concretization of K'_s. Let K_c be the θ-concretization of K_s. Hence, we have that:
  - $K_c = (\lfloor \mathsf{in}\ u_s\theta[\Phi_s\theta].P_s\theta\rfloor_n \cup \mathcal{Q}_s\theta; \mathcal{S}_s\theta; \mathcal{I}_s\theta)$,
  - $K'_c = (\lfloor P_s\theta'\rfloor_n \cup \mathcal{Q}_s\theta'; \mathcal{S}_s\theta'; \mathcal{I}_s\theta')$.

  Since θ' is a solution of $(\mathcal{C}', \mathcal{I}_{names})$, we have that $\mathcal{I}_s\theta' \vdash u_s\theta'$ and $[\![\Phi_s\theta']\!]_G = 1$. Thus, $\mathcal{I}_s\theta \vdash u_s\theta = t$ and $[\![\Phi_s\theta]\!]_G = 1$. Hence, we have that $K_c \rightarrow_{G,\mathcal{M}} K'_c$ by the IN rule.

- Rule COMM$_s$. We have that:

$$(\mathcal{P}^s_I \cup \lfloor \mathsf{out}(t_s).P_s\rfloor_n \cup \mathcal{Q}_s; \mathcal{S}_s; \mathcal{I}_s; \mathcal{C}; \Psi) \rightarrow^s_{G,\mathcal{M}} (\mathcal{P}^s_J \cup \mathcal{P}^s_{K,L} \cup \lfloor P_s\sigma\rfloor_n \cup \mathcal{Q}_s\sigma; \mathcal{S}_s\sigma; \mathcal{I}'_s; \mathcal{C}'; \Psi')$$

  where:

  - $\mathcal{P}^s_I = \{\lfloor \mathsf{in}\ u^s_i[\Phi^s_i].P^s_i\rfloor_{n_i} |\ (n_i, n) \in E, i \in I\}$,
  - $I = J \uplus K \uplus L$,
  - $\sigma = mgu(\{u_j = t \mid j \in J\} \cup \{u_l\alpha \mid l \in L\})$
  - $\mathcal{P}^s_J = \{\lfloor P^s_j\rfloor_{n_j}\sigma |\ j \in J\}$,
  - $\mathcal{P}^s_{K,L} = \{\lfloor(\mathsf{in}\ u_k[\Phi^s_k].P^s_k\rfloor_{n_k})\sigma |\ k \in K \uplus L\}$,
  - $\mathcal{C}' = \mathcal{C}\sigma$ and $\Psi' = (\Psi \wedge \Psi_J \wedge \Psi_K \wedge \Psi_L)\sigma$,
  - $\Psi_J = \{\Phi^s_j \mid j \in J\}$, $\Psi_K = \{\forall var(u^s_k) \smallsetminus rvar(\mathcal{C})\,.\, t_s \neq u^s_k \mid k \in K\}$, and $\Psi_L = \{\Phi^s_l\alpha_l \mid l \in L\}$ where α_l is a renaming of $var(u^s_l) \smallsetminus rvar(\mathcal{C})$ by fresh variables.
  - $\mathcal{I}'_s = (\mathcal{I}_s \cup \{t_s\})\sigma$ if $(n, n_I) \in E$ for some $n_I \in \mathcal{M}$ and $\mathcal{I}'_s = \mathcal{I}_s\sigma$ otherwise.

  Moreover, $\lfloor Q_s\rfloor_{n'} \in \mathcal{Q}_s$ implies that $(n, n') \notin E$ or Q_s is not of the form $\mathsf{in}\ u'_s[\Phi'_s].Q'_s$. We have also that $(n_i, n) \in E$ for every $i \in I$.

  Let $\theta = \theta' \circ \sigma$. Since θ' is a solution of $(\mathcal{C}\sigma, \mathcal{I}_{names})$ and $\Psi' \supseteq \Psi\sigma$, it is clear that θ is a solution of $(\mathcal{C}, \mathcal{I}_{names})$ and Ψ.

  As in the previous cases, we have that K'_c is the θ'-concretization of K'_s. Let K_c be the θ-concretization of K_s. Hence, we have that:
  - $K_c = (\mathcal{P}^s_I\theta \cup \lfloor \mathsf{out}(t_s\theta).P_s\theta\rfloor_n \cup \mathcal{Q}_s\theta; \mathcal{S}_s\theta; \mathcal{I}_s\theta)$,
  - $K'_c = (\mathcal{P}^s_J\theta' \cup \mathcal{P}^s_{K,L}\theta' \cup \lfloor P_s\sigma\theta'\rfloor_n \cup \mathcal{Q}_s\sigma\theta'; \mathcal{S}_s\sigma\theta'; \mathcal{I}'_s\theta')$.

  To conclude, it remains to show that $K_c \rightarrow_{G,\mathcal{M}} K'_c$. First, we have that:
  - $\mathcal{S}_s\sigma\theta' = \mathcal{S}_s\theta$,
  - if $(n, n_I) \in E$ for some $n_I \in \mathcal{M}$ then $\mathcal{I}'_s\theta' = (\mathcal{I}_s \cup \{t_s\})\sigma\theta' = \mathcal{I}_s\theta \cup \{t_s\theta\}$. Otherwise, we have that $\mathcal{I}'_s\theta' = \mathcal{I}_s\sigma\theta' = \mathcal{I}_s\theta$.

– $P_s\sigma\theta' = P_s\theta$, $\mathcal{Q}_s\sigma\theta' = \mathcal{Q}_s\theta$, and $\mathcal{P}^s_{K,L}\theta' = \mathcal{P}^s_{K,L}\theta$ (thanks to the renaming α_l).

Note also that the processes in $\mathcal{Q}_s\theta$ are not of the right form to evolve by receiving a message from the node n. Thus, to show that $K_c \rightarrow_G K'_c$, it remains to prove that $J = J'$ where

$$J' = \{i \mid \lfloor \mathsf{in}\ u^s_i[\Phi^s_i].P^s_i\rfloor_{n_i} \in \mathcal{P}^s_I, \overline{\sigma}_i = mgu(t\theta, u^s_i\theta) \text{ exists }, [\![(\Phi^s_i\theta)\overline{\sigma}_i]\!]_G = 1\}.$$

We prove the two inclusions separately.

First, we show that $J \subseteq J'$. Let $i \in J$. We know that $\lfloor \mathsf{in}\ u^s_i[\Phi^s_i].P^s_i\rfloor_{n_i} \in \mathcal{P}^s_I$. By definition of σ, $u^s_i\sigma = t\sigma$. Consequently, $u^s_i\theta = u^s_i\sigma\theta' = t\sigma\theta' = t\theta$. So $\overline{\sigma}_i = mgu(t\theta, u^s_i\theta)$ exists and $\overline{\sigma}_i = Id$. Since θ' is a solution of Ψ', we have that $[\![\Phi^s_i\sigma\theta']\!]_G = 1$. We deduce that $[\![\Phi^s_i\theta]\!]_G = 1$. This allows us to conclude that $i \in J'$.

Now, we show that $J' \subseteq J$. Let $i \in J'$. We have that $\lfloor \mathsf{in}\ u^s_i[\Phi^s_i].P^s_i\rfloor_{n_i} \in \mathcal{P}^s_I$, Hence, we have that $i \in I$. In order to conclude that $i \in J$, it is sufficient to show that $i \notin K$ and $i \notin L$.

1. $i \notin K$. By contradiction, assume that $i \in K$. Since θ' is a solution of $\Psi_K\sigma$, we have that θ' satisfies the constraint $\forall var(u^s_i) \smallsetminus rvar(\mathcal{C})\,.\, t_s\sigma \neq u^s_i\sigma$. This implies that $t_s\theta$ and $u^s_i\theta$ are not unifiable This is impossible since we know that $\overline{\sigma}_i = mgu(t\theta, u^s_i\theta)$ exists. Contradiction. Hence, we deduce that $i \notin K$.
2. $i \notin L$. By contradiction, assume that $i \in L$. Since θ' is a solution of $\Psi_L\sigma$, we have that $t\sigma\theta' = (u^s_i\alpha_i)\sigma\theta'$ and $[\![(\Phi^s_i\alpha_i)\sigma\theta']\!]_G = 0$. Actually, we have that:
$$(u^s_i\alpha_i)\theta = (u^s_i\theta)\alpha_i = (t\theta)\alpha_i.$$
Hence, we have that $\overline{\sigma}_i = \alpha_i$. We have also that:
$$(\Phi^s_i\alpha_i)\theta = ((\Phi^s_i\theta)\alpha_i).$$
We deduce that $[\![(\Phi^s_i\theta)\overline{\sigma}_i]\!]_G = 0$. Contradiction. Hence, we have that $i \notin L$.

This allows us to conclude that $K_c \rightarrow_{G,\mathcal{M}} K'_c$. □

These lemmas allow us to show that to a concrete derivation corresponds a symbolic one. Thus, checking for attacks can be done in the symbolic model, as shown in Theorem 3.2.4.

Theorem 3.2.4. *Let $G = (\mathcal{N}_{\mathsf{loc}}, E)$ be a graph and $\mathcal{M} \subseteq \mathcal{N}_{\mathsf{loc}}$. Let $K = (\mathcal{P}[\_];\mathcal{S};\mathcal{I})$ be a ground concrete configuration with a hole, and Φ be a formula. There is an $\mathcal{M}$-attack on K and Φ for graph G if, and only if,*

$$(\mathcal{P}[\mathsf{if}\ \Phi\ \mathsf{then}\ \mathsf{out}(\mathsf{error})];\mathcal{S};\mathcal{I};\emptyset;\emptyset) \quad \rightarrow^{s*}_{G,\mathcal{M}} \quad (\lfloor \mathsf{out}(u)\rfloor_n \cup \mathcal{P}_s;\mathcal{S}_s;\mathcal{I}_s;\mathcal{C};\Psi)$$

with $\mathcal{I}_s = \mathcal{I}_{names} \uplus \mathcal{I}_{\mathsf{terms}}$ and $\sigma = mgu(u, \mathsf{error})$ exists and the constraint system $(\mathcal{C}\sigma, \mathcal{I}_{names})$ with $\Psi\sigma$ has a solution for graph G.

Proof. We show the two directions separately.

($\Rightarrow$) First, let us suppose that there is an attack on K and Φ for graph G. By definition of an attack, there exists a concrete configuration K' such that:

- K' is of the form $(\lfloor \mathsf{out}(\mathsf{error})\rfloor_n \cup \mathcal{P}'; \mathcal{S}'; \mathcal{I}')$, and
- $K \rightarrow_G^* K'$.

By applying Proposition 3.2.2 recursively, we deduce that there exist a ground symbolic configuration K'_s and a substitution θ' such that:

- $(\mathcal{P}[\mathsf{if}\ \Phi\ \mathsf{then}\ \mathsf{out}(\mathsf{error})\ \mathsf{else}\ 0]; \mathcal{S}; \mathcal{I}; \emptyset; \emptyset) \rightarrow_G^{s*} K'_s$, and
- K' is the θ'-concretization of K'_s.

Consequently, K'_s is of the form $(\lfloor \mathsf{out}(u)\rfloor_n \cup \mathcal{P}'_s; \mathcal{S}'_s; \mathcal{I}'_s; \mathcal{C}'; \Psi')$, θ' is a solution of $\mathcal{C}'$ and Ψ' for G, and $u\theta' = \mathsf{error}$. Hence, $\sigma = mgu(u, \mathsf{error})$ exists and there exists a substitution θ' such that $\theta = \theta' \circ \sigma$. We conclude that θ' is a solution of $(\mathcal{C}\sigma, \mathcal{I}_{names})$ and $\Psi\sigma$ for graph G.

($\Leftarrow$) Conversely, assume that

$$K_s = (\mathcal{P}[\mathsf{if}\ \Phi\ \mathsf{then}\ \mathsf{out}(\mathsf{error})\ \mathsf{else}\ 0]; \mathcal{S}; \mathcal{I}; \emptyset; \emptyset)\ \rightarrow_G^{s*}\ (\lfloor \mathsf{out}(u)\rfloor_n \cup \mathcal{P}_s; \mathcal{S}_s; \mathcal{I}_s; \mathcal{C}; \Psi) = K'_s$$

and θ' is a solution of $(\mathcal{C}\sigma, \mathcal{I}_{names})$ and $\Psi\sigma$ where $\sigma = mgu(u, \mathsf{error})$.

First, we have that K_s is a ground symbolic configuration whose concretization is $K = (\mathcal{P}[\mathsf{if}\ \Phi\ \mathsf{then}\ \mathsf{out}(\mathsf{error})\ \mathsf{else}\ 0]; \mathcal{S}; \mathcal{I})$. Thanks to Lemma 3.2.1, we know that the symbolic configurations involved in this derivation are ground. Furthermore, $\theta'' = \theta' \circ \sigma$ is a solution of $(\mathcal{C}, \mathcal{I}_{names})$ and Ψ. Let K' be the θ''-concretization of K'_s, as $u\theta'' = (u\sigma)\theta' = \mathsf{error}$, we have that:

$$K' = (\lfloor \mathsf{out}(\mathsf{error})\rfloor_n \cup \mathcal{P}_s\theta''; \mathcal{S}_s\theta''; \mathcal{I}_s\theta'')$$

Hence, by applying recursively Proposition 3.2.3, we know that there exists a substitution θ and a ground concrete configuration K such that:

- K is the θ-concretization of K_s,
- $K \rightarrow_G^* K'$.

Hence, there is an attack on K and Φ for graph G. □

Note that our result holds for any signature, for any choice of predicates, and for processes possibly with replication. Of course, it then remains to decide the existence of a constraint system that has a solution.

Example 3.2.3. *Consider our former example of an attack on SRP, with initial configuration K_0. We can reach the configuration K_s, and the constraint system $\mathcal{C}$ has a solution σ for graph G_0 (cf. Example 3.2.1), so there is an $\{n_I\}$-attack on K_0 for G_0.*

3.3 Bounding the size of minimal solutions for solved forms

Applying the technique described in Section 3.2, we are left to decide the existence of a solution for a constraint system $(\mathcal{C}, \mathcal{I}_{names})$ together with disequality constraints and formulas of $\mathcal{L}_{\mathsf{route}}$.

In Chapter 2, we have developed a technique that allows us to consider only constraint systems in solved form.

In this section, we show how to bound the size of a minimal solution for solved constraint systems. First, we have in Subsection 3.3.1 preliminary results about substitutions, showing that applying substitutions does not increase the number of subterms. Then we have two propositions, corresponding to our two decidability results, in order to take into account a fixed topology as well as (a priori unbounded) unknown topology.

3.3.1 Preliminary results regarding substitutions

All throughout our procedures, we apply substitutions, and more precisely mgus of terms, to the systems we consider. We have to control the size of the terms with respect to the size of the inputs. Thus we need the following results to know how we can bound the size of terms in a system where we just applied a substitution.

In order to obtain these results, we use the following rules for computing an mgu, called DAG syntactic unification [JK91].

DELETE	$P \cup \{s = s\} \implies P$
DEC.	$P \cup \{f(s_1, \dots, s_n) = f(t_1, \dots, t_n)\} \implies P \cup \{s_1 = t_1, \dots, s_n = t_n\}$
CONF.	$P \cup \{f(s_1, \dots, s_n) = g(t_1, \dots, t_k)\} \implies \bot$ if $f \neq g$
COAL.	$P \cup \{x = y\} \implies P\{x \mapsto y\} \cup \{x = y\}$ if $x, y \in var(P)$ and $x \neq y$
CHECK	$P \cup \{x_1 = s_1[x_2], \dots, x_n = s_n[x_1]\} \implies \bot$ if $s_i \notin \mathcal{X}$ for some $i \in [1 \dots n]$
MERGE	$P \cup \{x = s, x = t\} \implies P \cup \{x = s, s = t\}$ if $0 < \lvert s \rvert \leq \lvert t \rvert$

Figure 3.5: Rules for DAG syntactic unification

Lemma 3.3.1. *Let T be a set of terms and P be a set of equations between terms in $st(T)$ with $\sigma = mgu(P)$. We have that $st(T\sigma) \subseteq st(T)\sigma$.*

Proof. We use the rules for DAG syntactic unification given in Figure 3.5. Applying these rules on P results in a set of equations $P' = \{x_1 = t_1, \dots, x_n = t_n\}$ in DAG solved form (see [JK91]). By definition of a DAG solved form, we have that:

- $x_i \neq x_j$ for all $1 \leq i < j \leq n$,
- $x_i \notin var(t_j)$ for all $1 \leq i < j \leq n$.

Let $\sigma = \{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$. By inspection of the rules in Figure 3.5, we can show by induction on the length of the derivation from P to P' that $st(P')\sigma \subseteq st(P)\sigma$. Since $st(P) \subseteq st(T)$, we easily deduce that $st(t_i)\sigma \subseteq st(T)\sigma$ for every $1 \leq i \leq n$.

Let $u \in st(T\sigma)$, we show that there exists $t \in st(T)$ such that $u = t\sigma$. Either there exists v a subterm of T such that $u = v\sigma$, and we conclude, or there exists $x_i \in dom(\sigma)$ such that u is a subterm of $x_i\sigma$. In that case, let $i_0 = \mathsf{max}\{i \mid u \in St(x_i\sigma)\}$.

- Either $u \in st(t_{i_0})\sigma \subseteq st(T)\sigma$, and we conclude.
- Or $u \in st(x\sigma)$ for some $x \in var(t_{i_0}) \cap dom(\sigma)$. By definition of a DAG solved form, we have that $var(t_{i_0}) \cap dom(\sigma) \subseteq \{x_{i_0+1}, \dots, x_n\}$. Hence, we have that $u \in st(x_j\sigma)$ for some $j > i_0$. This yields to a contradiction.

□

Let S be a set, we denote by $\#S$ the cardinal of S. Let u be a term. We denote by $|u|_d$ the maximal depth of a variable in u. The lemma below is useful to bound the depth of variables after application of a substitution. Intuitively, the depth of variables is bounded polynomially in the size of the domain of the substitution, as well as the size of the set of terms.

Lemma 3.3.2. *Let T be a set of terms, P be a set of equations between terms in T and $\sigma = mgu(P)$. For every variable $x \in st(T)$, we have that:*

$$|x\sigma|_d \leq \#dom(\sigma) \cdot \max\{|t|_d \mid t \in T\}.$$

Proof. We use the rules for DAG syntactic unification given in Figure 3.5. Applying these rules on P results in a set of equations representing a most general unifier of P in DAG solved form (see [JK91]): $\sigma = \{x_1 = t_1, \ldots, x_n = t_n\}$. By definition of a DAG solved form, we have that:

- $x_i \neq x_j$ for all $1 \leq i < j \leq n$,
- $x_i \notin var(t_j)$ for all $1 \leq i < j \leq n$.

Hence, we have that $|x\sigma|_d < |t_1|_d + \ldots + |t_n|_d$. Furthermore, by inspection of the rules, we can see that each t_i is a subterm (modulo a non-bijective renaming of the variables) of T. For every $1 \leq i \leq n$, we have that $|t_i|_d \leq \max\{|t|_d \mid t \in T\}$. Since $n = \#dom(\sigma)$, we deduce that $|x\sigma|_d < \#dom(\sigma) \cdot \max\{|t|_d \mid t \in T\}$. □

3.3.2 Bounding variables which are not of sort loc or lists

In this section, we prove that given any solution of $\mathcal{C}$, the variables which are not of sort loc or lists can be instantiated by any fresh name, still preserving the solution.

Lemma 3.3.3. *Let $(\mathcal{C}, \mathcal{I})$ be a constraint system in solved form, Φ_1 be a formula of $\mathcal{L}_{\mathsf{route}}$, Φ_2 be a set of disequality constraints, and $G = (\mathcal{N}_{\mathsf{loc}}, E)$ be a graph. Consider σ a solution of $(\mathcal{C}, \mathcal{I}) \wedge \Phi_1 \wedge \Phi_2$ for graph G. There is a solution σ' of $(\mathcal{C}, \mathcal{I}) \wedge \Phi_1 \wedge \Phi_2$ for graph G such that:*

- *$x\sigma' = x\sigma$ for every variable x of sort* loc *or* lists*;*
- *$x\sigma' \in \mathcal{I}$ otherwise.*

Proof. Since $(\mathcal{C}, \mathcal{I})$ is a constraint system in solved form, we have that

$$\mathcal{C} = T_1 \overset{?}{\vdash} x_1 \wedge \ldots \wedge T_n \overset{?}{\vdash} x_n$$

where:

- $x_1, \ldots, x_n$ are distinct variables, and
- $var((\mathcal{C}, \mathcal{I}) \wedge \Phi_1 \wedge \Phi_2) = \{x_1, \ldots, x_n\} = rvar(\mathcal{C})$.

We show the result by induction on:

$$\mu(\sigma) = \#\{x \in rvar(\mathcal{C}) \mid x \text{ is neither of sort } \mathsf{loc} \text{ nor of sort } \mathsf{lists} \text{ and } x\sigma \notin \mathcal{I}\}.$$

Base case: $\mu(\sigma) = 0$. In such a case, since $rvar(\mathcal{C})$ contains all the variables that occur in the constraint system, we easily conclude. The substitution σ is already of the right form.

Induction step: $\mu(\sigma) > 0$. Let i_0 be the maximal index $1 \leq i_0 \leq n$ such that $x_{i_0}\sigma \notin \mathcal{I}$ and x_{i_0} is not of sort loc or lists. Let a be a name in $\mathcal{I}$ that does not occur elsewhere. Let $\sigma' = \tau \cup \{x_{i_0} \mapsto a\}$ where $\tau = \sigma|_X$ with $X = dom(\sigma) \smallsetminus \{x_{i_0}\}$. Clearly, we have that $\mu(\sigma') < \mu(\sigma)$. In order to conclude, it remains to show that σ' is a solution of $(\mathcal{C}, \mathcal{I}) \wedge \Phi_1 \wedge \Phi_2$.

1. *We show that σ' is a solution of $(\mathcal{C}, \mathcal{I})$.* For every $i < i_0$, since σ is a solution of $(\mathcal{C}, \mathcal{I})$, we have that $T_i\sigma \cup \mathcal{I} \vdash x_i\sigma$. Since x_{i_0} does not occur in this constraint, we also have that $T_i\sigma' \cup \mathcal{I} \vdash x_i\sigma'$. Since $a \in \mathcal{I}$, we have that $T_{i_0}\sigma' \cup \mathcal{I} \vdash x_{i_0}\sigma'$.

   For every $i > i_0$, according to the definition of i_0, either x_i is of sort loc or lists, or $x_i\sigma \in \mathcal{I}$. In the first case, as for every term t of sort loc or lists, $\mathcal{N}_{\mathsf{loc}} \vdash t$, we have that $\mathcal{N}_{\mathsf{loc}} \vdash x_i\sigma$. In the second case, $\mathcal{I} \vdash x_i\sigma$. Hence, in both cases, we have that $T_i\sigma' \cup \mathcal{I} \vdash x_i\sigma'$.

2. *We show that σ' is a solution of Φ_1.* All the variables appearing in Φ_1 are of type loc or lists. Hence, we have that $\Phi_1\sigma = \Phi_1\sigma'$. This allows us to conclude.

3. *Lastly, we show that σ' is a solution of Φ_2.* Let $\forall Y.u \neq v$ be a disequality constraint in Φ_2. Assume w.l.o.g. that $dom(\sigma) \cap Y = \emptyset$. Since σ is a solution of $\forall Y.u \neq v$, we know that $u\sigma$ and $v\sigma$ are not unifiable.

   Assume by contradiction that there exists a substitution θ' such that $u\sigma'\theta' = v\sigma'\theta'$ (i.e. σ' does not satisfy $\forall Y.u \neq v$). We can assume w.l.o.g. that $u\sigma'\theta'$ and $v\sigma'\theta'$ are ground terms, and $x_{i_0} \notin dom(\theta')$. In such a case, we have that:

   $$\begin{array}{l}(u\sigma')\theta' = ((u\tau)\{x_{i_0} \mapsto a\})\theta' = ((u\tau)\theta')\{x_{i_0} \mapsto a\} \\ (v\sigma')\theta' = ((v\tau)\{x_{i_0} \mapsto a\})\theta' = ((v\tau)\theta')\{x_{i_0} \mapsto a\}\end{array}$$

   Since a is fresh, we deduce that $(u\tau)\theta' = (v\tau)\theta'$. Hence, we have also that:

   $$((u\tau)\theta')\{x_{i_0} \mapsto x_{i_0}\sigma\} = ((v\tau)\theta')\{x_{i_0} \mapsto x_{i_0}\sigma\}$$

   i.e. $u\sigma\theta' = v\sigma\theta'$. This contradicts the fact that $u\sigma$ and $v\sigma$ are not unifiable.

Hence, σ' is a solution of $(\mathcal{C}, \mathcal{I}) \wedge \Phi_1 \wedge \Phi_2$. □

It remains to show that it is possible to find a solution in which lists are polynomially bounded. We need to prove two separate propositions, according to whether the network topology is fixed or not. The proofs of these propositions use the facts that, on the one hand, disequality constraints can be satisfied using fresh node names (hence the use of the set $\mathcal{I}_{names}$) and, on the other hand, the predicates of the logic $\mathcal{L}_{\mathsf{route}}$ involve only a finite number of nodes.

3.3.3 Case of a fixed topology

In case the network topology is fixed, we show that we can bound the size of an attack, where the bound depends on the size of the graph and the size of the constraints.

In particular, we show that, if there is a solution of $(\mathcal{C}, \mathcal{I}) \wedge \Phi_1 \wedge \Phi_2$ for graph G, then there exists a substitution σ such that σ is a solution of $(\mathcal{C}, \mathcal{I}) \wedge \Phi_1 \wedge \Phi_2$ for G, and variables of sort lists are instantiated by lists of length at most M where M is a bound that depends on Φ_2, Φ_1, and E.

To prove this result, we consider a *smallest* solution σ of $(\mathcal{C}, \mathcal{I}) \wedge \Phi_1 \wedge \Phi_2$ for G, and we assume that there exists a variable x_ℓ of sort lists such that $x_\ell\sigma$ is a list of length greater than M. We built a solution σ' (smaller that σ) by changing only the value of $x_\ell\sigma$ in order to reduce its length, preserving the satisfiability of our constraints. We build $x_\ell\sigma'$ by first marking the names we want to keep in $x_\ell\sigma$ getting a *marked list*, i.e. a list in which some elements are marked.

For instance, in order to ensure that a loop predicate will still be satisfied, two names are actually sufficient, whereas for a checkl predicate, three names are needed. Note that, to satisfy a positive occurrence of a route predicate, we know that the list contains at most $\#E$ names (since names in the list have to be distinct), thus we know that the variable x_ℓ is not involved in a positive occurrence of a route predicate. We also have to keep some names to take the disequality constraints into account.

Definition 3.3.1 (extracted list). *An* extracted list *from a list $l = [a_1; \ldots; a_n]$ is a list $[a_{i_1}; \ldots; a_{i_k}]$ such that $1 \leq i_1 \leq i_2 \leq \ldots \leq i_k \leq n$ with $0 \leq k \leq n$.*

Then, we consider the list extracted from $x_\ell\sigma$ by keeping the marked names plus an additional one, and we consider variations of this extracted list. Note that the length of this extracted list is bounded by the size of the graph and the size of the constraints.

Definition 3.3.2 (variation). *Let l' be a marked list in which at least one of its element is not marked. A* variation *of $l' = [a'_1; \ldots; a'_n]$ is a list $l = [a_1; \ldots; a_n]$ such that:*

- *there exists $1 \leq j_0 \leq n$ such that a'_{j_0} is not marked and a_{j_0} is a fresh name,*
- *for all $1 \leq i \leq n$ such that $i \neq j_0$, we have that $a_i = a'_i$.*

Intuitively, a variation of a list l which contains only one unmarked name is a list l' that coincides with l on all marked names, and that replaces the unmarked one by a fresh name.

Actually, instantiating x_ℓ by any variation of this extracted list allows us to ensure that our constraints are still satisfied.

We prove that we can find a solution in which lists are polynomially bounded. In the case where the network topology is fixed, the bound depends on the size of the graph, i.e. its number of edges. Let l be a list, we denote by $|l|_\ell$ the length of l.

Proposition 3.3.4. *Let $(\mathcal{C}, \mathcal{I})$ be a special constraint system in solved form, Φ_1 be a conjunction of atomic formulas of $\mathcal{L}_{\mathsf{route}}$, Φ_2 be a set of disequality constraints, and $G = (\mathcal{N}_{\mathsf{loc}}, E)$ be a graph. If there is a solution of $(\mathcal{C}, \mathcal{I}) \wedge \Phi_1 \wedge \Phi_2$ for G, then there exists a solution σ of $(\mathcal{C}, \mathcal{I}) \wedge \Phi_1 \wedge \Phi_2$ for G that is polynomially bounded in the size of Φ_1, Φ_2 and E.*

Proof. We write $\Phi_2 = \bigwedge_n \forall Y_n . u_n \neq v_n$, and

$$\Phi_1 = \bigwedge_i \pm_i \mathsf{check}(a_i, b_i) \wedge \bigwedge_j \bigwedge_k^{p_j} \pm_{j_k} \mathsf{checkl}(c_{j_k}, l_j) \wedge \bigwedge_l \pm_l \mathsf{route}(r_l) \wedge \bigwedge_h \pm_h \mathsf{loop}(p_h)$$

with $\pm \in \{+, -\}$, a_i, b_i, c_{j_k} are of sort loc, l_j, r_l, p_h are terms of sort lists, u_n, v_n are terms and Y_n are sets of variables.

In the following, we denote:

- N the maximal depth of a variable in the disequality constraints,
- k the maximal number of variables in a disequality constraint,
- C the number of constraints $\pm\mathsf{checkl}$ in Φ_1,
- L the number of constraints loop in Φ_1,
- R the number of constraints $\neg\mathsf{route}$ in Φ_1, and
- $M = \mathsf{max}(kN + 3C + L + R + 3, \#E)$.

We show that, if there is a solution of $(\mathcal{C}, \mathcal{I}) \wedge \Phi_1 \wedge \Phi_2$ for graph G, then there exists a substitution σ such that σ is a solution of $(\mathcal{C}, \mathcal{I}) \wedge \Phi_1 \wedge \Phi_2$ for G, and

- for all variables x of sort lists, $|x\sigma|_\ell \leq M$, and
- $x\sigma \in \mathcal{I} \cup \mathcal{N}_{\mathsf{loc}}$ otherwise.

First, we have that $x\sigma \in \mathcal{N}_{\mathsf{loc}}$ when x is a variable of sort loc. Moreover, thanks to Lemma 3.3.3, we can assume that $x\sigma \in \mathcal{I}$ when x is a variable that is neither of sort loc nor of type lists. Now, among these solutions, consider a smallest solution σ of $(\mathcal{C}, \mathcal{I}) \wedge \Phi_1 \wedge \Phi_2$ for G, where the size of a solution σ is given by $|\sigma| = |x_1\sigma|_\ell + \ldots + |x_n\sigma|_\ell$ where $x_1, \ldots, x_n$ are the variables of sort lists that occur in $(\mathcal{C}, \mathcal{I}) \wedge \Phi_1 \wedge \Phi_2$.

If $|x\sigma|_\ell \leq M$ for all variables x of sort lists, then we easily conclude. Otherwise, there exists a variable x_ℓ of sort lists such that the length of $x_\ell\sigma$ is greater than M. We are going to show that we can build σ' from σ, solution of $(\mathcal{C}, \mathcal{I}) \wedge \Phi_1 \wedge \Phi_2$ for G, smaller than σ. More specifically, we build σ' such that for all $x \neq x_\ell, x\sigma' = x\sigma$, and $|x_\ell\sigma'|_\ell \leq M < |x_\ell\sigma|_\ell$.

We build $x_\ell\sigma'$ by marking the names we want to keep in the list in the following manner:

$$x_\ell\sigma = \begin{array}{|c|c|c|c|c|c|} \hline a_1 & a_2 & \ldots & a_{kN} & \ldots & a_P \\ \hline \end{array}$$

We mark the first kN names in the list:

$$\begin{array}{|c|c|c|c|c|} \hline \overline{a_1} & \overline{a_2} & \ldots & \overline{a_{kN}} & \ldots \\ \hline \end{array}$$

We then mark the other names we want to keep in the list in the following way:

Case of a checkl *that occurs positively.*

If there exists c_{j_k} such that $\mathsf{checkl}(c_{j_k}, l_j)$ is a constraint that occurs positively in Φ_1, i.e. $\pm_{j_k} = +$, and $x_\ell \in var(l_j)$. Assume that $l_j = d_1 :: \ldots :: d_p :: x_\ell$. As σ is a solution for Φ_1, in particular we know that $c = c_{j_k}\sigma$ appears exactly once in $l_j\sigma$, and for any l' sublist of $l_j\sigma$,

- if $l' = a :: c :: l_1$, then $(a, c) \in E$.
- if $l' = c :: b :: l_1$, then $(b, c) \in E$.

Since c appears exactly once in $l_j\sigma$, either there exists n such that $c = d_n\sigma$, or there exists m such that $c = a_m$. In the first case and if $n = p$, we mark a_1. In the second case, we mark a_m, a_{m-1}(if $m > 1$) and a_{m+1}(if $m < P$). Any variation of a list extracted from $x_\ell\sigma$ containing at least the marked names plus another one satisfies the checkl condition for graph G.

a_1	$\dots$	$\overline{a_{m-1}}$	$\overline{a_m}$	$\overline{a_{m+1}}$	$\dots$	a_P

Case of a checkl *that occurs negatively.*

If there exists c_{j_k} such that $\mathsf{checkl}(c_{j_k}, l_j)$ is a constraint that occurs negatively in Φ_1, i.e. $\pm_{j_k} = -$, and $x_\ell \in var(l_j)$. Assume that $l_j = b_1 :: \dots :: b_p :: x_\ell$. As σ is a solution for Φ_1, we can have three different cases depending on $c = c_{j_k}\sigma$:

- c does not appear in $l_j\sigma$: for every n, m, $b_n\sigma \neq c$ and $a_m \neq c$. In that case, we mark nothing.
- c appears at least twice in $l_j\sigma$. In that case, we choose two occurrences of c and we mark them when they appear in $x_\ell\sigma$.

a_1	$\dots$	$\overline{c}$	$\dots$	$\overline{c}$	$\dots$	a_P

- c appears once in $l_j\sigma$, but one of his neighbors in the list is not a neighbor of it in the graph. For example, $c = a_i$ and $(a_i, a_{i+1}) \notin E$. We mark c and this false neighbor when they appear in $x_\ell\sigma$.

a_1	$\dots$	$\overline{a_i}$	$\overline{a_{i+1}}$	$\dots$	a_M

Any variation of a list extracted from $x_\ell\sigma$ containing at least the marked names plus another one satisfies the $\neg$checkl condition for graph G.

Case of a loop *that occurs positively.*

If there exists h such that $\mathsf{loop}(p_h)$ is a constraint that occurs positively in Φ_1, i.e. $\pm_h = +$, and $x_\ell \in var(p_h)$. Assume $p_h = b_1 :: \dots :: b_p :: x_\ell$. Then there exists a name c repeated in $p_h\sigma$. We mark two occurrences of such a c, when they appear in $x_\ell\sigma$.

a_1	$\dots$	$\overline{c}$	$\dots$	$\overline{c}$	$\dots$	a_P

Any variation of a list extracted from $x_\ell\sigma$ containing at least the marked names plus another one satisfies the loop condition for graph G. Indeed, the condition does not depend on the graph.

Case of a loop *that occurs negatively.*

If there exists h such that $\mathsf{loop}(p_h)$ occurs negatively in Φ_1, i.e. $\pm_h = -$, and $x_\ell \in var(p_h)$. Assume that $p_h = b_1 :: \dots :: b_p :: x_\ell$. Removing nodes from the list preserves this condition, so any extracted list of $x_\ell\sigma$ satisfies the $\neg\mathsf{loop}$ condition. Moreover, as a variation of a list is built with a fresh constant, any variation of a list extracted from $x_\ell\sigma$ satisfies the condition.

Case of a route *that occurs negatively.*

If there exists r_l such that $\mathsf{route}(r_l)$ occurs negatively in Φ_1, i.e. $\pm_l = -$, and $x_\ell \in var(r_l)$. Assume that $r_l = b_1 :: \ldots :: b_p :: x_\ell$. As σ is a solution for Φ_1, we can have two different cases:

- There exists a name c repeated in $r_l\sigma$. Then we mark two occurrences of such a c, when they appear in $x_\ell\sigma$.
- There exists a sublist l of $r_l\sigma$ such that $l = c :: d :: l_1$ and $(c, d) \notin E$. We mark c and d if they appear in $x_\ell\sigma$.

$$\begin{array}{|c|c|c|c|c|c|}\hline a_1 & \ldots & \bar{c} & \bar{d} & \ldots & a_P \\ \hline\end{array}$$

Any variation of a list extracted from $x_\ell\sigma$ containing at least the marked names plus another one satisfies the $\neg\mathsf{route}$ condition for G.

Case of a route *that occurs positively.*

If there exists r_l such that $\mathsf{route}(r_l)$ occurs positively in Φ_1, i.e. $\pm_l = +$, and $x_\ell \in var(r_l)$. Assume that $r_l = b_1 :: \ldots :: b_p :: x_\ell$. Write $r_l\sigma = c_1 :: \ldots :: c_n$. As σ is a solution for Φ_1 in G, for every $0 < i < n$, $(c_i, c_{i+1}) \in E$ and for every $i \neq j$, $c_i \neq c_j$. Consequently, $|r_l\sigma|_\ell \leq \#E$, and as $|x_\ell\sigma|_\ell \leq |r_l\sigma|_\ell$, we have that $|x_\ell\sigma|_\ell \leq \#E$. But our hypothesis tells us that $|x_\ell\sigma|_\ell > M \geq \#E$. So there is no positive route condition on x_ℓ.

We count the number of marked names. We have marked the first kN names in the list. For each constraint $\pm\mathsf{checkl}$, we mark at most 3 names in the list. Suppose there are several constraints $\neg\mathsf{route}(l)$ with x_ℓ sublist of l. Either $\neg\mathsf{route}(x_\ell\sigma)$ holds, and we can mark two names in $x_\ell\sigma$ which will make all the $\neg\mathsf{route}$ constraints true; or the constraint is satisfied by marking one name for each constraint. Thus, we need only mark $\mathsf{max}(R, 2)$ names when $R \geq 1$ and 0 otherwise. Thus, in any case, it is sufficient to mark $R + 1$ names in $x_\ell\sigma$. Similarly, it is sufficient to mark $L + 1$ names in $x_\ell\sigma$ to satisfy the loop constraints. The number of names marked in the list is at most

$$kN + 3C + (R + 1) + (L + 1) \leq M.$$

Consider l_1 extracted from $x_\ell\sigma$ by keeping only the marked names in $x_\ell\sigma$ and the first unmarked name. Such an unmarked name exists, because $|x_\ell\sigma|_\ell \geq M$. Let l_2 be the variation of l_1 replacing the first unmarked name with a fresh constant a_ℓ. For each condition considered above, l_2 satisfies it, as it is a variation of a list extracted from $x_\ell\sigma$ containing the marked names.

Let σ_0 be the substitution such that $x\sigma_0 = x\sigma$ for every $x \in dom(\sigma) \smallsetminus \{x_\ell\}$, and $x\sigma = x$ otherwise. Let $\sigma' = \sigma_0 \cup \{x_\ell \mapsto l_2\}$. By hypothesis, σ is a solution of Φ_1 for G, so by construction, σ' is a solution of Φ_1 for G. Now, it remains for us to show that σ' is a solution of $(\mathcal{C}, \mathcal{I})$ and Φ_2.

Deduction constraints. Consider a deduction constraint $T_i \vdash x_i$ in $\mathcal{C}$. Either x_i is of sort loc or lists, which means that $\mathcal{N}_{\mathsf{loc}} \vdash x_i\sigma'$, thus $T_i\sigma' \cup \mathcal{I} \subseteq T_0 \cup \mathcal{I} \vdash x_i\sigma'$. Or x_i is not of sort loc or lists, so in particular $x_i \in dom(\sigma) \smallsetminus x_\ell$, and $x_i\sigma' = x_i\sigma \in \mathcal{I} \cup \mathcal{N}_{\mathsf{loc}}$, so again $T_i\sigma' \cup \mathcal{I} \subseteq T_0 \cup \mathcal{I} \vdash x_i\sigma'$. Hence, in both cases, we have that $T_i\sigma' \cup \mathcal{I} \vdash x_i\sigma'$. Consequently, σ' is a solution of $(\mathcal{C}, \mathcal{I})$.

Disequality constraints. Consider a disequality constraint $\forall Y.u \neq v \in \Phi_2$. We assume w.l.o.g. that $dom(\sigma) \cap Y = \emptyset$. We have to show that $u\sigma'$ and $v\sigma'$ are not unifiable. We distinguish two cases. Either $u\sigma_0$ and $v\sigma_0$ are not unifiable, but in such a case, we easily deduce that $u\sigma'$ and $v\sigma'$ are not unifiable too. This allows us to conclude. Otherwise, let $\mu = mgu(u\sigma_0, v\sigma_0)$.

If $dom(\mu) \subseteq Y$, let $\tau = \{x_\ell \mapsto x_\ell\sigma\} \circ \mu$. We have that:

$$\begin{aligned}(u\sigma)\tau = ((u\overline{\sigma})\{x_\ell \mapsto x_\ell\sigma\})\tau = (u\overline{\sigma}\mu)\{x_\ell \mapsto x_\ell\sigma\}\\ (v\sigma)\tau = ((v\overline{\sigma})\{x_\ell \mapsto x_\ell\sigma\})\tau = (v\overline{\sigma}\mu)\{x_\ell \mapsto x_\ell\sigma\}.\end{aligned}$$

Hence, we deduce that $u\sigma$ and $v\sigma$ are unifiable, and we obtain a contradiction since σ satisfies the constraint $\forall Y.u \neq v$. Hence, this case is impossible.

Otherwise, there exists a term t such that $\mu(x_\ell) = t$, and $var(t) \subseteq Y$. We apply Lemma 3.3.2 to the set $T = \{u\sigma_0, v\sigma_0\}$, and the set of equations $P = \{u\sigma_0 = v\sigma_0\}$. We have that $\mu = mgu(P)$. Since $\overline{\sigma}$ is ground, we get that:

$$\begin{array}{rcl}|t|_d & \leq & \#dom(\mu).\mathsf{max}(|u\overline{\sigma}|_d, |v\overline{\sigma}|_d)\\ & \leq & \#dom(\mu).\mathsf{max}(|u|_d, |v|_d)\\ & \leq & kN\end{array}$$

We reason by case over t:

- If t is not of sort lists, as σ' is well-sorted, $u\sigma'$ and $v\sigma'$ are not unifiable.

- Suppose $t = [a_1; \dots; a_n]$, with $a_1, \dots, a_n$ terms of sort loc. We write $t = t_1@t_2$ with t_2 ground term of maximal size, where @ denotes the concatenation of lists. We have shown that $|t_1|_d = |t|_d \leq kN$.

  We know that $x_\ell\sigma' = [b_1; \dots; b_p]$ and there exists $k' > kN$ such that $b_{k'} = a_\ell$ and a_ℓ is a name of $\mathcal{I}$ which does not appear anywhere else in the constraints. Consequently, $a_{k'} \neq a_\ell$, and so $x_\ell\sigma' \neq t\theta$ for any substitution θ.

  Now, assume by contradiction that $u\sigma'$ and $v\sigma'$ are unifiable. This means that there exists τ such that $(u\sigma')\tau = (v\sigma')\tau$. Hence, we have that $\tau \circ \{x_\ell \mapsto x_\ell\sigma'\}$ is an unifier of $u\sigma_0$ and $v\sigma_0$. By hypothesis, we have that $\mu = mgu(u\sigma_0, v\sigma_0)$. Hence, we deduce that there exists θ' such that $\tau \circ \{x_\ell \mapsto x_\ell\sigma'\} = \theta' \circ \mu$. We have that:

  - $\tau \circ \{x_\ell \mapsto x_\ell\sigma'\}(x_\ell) = x_\ell\sigma'$, and
  - $\theta' \circ \mu(x_\ell) = t\theta'$.

  This leads to a contradiction.

- Suppose $t = a_1 :: \dots :: a_n :: y_\ell$, with $y_\ell \in Y$ variable of sort lists. We know that $|t|_d \leq kN$, thus we must have $n < kN$. We reason by contradiction. Assume that there exists θ' such that $(u\sigma')\theta' = (v\sigma')\theta'$. In the remaining of the proof, we show that $u\sigma$ and $v\sigma$ are unifiable.

By hypothesis, we have that $\theta' \circ \{x_\ell \mapsto x_\ell\sigma'\}$ is an unifier of $u\sigma_0$ and $v\sigma_0$. Since $\mu = mgu(u\sigma_0, v\sigma_0)$, we deduce that there exists ρ' such that:

$$\rho' \circ \mu \;=\; \theta' \circ \{x_\ell \mapsto x_\ell\sigma'\}.$$

We have that $x_\ell\sigma' = (x_\ell\mu)\rho' = t\rho'$. By hypothesis, we know that the size of $x_\ell\sigma$ is greater than $M \geq kN > n$. Let l_t be the list obtaining from $x_\ell\sigma$ by removing its n first elements. Let ρ_0 be a substitution such that $x\rho_0 = x\rho'$ for every $x \in dom(\rho) \smallsetminus \{y_\ell\}$, and $y\rho_0 = y$ otherwise. Let $\rho = \rho_0 \circ \{y_\ell \mapsto l_t\}$. In order to conclude, it remains to show that $\rho \circ \mu$ is an unifier of $u\sigma$ and $v\sigma$.

We have that $x_\ell\sigma' = (x_\ell\mu)\rho' = t\rho' = a_i\rho' :: \ldots a_n\rho' :: y_\ell\rho'$. Moreover, we know that $x_\ell\sigma$ and $x_\ell\sigma'$ have the same first kN elements by construction, and $n < kN$. Relying on this fact to establish the last equality, we have that:

$$\begin{array}{rcl} (x_\ell\mu)\rho & = & t\rho \\ & = & (a_1 :: \ldots :: a_n :: y_\ell)\rho \\ & = & a_1\rho :: \ldots :: a_n\rho :: l_t \\ & = & a_1\rho' :: \ldots :: a_n\rho' :: l_t \\ & = & x_\ell\sigma. \end{array}$$

Hence, we have that $((u\sigma)\mu)\rho = ((u\sigma_0)\mu)\rho$, and $((v\sigma)\mu)\rho = ((v\sigma_0)\mu)\rho$. We easily conclude that $u\sigma$ and $v\sigma$ are unifiable since we know that $(u\sigma_0)\mu = (v\sigma_0)\mu$.

In all possible cases, σ' satisfies the disequality constraint.

As a conclusion, σ' is a solution of $(\mathcal{C}, \mathcal{I}) \wedge \Phi_1 \wedge \Phi_2$, smaller than σ, which leads to a contradiction. ☐

3.3.4 Case of an *a priori* unknown topology

In the case where the network topology is not fixed, we show that we can bound the size of an attack.

The method for bounding the lists follows the same lines as the proof of Proposition 3.3.4. However, we can not consider the size of the graph to bound the size of the lists. This was used in the proof of Proposition 3.3.4 to deal with the case of route that occur positively in the formula. Here, we rely on the fact that we can change the graph to solve this problem, and we consider ubiquitous graphs. More precisely, we introduce the notion of ubiquitous nodes, that is, nodes connected to every other nodes in the graph. We associate to a graph G a ubiquitous graph where all the nodes that are not already part of an edge in G become ubiquitous.

Definition 3.3.3 (ubiquitous graph). *Let $G = (\mathcal{N}_{\mathsf{loc}}, E)$ be a finite graph (i.e. such that E is finite). Consider the sets of nodes $V = \{n \mid \exists n'$ such that $(n, n') \in E\}$, and $V_{ubi} \subseteq \mathcal{N}_{\mathsf{loc}} \smallsetminus V$. The graph $(\mathcal{N}_{\mathsf{loc}}, E \cup E_{ubi})$ where $E_{ubi} = \{(a, b) \mid a \in V \cup V_{ubi},\, b \in V_{ubi}\}$ is called the* ubiquitous graph associated to G and V_{ubi}.

Moreover, we consider ubiquitous variations instead of variations. Ubiquitous variations replace the unmarked names in a list by names of ubiquitous nodes. This is once again needed to satisfy a formula route that occurs positively.

Definition 3.3.4 (ubiquitous variation). *Let l' be a marked list and n be the number of unmarked elements in l'. Let V_{ubi} be a set of nodes such that $\#V_{ubi} > n$ and names in V_{ubi} do not occur in l'. A* ubiquitous variation *according to V_{ubi} of $l' = [a'_1; \ldots; a'_n]$ is a list $l = [a_1; \ldots; a_n]$ such that:*

- *for all $1 \leq i \leq n$ such that a'_i is not marked, $a_i \in V_{ubi}$,*
- *for all $1 \leq i \leq n$ such that a'_i is marked, $a_i = a'_i$.*

Moreover we require that the ubiquitous nodes of l are all distinct.

First, we show that changing the graph does not change the solution, provided we do not change the links between the nodes taking part in the protocol.

Lemma 3.3.6. *Let $G = (\mathcal{N}_{\mathsf{loc}}, E)$ be a graph, $(\mathcal{C}, \mathcal{I})$ be a special constraint system, Φ_1 be a formula of $\mathcal{L}_{\mathsf{route}}$, and Φ_2 be a set of disequality constraints. Let σ be a solution of $(\mathcal{C}, \mathcal{I}) \wedge \Phi_1 \wedge \Phi_2$ for G and $\mathcal{N}'_{\mathsf{loc}} = \mathcal{N}_{\mathsf{loc}} \cap names(\mathcal{C}, \Phi_1, \Phi_2, \sigma)$. Let $G' = (\mathcal{N}_{\mathsf{loc}}, E')$ be a graph that coincides with G on $\mathcal{N}'_{\mathsf{loc}}$, i.e. such that $E = \{(n_1, n_2) \mid (n_1, n_2) \in E'$ and $n_1, n_2 \in \mathcal{N}'_{\mathsf{loc}}\}$. Then σ is a solution of $(\mathcal{C}, \mathcal{I}) \wedge \Phi_1 \wedge \Phi_2$ for G'.*

Proof. We show that σ satisfies each constraint in $(\mathcal{C}, \mathcal{I}) \wedge \Phi_1 \wedge \Phi_2$ when the underlying graph is G'. First, not that σ trivially satisfies the deduction constraints, the disequality constraints and the loop constraints.

In order to conclude, we have to check that this result also holds for the remaining constraints in Φ_1.

- $[\![\mathsf{check}(a\sigma, b\sigma)]\!]_G = 1$ if, and only if, $(a\sigma, b\sigma) \in E$. We have that $[\![\mathsf{check}(a\sigma, b\sigma)]\!]_G = 1$ if, and only if, $[\![\mathsf{check}(a\sigma, b\sigma)]\!]_{G'} = 1$.
- $[\![\mathsf{checkl}(c\sigma, l\sigma)]\!]_G = 1$ if, and only if, $l\sigma$ is of sort lists, $c\sigma$ appears exactly once in $l\sigma$, and for any l' sub-list of $l\sigma$,
  - if $l' = a :: c\sigma :: l_1$, then $(a, c\sigma) \in E$.
  - if $l' = c\sigma :: b :: l_1$, then $(b, c\sigma) \in E$.

  As in the previous case, we easily conclude that $[\![\mathsf{checkl}(c\sigma, l\sigma)]\!]_G = 1$ if, and only if, $[\![\mathsf{checkl}(c\sigma, l\sigma)]\!]_{G'} = 1$.
- $[\![\mathsf{route}(l\sigma)]\!]_G = 1$ if, and only if, $l\sigma$ is of sort lists, $l\sigma = [a_1; \ldots; a_n]$, for every $1 \leq i < n$, $(a_i, a_{i+1}) \in E$, and for every $1 \leq i, j \leq n, i \neq j$ implies that $a_i \neq a_j$. As in the previous case, $(a_i, a_{i+1}) \in E$ if, and only if, $(a_i, a_{i+1}) \in E'$. Hence, $[\![\mathsf{route}(l\sigma)]\!]_G = 1$ if, and only if, $[\![\mathsf{route}(l\sigma)]\!]_{G'} = 1$.

Hence, σ is a solution of $(\mathcal{C}, \mathcal{I}) \wedge \Phi_1 \wedge \Phi_2$ for G'. □

We prove that we can find a solution in which lists are polynomially bounded. In the case where the network topology is unknown, the bound depends on the size of the formulas.

Proposition 3.3.5. *Let $(\mathcal{C}, \mathcal{I})$ be a special constraint system in solved form, Φ_1 be a conjunction of atomic formulas of $\mathcal{L}_{\mathsf{route}}$, Φ_2 be a set of disequality constraints. If there is a solution of $(\mathcal{C}, \mathcal{I}) \wedge \Phi_1 \wedge \Phi_2$ for the graph $G = (\mathcal{N}_{\mathsf{loc}}, E)$, then there exists a graph $G' = (\mathcal{N}_{\mathsf{loc}}, E')$ and a substitution σ such that σ is a solution of $(\mathcal{C}, \mathcal{I}) \wedge \Phi_1 \wedge \Phi_2$ for G', and σ is polynomially bounded in the size of Φ_1 and Φ_2. Moreover, we have that G' coincides with G on $V = \{n \mid \exists n'$ such that $(n, n') \in E\}$, i.e. $E = \{(n_1, n_2) \in E' \mid n_1, n_2 \in V\}$.*

Proof. We adapt the proof of Proposition 3.3.4 by showing that there exists a solution σ such that for every variable x of sort lists, we have that $|x\sigma|_\ell \leq M = 2\times(kN+3C+L+R+2)$ where k, N, C, L, and R are defined as in Proposition 3.3.4.

Let σ be a solution of $(\mathcal{C},\mathcal{I})\wedge\Phi_1\wedge\Phi_2$ for graph G and assume that there exists a variable x_ℓ of sort lists such that $|x_\ell\sigma|_\ell > M$. Let V_{ubi} be a set of $M/2$ fresh nodes, i.e. names in $\mathcal{N}_{\mathsf{loc}}$ that do not occur in $\mathcal{C}$, Φ_1, Φ_2, . Consider G' the ubiquitous graph associated to G and V_{ubi}. We show that we can build σ', a solution of $(\mathcal{C},\mathcal{I})\wedge\Phi_1\wedge\Phi_2$ for graph G', such that for $x \neq x_\ell$, $x\sigma' = x\sigma$, and $|x_\ell\sigma'|_\ell \leq M$.

We build σ' in a similar way as in the previous proof. We mark $x_\ell\sigma$ as in the previous proof. The number of names marked in the list is at most:

$$kN + 3C + (R+1) + (L+1) \leq M/2.$$

Consider l_1 extracted from $x_\ell\sigma$ by leaving exactly one unmarked name between sequences of marked names. Note that, we have no more than $M/2$ unmarked names in l_1. Let l_2 be the ubiquitous variation of l_1 according to V_{ubi}. The fact that we consider a ubiquitous variation allows one to satisfy the constraint route that occurs positively. Note that, we have no more than $M/2$ ubiquitous names in l_2, so $|l_2|_\ell \leq M$.

Let σ_0 be the substitution such that $x\sigma_0 = x\sigma$ for every $x \in dom(\sigma) \smallsetminus \{x_\ell\}$, and $x\sigma = x$ otherwise. Let $\sigma' = \sigma_0 \cup \{x_\ell \mapsto l_2\}$. By construction, we have that the substitution σ' satisfies Φ_1. We show that σ' is a solution of $(\mathcal{C},\mathcal{I})$ and Φ_2 for G' as in Proposition 3.3.4. □

3.4 Decidability result

We are now ready to state our two main decidability results.

Simple properties like secrecy are undecidable when considering an unbounded number of role executions, even for classical protocols [DLMS99]. Since our class of processes encompasses classical protocols, the existence of an attack is also undecidable. In what follows, we thus consider a finite number of sessions, i.e. processes without replication.

In most existing frameworks, the intruder is given as initial knowledge a finite number of messages (e.g. some of the secret keys or messages learned in previous executions). However, in the context of routing protocols, it is important to give an a priori unbounded number of node names to the attacker that he can use as its will, in particular for possibly passing some disequality constraints and for creating false routes.

Definition 3.4.1. *We say that a process is* finite *if it does not contain the replication operator. A concrete configuration $K = (\mathcal{P}[\_];\mathcal{S};\mathcal{I})$ is said* initial *if K is ground, $\mathcal{P}$ is finite, $\mathcal{S}$ is a finite set of terms and $\mathcal{I} = \mathcal{I}_{names} \cup \mathcal{I}_{\mathsf{terms}}$ where $\mathcal{I}_{\mathsf{terms}}$ is a finite set of terms and $\mathcal{I}_{names}$ is an infinite set of names. Moreover, $\mathcal{N}_{\mathsf{loc}} \subseteq \mathcal{I}_{names} \cup \mathcal{I}_{\mathsf{terms}}$ (the intruder is given all the node names in addition to its usual initial knowledge) .*

The intruder is thus given an infinity of node names in addition to its usual initial knowledge. In practice, this enables him to generate any IP address that he chooses.

We show that accessibility properties are decidable for finite processes of our process algebra, which models secured routing protocols, for a bounded number of sessions. We actually provide two decision procedures, according to whether the network is *a priori* given or not. In the case where the network topology is not fixed in advance, our procedure enables us to automatically discover whether there exists a (worst-case) topology that would yield an attack.

Note that Theorem 3.4.1(unknown topology) does not imply Theorem 3.4.2(fixed topology) and reciprocally. Indeed, in Theorem 3.4.2, the whole topology is fixed, including in particular the location of the intruder nodes. Theorems 3.4.1 and 3.4.2 ensure in particular that we can decide whether a routing protocol like SRP can guarantee that any route accepted by the source is indeed a route (a path) in the network (which can be fixed by the user or discovered by the procedure). The NP-hardness of the existence of an attack comes from the NP-hardness of the existence of a solution for deduction constraint systems [RT01].

3.4.1 Case of an unknown topology

We show that in the case of an *a priori* unknown topology, deciding whether there is an attack is decidable, providing an NPTIME complexity bound.

Theorem 3.4.1. *Let $K = (\mathcal{P}[\_];\mathcal{S};\mathcal{I})$ be an initial concrete configuration with a hole, $\mathcal{M} \subseteq \mathcal{N}_{\mathsf{loc}}$ be a finite set of nodes, and $\Phi \in \mathcal{L}_{\mathsf{route}}$ be a formula. Deciding whether there exists a graph $G = (\mathcal{N}_{\mathsf{loc}}, E)$ such that there is an $\mathcal{M}$-attack on K and Φ for the topology G is NP-complete.*

We provide first a very general sketch of the proof:

- We first use the symbolic semantics (Section 3.2) based on constraint systems, more amenable to automation. We have already shown its correctness and completeness w.r.t. the concrete semantics.
- We transform the constraint systems obtained through the execution of a routing protocol into *solved* constraint systems (as demonstrated in Chapter 2) .
- We show how to bound the size of a minimal attack on a solved constraint system (Section 3.3).

Let us now detail the decision procedure.

Let $K_s = (\mathcal{P}[\mathsf{if}\ \Phi\ \mathsf{then}\ \mathsf{out}(\mathsf{error})\ \mathsf{else}\ 0];\mathcal{S};\mathcal{I};\emptyset;\emptyset)$. K_s is a ground symbolic configuration whose concretization is $(\mathcal{P}[\mathsf{if}\ \Phi\ \mathsf{then}\ \mathsf{out}(\mathsf{error})\ \mathsf{else}\ 0];\mathcal{S};\mathcal{I})$. Let V_K be the set of names of sort loc that occur in $\mathcal{P}$ and $\mathcal{M}$. Our decision procedure works as follows:

Step 1 We partially guess the graph $G = (\mathcal{N}_{\mathsf{loc}}, E)$. Actually, we guess whether $(n_1, n_2) \in E$ for every $n_1, n_2 \in V_K$.
Let $G_K = (\mathcal{N}_{\mathsf{loc}}, E_K)$ where $E_K = \{(n_1, n_2) \mid (n_1, n_2) \in E \text{ and } n_1, n_2 \in V_K\}$.

Step 2 We guess a path of execution of the symbolic transition rules w.r.t. the graph G_K.

$$K_s \rightarrow^{s*}_{G_K,\mathcal{M}} (\lfloor \mathsf{out}(u) \rfloor_n \cup \mathcal{P}';\mathcal{S}';\mathcal{I}';\mathcal{C};\Psi).$$

Step 3 Let $\mathcal{I}' = \mathcal{I}_{names} \uplus \mathcal{I}_{\mathsf{terms}}$ such that $(\mathcal{C}, \mathcal{I}_{names})$ is a constraint system. Let $\sigma = mgu(u, \mathsf{error})$ and $\mathcal{C}\sigma = \mathcal{D}$ and $\Psi\sigma = \Phi'_1$ where $\mathcal{D}$ is a finite set of deduction constraints and Φ'_1 contains disequality constraints and formulas of $\mathcal{L}_{\mathsf{route}}$.

Step 4 We guess a sequence of transformation rules from $(\mathcal{D}, \mathcal{I}_{names})$ to $(\mathcal{D}', \mathcal{I}_{names})$ where $(\mathcal{D}', \mathcal{I}_{names})$ is a constraint system in solved form. We have that:

$$(\mathcal{D}, \mathcal{I}_{names}) \rightsquigarrow^*_{\sigma'} (\mathcal{D}', \mathcal{I}_{names}) \text{ with } (\mathcal{D}', \mathcal{I}_{names}) \text{ in solved form.}$$

Step 5 We compute the conjunctive normal form of the formula Φ'_1. Hence, Φ'_1 is equivalent to

$$\bigwedge_k \phi^k_1 \vee \cdots \vee \phi^k_{i_k}.$$

We choose non-deterministically $\phi^k_{\alpha_k}$ for every k. Let $\Phi_2 = \bigwedge_k \phi^k_{\alpha_k}$.

Step 6 Let S be the DAG size of $\mathcal{P}$, $\mathcal{S}$, Φ, $\mathcal{M}$, and $\mathcal{I}_f$. Let $\mathcal{I}_0$ be a finite subset of $\mathcal{I}_{names}$ of size $2\mathsf{S}^2 \times (\mathsf{S}^4 + 5\mathsf{S}^2 + 2)$. Guess the values of variables which are not of sort lists in $\mathcal{I}_0 \cup names(\mathcal{P}, \mathcal{S}, \Phi, \mathcal{M}, \mathcal{I}_f)$. Guess the values of variables of sort lists among lists of nodes in $\mathcal{I}_0 \cup names(\mathcal{P}, \mathcal{S}, \Phi, \mathcal{M}, \mathcal{I}_f)$ of length at most $2 \times (\mathsf{S}^4 + 5\mathsf{S}^2 + 2)$. This gives us a substitution σ and we guess a graph $G = (\mathcal{N}_{\mathsf{loc}}, E)$ such that $E \subseteq \{(n_1, n_2) \mid n_1, n_2 \in \mathcal{I}_0 \cup names(\mathcal{P}, \mathcal{S}, \Phi, \mathcal{M}, \mathcal{I}_f)\}$ and that coincides with G_K on V_K, i.e:

$$E_K = \{(n_1, n_2) \in E \mid n_1, n_2 \in V_K\}.$$

Lastly, we check whether σ is a solution of $(\mathcal{D}', \mathcal{I}_{names}) \wedge \Phi_2$ for the graph G.

Proof. We now explain each step of our algorithm.

Step 1. We have that $\#V_K < \#names(\mathcal{P}, \mathcal{M})$. Hence, we can guess G_K whose size is polynomially bounded.

Step 2. For every graph $G' = (\mathcal{N}_{\mathsf{loc}}, E')$ with $E_K = \{(n_1, n_2) \in E' \mid n_1, n_2 \in V_K\}$, we have that:

$$(\mathcal{P}; \mathcal{S}; \mathcal{I}; \mathcal{C}; \Psi) \rightarrow^{s*}_{G_K, \mathcal{M}} (\mathcal{P}'; \mathcal{S}'; \mathcal{I}'; \mathcal{C}'; \Psi') \text{ iff } (\mathcal{P}; \mathcal{S}; \mathcal{I}; \mathcal{C}; \Psi) \rightarrow^{s*}_{G', \mathcal{M}} (\mathcal{P}'; \mathcal{S}'; \mathcal{I}'; \mathcal{C}'; \Psi').$$

So we can guess the transitions knowing only E_K. Now, thanks to Theorem 3.2.4 we deduce that there is an $\mathcal{M}$-attack on K and Φ for graph G if, and only if, there is a derivation

$$(\mathcal{P}[\mathsf{if}\ \Phi\ \mathsf{then}\ \mathsf{out}(\mathsf{error})\ \mathsf{else}\ 0]; \mathcal{S}; \mathcal{I}; \emptyset; \emptyset) \ \rightarrow^{s*}_{G_K, \mathcal{M}} \ (\lfloor \mathsf{out}(u) \rfloor_n \cup \mathcal{P}_s; \mathcal{S}_s; \mathcal{I}_s; \mathcal{C}; \Psi)$$

with $\sigma = mgu(u, \mathsf{error})$ and the constraint system $(\mathcal{C}\sigma, \mathcal{I}_{names})$ together with $\Psi\sigma$ has a solution for graph G.

Actually, we can guess such a path. Indeed, the number of derivations starting from configuration K_s is bounded. Actually, the length of possible paths is bounded by the size of the protocol: as there is no replication in the initial configuration, each transition leads to a smaller process. Moreover, the number of configurations reachable with one symbolic transition is bounded as well: we can first guess which process is going to evolve and which is the corresponding transition. There is only one possible resulting configuration once this is chosen, except for the communication transition, where we also have to guess which neighbors will receive the message, and for the read transition, where we have to choose which term to read.

When a transition $K_s \rightarrow^s{}_{G_K,\mathcal{M}} K'_s$ occurs and K_s is finite, in particular it does not contain the replication operator, we have to control the size of K'_s. If the rule considered is not the $\textsc{Comm}_s$ rule, then the number of subterms in K'_s is smaller or equal to the number of subterms in K_s. Indeed, it is straightforward to see when considering rules $\textsc{In}_s$, $\textsc{Store}_s$, $\textsc{Read-Else}_s$, $\textsc{If-Then}_s$, $\textsc{If-Else}_s$, $\textsc{Par}_s$ and $\textsc{New}_s$. When applying rule $\textsc{Read-Then}_s$, a substitution is applied to all the terms in K'_s, but as the substitution is a most general unifier of terms in K_s, the number of subterms does not increase. But when considering a $\textsc{Comm}_s$ rule, new terms can be produced by renaming some variables. However, the number of $\textsc{Comm}_s$ steps is bounded by the number of terms of the form $\mathsf{out}(t)$ in the protocol, and each step can produce in the worst case a number of terms polynomial in the size of the configuration, as they are terms that appear in formulas. Consequently, at the end of the derivation, the number of subterms considered is polynomial in the number of subterms at the beginning of the derivation.

Step 3. Straightforward.

Step 4. We apply Theorem 2.3.1. Thus, there exists a solution θ of $(\mathcal{D}, \mathcal{I}_{names})$ and Φ_1 for graph G if, and only if, there exists a constraint system $(\mathcal{D}', \mathcal{I}_{names})$ in solved form and some substitutions σ', and θ' such that $\theta = \theta' \circ \sigma'$, $(\mathcal{D}, \mathcal{I}_{names}) \rightsquigarrow^*_{\sigma'} (\mathcal{D}', \mathcal{I}_{names})$ and θ' is a solution for $(\mathcal{D}', \mathcal{I}_{names})$ and $\Phi_1\sigma'$ for graph G.

Step 5. This step is straightforward. The formula $\Phi_1\sigma'$ contains disequality constraints and formulas of $\mathcal{L}_{\mathsf{route}}$. Consequently, $\Phi_2 = \bigwedge_k \phi^k_{\alpha_k}$, obtained from $\Phi_1\sigma'$, can be written:

$$\begin{aligned}\Phi_2 = \bigwedge_i \forall Y_i . u_i \neq v_i \ \wedge\ \bigwedge_j \pm_j\, \mathsf{check}(a_j, b_j)\ \wedge \\ \bigwedge_k \bigwedge_i \pm_{i_k}\, \mathsf{checkl}(c_{i_k}, l_k)\ \wedge\ \bigwedge_h \pm_h\, \mathsf{loop}(p_h)\ \wedge\ \bigwedge_l \pm_l\, \mathsf{route}(r_l)\end{aligned}$$

Finally, we are left to decide whether there exists a solution to a solved special constraint system $(\mathcal{D}', \mathcal{I}_{names})$ and a formula Φ_2 as described above.

Step 6. First, we show that for any term $t \in st(\mathcal{D}', \Phi_2)$, there exists t' in $st(\mathcal{D}, \Phi'_1)$ such that $t = (t'\sigma)\sigma'$. Thanks to Theorem 2.3.1, we have that

$$st(\mathcal{D}') \subseteq st(\mathcal{D}\sigma') \subseteq st(\mathcal{D})\sigma'.$$

Moreover, we have that

$$st(\Phi_2) \subseteq st(\Phi'_1\sigma') \subseteq st(\Phi'_1)\sigma' \cup \bigcup_{x \in var(\mathcal{D})} st(x\sigma') \subseteq st(\Phi'_1)\sigma' \cup st(\mathcal{D})\sigma'.$$

The last inclusion is a direct consequence of the inclusion $st(\mathcal{D}\sigma') \subseteq st(\mathcal{D})\sigma'$. Hence, we have that: $st(\mathcal{D}', \Phi_2) \subseteq st(\Phi_1)\sigma' \cup st(\mathcal{D})\sigma' \subseteq st(\mathcal{C}\sigma)\sigma'$. By relying on Lemma 3.3.1, we obtain that $st(\mathcal{C}\sigma) \subseteq st(\mathcal{C})\sigma$. Since $names(\mathcal{D}', \Phi_2) \cap \mathcal{I}_0 = \emptyset$, we deduce that

$$\begin{aligned} st(\mathcal{D}', \Phi_2) &\subseteq st(\mathcal{C})\sigma\sigma' \smallsetminus \mathcal{I}_0 \\ &\subseteq (st(\mathcal{C})\sigma\sigma' \smallsetminus \mathcal{N}_{\mathsf{loc}}) \cup names(\mathcal{P}, \mathcal{S}, \Phi, \mathcal{M}, \mathcal{I}_f)\end{aligned}$$

Let S be the DAG size of $\mathcal{P}$, $\mathcal{S}$, Φ, $\mathcal{M}$, and $\mathcal{I}_f$. By inspection of the symbolic transition rules, we see that at each step, the constraint system can grow at most of size S (because of the communication rule). Hence, we have that $\#st(\mathcal{D}', \Phi_2) \leq \mathsf{S}^2$.

Let N be the maximal depth of variables in the terms of all disequality constraints in Φ_2, and k the maximal total number of variables in a disequality constraint. We have that $kN \leq \mathsf{D}^2$ where D is the DAG size of the largest disequality constraint that occurs in $\mathcal{D}'$. Since $\mathsf{D} \leq \#st(\mathcal{D}', \Phi_2)$, we deduce that $kN \leq \mathsf{D}^2 \leq \mathsf{S}^4$.

Let L be the number of occurrences of a loop predicate in Φ_2, R be the number of occurrences of a route predicate in Φ_2, and C be the number of occurrences of a checkl predicate in Φ_2. We have that:

$$L \leq \mathsf{S}^2,\ R \leq \mathsf{S}^2, \text{and } C \leq \mathsf{S}^2.$$

Now, we have to show that if there exists a graph $G = (\mathcal{N}_{\mathsf{loc}}, E)$ such that $E_K = \{(n_1, n_2) \in E \mid n_1, n_2 \in V_K\}$ and on which there is an attack, then there exists a graph as described in Step 6 for which there is an attack and the substitution witnessing the fact that there exists an attack is also as described in Step 6 of our algorithm.

- Thanks to Lemma 3.3.3, we know that there is a solution where the variables which are not of sort loc or lists are substituted by names in $\mathcal{I}_0$ (independently of the underlying graph).
- Thanks to Proposition 3.3.5, we know that if there is a graph $G = (\mathcal{N}_{\mathsf{loc}}, E)$ leading to a solution, there exists a substitution σ where the size of the instantiated variables of sort lists is bounded by $M = 2 \times (kN + 3C + R + L + 2)$ and there exists a graph $G' = (\mathcal{N}_{\mathsf{loc}}, E')$ that coincides with G on $V = \{n \mid \exists n' \text{ such that } (n, n') \in E\}$.

  We have that: $M \leq 2 \times (\mathsf{S}^4 + 5\mathsf{S}^2 + 2)$. Hence, the number of distinct names of sort loc in σ is bounded by $\#var(\mathcal{D}', \Phi_2) \times M \leq 2\mathsf{S}^2 \times (\mathsf{S}^4 + 5\mathsf{S}^2 + 2)$. We consider a set $\mathcal{I}_0'$ having this size. So, there is a solution σ for G' such that $names(\sigma) \subseteq \mathcal{I}_0' \cup names(\mathcal{P}, \mathcal{S}, \Phi, \mathcal{M}, \mathcal{I}_f)$.
- Thanks to Lemma 3.3.6, we know that if σ is a solution for graph $G' = (\mathcal{N}_{\mathsf{loc}}, E')$, then σ is also a solution for any graph $G'' = (\mathcal{N}_{\mathsf{loc}}, E'')$ that coincides with G' on $\mathcal{N}'_{\mathsf{loc}}$ where $\mathcal{N}'_{\mathsf{loc}}$ represents the names in $\mathcal{N}_{\mathsf{loc}}$ that occur in $\mathcal{D}'$, Φ_2, and σ. Note that $\mathcal{N}'_{\mathsf{loc}} \subseteq \mathcal{I}_0' \cup names(\mathcal{P}, \mathcal{S}, \Phi, \mathcal{M}, \mathcal{I}_f)$

  Let $G'' = (\mathcal{N}_{\mathsf{loc}}, E'')$ be the graph such that

  $$E'' = \{(n_1, n_2) \in E' \mid n_1, n_2 \in \mathcal{I}_0' \cup names(\mathcal{P}, \mathcal{S}, \Phi, \mathcal{M}, \mathcal{I}_f)\}.$$

  We have that σ is a solution for the graph G'' and the graph G'' is as described in Step 6.

□

3.4.2 Case of a fixed topology

We will now explain how to decide the existence of an attack given a fixed graph G. The procedure is similar to the procedure in the case of an unknown topology, but we no longer have to guess part of the topology at the beginning of the decision procedure.

Theorem 3.4.2. *Let $K = (\mathcal{P}[\_]; \mathcal{S}; \mathcal{I})$ be an initial concrete configuration with a hole, $G = (\mathcal{N}_{\mathsf{loc}}, E)$ be a finite graph, $\mathcal{M} \subseteq \mathcal{N}_{\mathsf{loc}}$ be a finite set of nodes, and $\Phi \in \mathcal{L}_{\mathsf{route}}$ be a formula. Deciding whether there exists an $\mathcal{M}$-attack on K and Φ for the topology G is NP-complete.*

Let $K_s = (\mathcal{P}[\mathsf{if}\ \Phi\ \mathsf{then}\ \mathsf{out}(\mathsf{error})\ \mathsf{else}\ 0]; \mathcal{S}; \mathcal{I}; \emptyset; \emptyset)$. First, K_s is a ground symbolic configuration whose concretization is $(\mathcal{P}[\mathsf{if}\ \Phi\ \mathsf{then}\ \mathsf{out}(\mathsf{error})\ \mathsf{else}\ 0]; \mathcal{S}; \mathcal{I})$. We write $G = (\mathcal{N}_{\mathsf{loc}}, E)$. Let $V = \{n \mid \exists n' \text{ such that } (n, n') \in E\}$. Our decision procedure works as follows:

Step 1 We guess a path of execution of the symbolic transition rules w.r.t. graph G.

$$K_s \rightarrow^{s*}_{G,\mathcal{M}} (\lfloor \mathsf{out}(u) \rfloor_n \cup \mathcal{P}'; \mathcal{S}'; \mathcal{I}'; \mathcal{C}; \Psi).$$

Step 2 Let $\mathcal{I}' = \mathcal{I}_{names} \uplus \mathcal{I}_{\mathsf{terms}}$ such that $(\mathcal{C}, \mathcal{I}_{names})$ is a constraint system. Let $\sigma = mgu(u, \mathsf{error})$ and $\mathcal{C}\sigma = \mathcal{C}'_0$ and $\Psi\sigma = \Phi'_1$.

Step 3 We guess a sequence of transformation rules from $(\mathcal{D}, \mathcal{I}_{names})$ to $(\mathcal{D}', \mathcal{I}_{names})$ where $(\mathcal{D}', \mathcal{I}_{names})$ is a constraint system in solved form. We have that:

$$(\mathcal{D}, \mathcal{I}_{names}) \rightsquigarrow^*_{\sigma'} (\mathcal{D}', \mathcal{I}_{names}) \text{ with } (\mathcal{D}', \mathcal{I}_{names}) \text{ in solved form.}$$

Step 4 We compute the conjunctive normal form of formula $\Phi'_1\sigma$. Hence, $\Phi'_1\sigma$ is equivalent to

$$\bigwedge_k \phi^k_1 \vee \cdots \vee \phi^k_{i_k}.$$

We choose non-deterministically $\phi^k_{\alpha_k}$ for every k. Let $\Phi_2 = \bigwedge_k \phi^k_{\alpha_k}$.

Step 5 Let S be the DAG size of $\mathcal{P}$, $\mathcal{S}$, Φ, $\mathcal{M}$, and $\mathcal{I}_f$. Let $\mathcal{I}'_0$ be a finite subset of $\mathcal{I}_0$ of size $\mathsf{S}^2 \times \mathsf{max}(\mathsf{S}^4 + 5\mathsf{S}^2 + 3, \#E)$. Guess the values of variables of sort lists among lists of nodes in $\mathcal{I}'_0 \cup names(\mathcal{P}, \mathcal{S}, \Phi, \mathcal{M}, \mathcal{I}_f) \cup V$ of length at most $\mathsf{max}(\mathsf{S}^4 + 5\mathsf{S}^2 + 3, \#E)$. Guess the values of the other variables, i.e. those that are not of sort lists, in $\mathcal{I}'_0 \cup names(\mathcal{P}, \mathcal{S}, \Phi, \mathcal{M}, \mathcal{I}_f) \cup V$. This gives us a substitution σ. Lastly, we check whether σ is a solution of $(\mathtt{D}\ ', \mathcal{I}_0) \wedge \Phi_2$ for graph G.

Proof. The first four steps are the same as Steps 2 to 5 in Theorem 3.4.1. Thus, it remains to justify Step 5 of the procedure described above. As shown in the proof of Step 6 in Theorem 3.4.1, we have that:

- $N \leq \mathsf{S}^2$ where N is the maximal depth of variables in the terms of all disequality constraints in Φ_2;
- $k \leq \mathsf{S}^2$ where k is the maximal total number of variables in a disequality constraint in Φ_2;
- $L \leq \mathsf{S}^2$ where L is the number of occurrences of a loop predicate in Φ_2;
- $C \leq \mathsf{S}^2$ where C is the number of occurrences of a checkl predicate in Φ_2;
- $R \leq \mathsf{S}^2$ where R is the number of occurrences of a route predicate in Φ_2.

Now, we want to show that if there exists an attack for graph G, then there is an attack captured by a substitution as described in Step 5 of our algorithm.

- Thanks to Lemma 3.3.3, we know that there is a solution where the variables which are not of sort lists are substituted by names in $\mathcal{I}_0$.
- Thanks to Proposition 3.3.4, we know that if there is a solution, then there exists in particular a solution, say σ, such that $|x\sigma| \leq M$ for any x of type lists where:

$$M = \mathsf{max}(kN + 3C + L + R + 3, \#E).$$

Actually, we have that $M \leq \mathsf{max}(\mathsf{S}^4 + 5\mathsf{S}^2 + 3, \#E|)$.
Hence, the number of distinct names of sort loc in σ is bounded by

$$\#var(\mathcal{D}', \Phi_2) \times M \leq \mathsf{S}^2 \times \mathsf{max}(\mathsf{S}^4 + 5\mathsf{S}^2 + 3, \#E).$$

We consider a set $\mathcal{I}'_0$ having this size. This allows us to conclude. □

3.5 Applications

We present now a few applications and we discuss some limitations of our results.

3.5.1 Routing protocol SRP applied to DSR

Our decision procedure allows us to retrieve the attack on the protocol SRP applied to DSR, mentioned in Example 3.1.4. Indeed, consider the formal model of SRP applied to DSR (defined in Section 3.1.2) and of its desired property (defined in Example 3.1.4). We would first guess the graph G_0 defined in Example 3.1.2. Executing symbolically (non deterministically) the process modeling SRP applied to DSR, we would obtain the symbolic configuration of Example 3.2.2. Applying our transformation rules, we would then (non deterministically) obtain a solved constraint system. We can finally guess the (bounded) solution $\theta' = \{x_a \mapsto X, x_l \mapsto [W; S]\}$.

3.5.2 Routing protocol SDMSR

The secured routing protocol SDMSR introduced in [BYLM06] is a multipath routing protocol that can be modeled in our framework. The goal of a multipath routing protocol is to find several paths leading from a source node S to a destination node D. In order to achieve such a result, the intermediate nodes may proceed the same request several times. This protocol is based on two authentication mechanisms: RSA signatures and signatures based on hash chains. The purpose of the latter scheme is to decrease computation time. For the sake of simplicity, we describe the protocol without this mechanism. The description in [BYLM06] does not really state whether neighbor verification is performed in the protocol. To avoid straightforward attacks, we assume that it is the case: each node checks whether the received information are consistent with its knowledge of the network.

To discover a route to the destination, the source constructs a request packet and broadcasts it to its neighbors. The request packet contains its name S, the name of the destination D, an identifier of the request Id, a list containing the beginning of a route to D, and a signature over the content of the request, computed with the private key $\mathsf{priv}(S)$. The source then waits for a reply containing a route to D signed by one of his neighbors, and checks that this route is plausible.

The process executed by a source node S initiating the search of a route towards a destination node D is

$$P_{init}(S,D) = \mathsf{new}\ Id.\mathsf{out}(u_1).\mathsf{in}\ u_2[\Phi_S].0$$

$$\text{where } \begin{cases} u_1 = \langle \mathsf{req}, S, D, Id, S :: [], [\![\langle \mathsf{req}, S, D, Id\rangle]\!]_{\mathsf{priv}(S)}\rangle \\ u_2 = \langle \mathsf{rep}, D, S, Id, x_A, x_L, [\![\langle \mathsf{rep}, D, S, Id, x_L\rangle]\!]_{\mathsf{priv}(x_A)}\rangle \\ \Phi_S = \mathsf{check}(S, x_A) \wedge \mathsf{checkl}(S, x_L) \end{cases}$$

The names of the intermediate nodes are accumulated in the route request packet. Intermediate nodes relay the request over the network, except if they have already seen a shorter one. In order to simplify the presentation, we consider that they relay all requests as long as they contain different routes. An intermediate node also checks that the received request is correctly authenticated by checking the attached signature. Below, $V \in \mathcal{N}_{\mathsf{loc}}$, x_S, x_a and x_D are variables of sort loc whereas x_r is a variable of sort lists and x_{Id} is a variable of sort terms. The process executed by an intermediate node V when forwarding a request is as follows:

$$P_{\mathsf{req}}(V) = \mathsf{in}\ w_1[\Phi_V].\mathsf{read}\ t\ \mathsf{then}\ 0\ \mathsf{else}\ (\mathsf{store}(t).\mathsf{out}(w_2))$$

$$\text{where } \begin{cases} w_1 = \langle \mathsf{req}, x_S, x_D, x_{Id}, x_a :: x_r, [\![\langle \mathsf{req}, x_S, x_D, x_{Id}\rangle]\!]_{\mathsf{priv}(x_S)}\rangle \\ \Phi_V = \mathsf{check}(V, x_a) \\ t = \langle x_S, x_D, x_{Id}, x_a :: x_r\rangle \\ w_2 = \langle \mathsf{req}, x_S, x_D, x_{Id}, V :: x_a :: x_r, [\![\langle \mathsf{req}, x_S, x_D, x_{Id}\rangle]\!]_{\mathsf{priv}(x_S)}\rangle \end{cases}$$

When the request reaches the destination D, he checks that the request comes from one of its neighbors, has a correct signature, and that the list of accumulated nodes does not contain a loop. Then, the destination D constructs a route reply, in particular it computes a signature over the route accumulated in the request packet with its private key $\mathsf{priv}(D)$. It then sends the reply back over the network. The process executed by the destination node D is $P_{dest}(D) = \mathsf{in}\ v_1[\Phi_D].\mathsf{out}(v_2).0$ where:

$$\begin{array}{l} v_1 = \langle \mathsf{req}, x_S, D, x_{Id}, x_b :: x_l, [\![\langle \mathsf{req}, x_S, D, x_{Id}\rangle]\!]_{\mathsf{priv}(x_S)}\rangle \\ \Phi_D = \mathsf{check}(D, x_b) \wedge \neg\mathsf{loop}(x_b :: x_l) \\ v_2 = \langle \mathsf{rep}, D, x_S, x_{Id}, D, D :: x_b :: x_l, [\![\langle \mathsf{rep}, D, x_S, x_{Id}, D :: x_b :: x_l\rangle]\!]_{\mathsf{priv}(D)}\rangle \end{array}$$

Then, the reply travels along the route back to S. The intermediate nodes check that the signature in the reply packet is correct, and that the route is plausible, before forwarding it. Each node replaces the signature in the reply packet by its own signature. The process executed by an intermediate node V when forwarding a reply is the following one:

$$P_{\mathsf{rep}}(V) = \mathsf{in}\ w'[\Phi'_V].\mathsf{out}(w'').0$$

$$\text{where } \begin{cases} w' = \langle \mathsf{rep}, x_D, x_S, x_{Id}, x_a, x_r, [\![\langle \mathsf{rep}, x_D, x_S, x_{Id}, x_r\rangle]\!]_{\mathsf{priv}(x_a)}\rangle \\ \Phi'_V = \mathsf{checkl}(V, x_r) \wedge \mathsf{check}(V, x_a) \\ w'' = \langle \mathsf{rep}, x_D, x_S, x_{Id}, V, x_r, [\![\langle \mathsf{rep}, x_D, x_S, x_{Id}, x_r\rangle]\!]_{\mathsf{priv}(V)}\rangle \end{cases}$$

We have found that SDMSR is subject to the same kind of attack than SRP applied to DSR. Consider the same graph G_0 as for the attack we described on SRP. Let

$$K_0 = (\lfloor P_{init}(S,D)\rfloor_S \mid \lfloor P_{dest}(D)\rfloor_D; \emptyset; \mathcal{I}_0)$$

The attack scenario is the following one. The source S sends a route request towards D. The request reaches the node n_I. Thus, the attacker receives the following message:

$$\langle \mathsf{req}, S, D, Id, S :: [], [\![\langle \mathsf{req}, S, D, Id\rangle]\!]_{\mathsf{priv}(S)}\rangle.$$

The attacker then broadcasts the following message in the name of X:

$$\langle \mathsf{req}, S, D, id, [X; W; I; S], [\![\langle \mathsf{req}, S, D, Id\rangle]\!]_{\mathsf{priv}(S)}\rangle.$$

Since D is a neighbor of n_I, it will hear the transmission. In addition, the list of nodes $[X; W; I; S]$ ends with X, which is also a neighbor of D, and does not contain any loop, and signature $[\![\langle \mathsf{req}, S, D, Id\rangle]\!]_{\mathsf{priv}(S)}$ is valid. Consequently, the destination D will process this request and will send the following route reply back to S:

$$\langle \mathsf{rep}, D, S, Id, D, [D; X; W; I; S], [\![\langle \mathsf{rep}, D, S, Id, [D; X; W; I; S]\rangle]\!]_{\mathsf{priv}(D)}\rangle.$$

The attacker will put its own signature $[\![\langle \mathsf{rep}, D, S, Id, [D; X; W; I; S]\rangle]\!]_{\mathsf{priv}(n_I)}$ instead of the signature of D, and it will send the resulting message to S.

To model security in our model, we replace in P_{init} the process 0 by a hole and we check whether the formula $\neg\mathsf{route}(x_L)$ holds. Applying our procedure to the initial configuration K_0, we can reach the configuration

$$K_s = (\lfloor \mathsf{out}(\mathsf{error}).0\rfloor_S; \emptyset; \mathcal{I}_0 \cup \{u_1, v_2\}; \mathcal{C}; \Psi)$$

where

$$\mathcal{C} = \{\mathcal{I}_0 \cup \{u_1\} \Vdash v_1 \ \wedge \ \mathcal{I}_0 \cup \{u_1, v_2\} \text{ and } \Psi = \Phi_D \ \wedge \Phi_S \ \wedge \ \neg\mathsf{route}(x_L)\}$$

with:

$$\begin{aligned}
u_1 &= \langle \mathsf{req}, S, D, Id, S :: [], [\![\langle \mathsf{req}, S, D, Id\rangle]\!]_{\mathsf{priv}(S)}\rangle\\
u_2 &= \langle \mathsf{rep}, D, S, Id, x_A, x_L, [\![\langle \mathsf{rep}, D, S, Id, x_L\rangle]\!]_{\mathsf{priv}(x_A)}\rangle\\
\Phi_D &= \mathsf{check}(D, x_b) \wedge \neg\mathsf{loop}(x_b :: x_l)\\
\Phi_S &= \mathsf{check}(S, x_A) \wedge \mathsf{checkl}(S, x_L)\\
v_1 &= \langle \mathsf{req}, x_S, D, x_{Id}, x_b :: x_l, [\![\langle \mathsf{req}, x_S, D, x_{Id}\rangle]\!]_{\mathsf{priv}(x_S)}\rangle\\
v_2 &= \langle \mathsf{rep}, D, x_S, x_{Id}, D, D :: x_b :: x_l, [\![\langle \mathsf{rep}, D, x_S, x_{Id}, D :: x_b :: x_l\rangle]\!]_{\mathsf{priv}(D)}\rangle
\end{aligned}$$

and the constraint system $(\mathcal{C}, \mathcal{I}_{names})$ together with Ψ has a solution

$$\theta = \{x_{id} \mapsto id, x_S \mapsto S, x_A \mapsto n_I, x_b \mapsto X, x_l \mapsto [W; n_I; S], x_L \mapsto [D; X; W; n_I; S]\}$$

for graph G_0, so there is an $\{n_I\}$-attack on K_0 for G_0. We therefore retrieve the attack mentioned above, that we have discovered while analysing the protocol.

3.5.3 Other routing protocols

Table routing

The model of processes we propose includes the possibility for nodes to store information in some memory. We can therefore model routing protocols based on routing tables, such as SAODV [ZA02], SEAD [HJP03] and ARAN [SDL+02]. However, in such protocols, the actual found route is not sent to the source node but depends on the internal states of the nodes. Security properties such as route validity can thus not be expressed using our route predicate. Due to the way properties are modeled, it is not yet possible to straightforwardly analyze routing correctness of table-based routing protocols in our framework. Indeed, we can only model reachability properties at the moment. In the case of table based routing protocols, it is also crucial to detect the existence of loops, as they could interfere with the good function of the network. Both correctness of the route and loop-freeness depend on the routing tables stored in the internal states. Moreover, in order to model accurately table based routing protocols, our model would have to be modified in the way it deals with stores to add the possibility of updating a routing table. At the moment, we can only add elements to the store and they are never deleted. Analyzing table-based routing protocols would be applicable to wireless ad hoc networks but also to wired networks.

Protocols using different tests

Some protocols aim at ensuring other security property than route correctness, for example that the intruder does not appear on the route obtained through the protocol. This property, and others, are sometimes desired, and we have not modeled them. One restriction of our decidability result is that it holds for a particular logic $\mathcal{L}_{\mathsf{route}}$. We would like to have a more general result, with general conditions on the logic to retain decidability. If we want to add predicates, our result does not hold any more. In this precise case, the addition seems straightforward, we could add a predicate in the logic encoding a property stipulating that the route found is free of malicious nodes, and we are confident that we would still get decidability, but this would require writing a new proof. Intuitively, our proof is built in the following way: we mark some names and then we remove the unmarked names from the lists, possibly adding special nodes in between the cuts. So it seems probable that local predicates, constraining only a small number of names, as do predicates check or loop, could be added very easily. Properties about the entire route would require more attention: if they are preserved by any cut, it is easy to add them, else we have to adapt the special nodes used to connect cut portions of the list (in our model, the ubiquitous nodes to preserve the route property). Defining such abstract conditions would be useful as there are many security properties that could be tested this way.

Recursivity

We have modeled route validity in Example 3.1.4 for the protocol SRP applied to DSR. The same modeling can be applied to most source routing protocols such as Ariadne [HPJ05], endairA [BV04], SRDP [KT09], BISS [CH03]. However, source routing protocols may also perform recursive tests: it is the case for Ariadne and endairA for instance. Such tests are typically performed either by the source or the destination and aim at securing respectively the request or reply phase. These tests can not yet be included in our decision procedures. We wish to add the possibility of testing recursivity to our model. A first step towards this end is presented in Chapter 4.

Chapter 4

Protocols with Recursive Tests

In the previous chapter, we have proposed a formal model to analyze ad hoc routing protocols. However, this model does not allow to analyze all existing routing protocols. In particular, such protocols as make use of recursivity are out of his scope. Such routing protocols [BV04, HPJ05, FGML09] require the nodes (typically the node originating the request) to check that the list they receive could be a valid route. This is usually performed by checking that each node has properly signed (or MACed) some part of the route, the whole incoming message forming a chain where each component is a contribution from a node in the path. Moreover, recursivity is also a component of other protocols. For example, in group protocols, the server or the leader typically has to process a request that contains the contributions of each different agent in the group and these contributions are used to compute a common shared key (see *e.g.* the Asokan-Ginzboorg group protocol [AG00]). Other examples of protocols performing recursive operations involve certification paths for public keys (see *e.g.* X.509 certification paths [HFP98]) or right delegation in distributed systems [Aur99].

Recursive operations may yield complex computations. Therefore it is difficult to check the security of protocols with recursive primitives and very few decision procedures have been proposed for recursive protocols. One of the first decidability results [KW04] holds when the recursive operation can be modeled using tree transducers, which forbids any equality test and also forbids composed keys and chained lists. In [Tru05] recursive computation is modeled using Horn clauses and an NEXPTIME procedure is proposed. This is extended in [KT07] to include the Exclusive Or operator. This approach does however not allow composed keys nor list mapping (where the same operation, *e.g.* signing, is applied to each element of the list). To circumvent these restrictions, another procedure has been proposed [CTR09] to handle list mapping provided that each element of the list is properly tagged. No complexity bound is provided. All these results hold for rather limited classes of recursive operations (on lists of terms). This is due to the fact that even a single input/output step of a protocol may reveal complex information, as soon as it involves a recursive computation. Consequently, recursive primitives very quickly yield undecidability.

In order to obtain decidability for a class of protocols including routing protocols, we limited ourselves to considering protocols that perform standard input/output actions (modeled using usual pattern matching) but that are allowed to perform *recursive tests* such as checking the validity of a route or the validity of a chain of certificates. Indeed, several families of protocols use recursivity only for performing sanity checks at some steps of the protocol. This is in particular the case of secured routing protocols, distributed right delegation, and PKI certification paths.

For checking security of protocols with recursive tests (for a bounded number of sessions), we use the setting of constraint systems defined in Chapter 2 and add tests of membership to recursive languages.

In this chapter, we propose (NPTIME) decision procedures for two classes of recursive languages (used for tests): *link-based recursive languages* and *mapping-based languages*. A link-based recursive language contains chains of links where consecutive links have to satisfy a given relation. A typical example is X.509 public key certificates as defined in [HFP98] that consist in a chain of signatures of the form:

$$[[\![\langle A_1, \mathsf{pub}(A_1)\rangle]\!]_{\mathsf{sk}(A_2)}; [\![\langle A_2, \mathsf{pub}(A_2)\rangle]\!]_{\mathsf{sk}(A_3)}; \cdots ; [\![\langle A_n, \mathsf{pub}(A_n)\rangle]\!]_{\mathsf{sk}(S)}].$$

A mapping-based language contains lists that are based on a list of names (typically names of agents involved in the protocol session) and are uniquely defined by it. Typical examples can be found in the context of routing protocols, when nodes check for the validity of the route. For example, in the SMNDP protocol [FGML09], a route from the source A_0 to the destination A_n is represented by a list $l_{route} = [A_n; \ldots ; A_1]$. This list is accepted by the source node A_0 only if the received message is of the form:

$$[[\![\langle A_n, A_0, l_{route}\rangle]\!]_{\mathsf{sk}(A_1)}; [\![\langle A_n, A_0, l_{route}\rangle]\!]_{\mathsf{sk}(A_2)}; \ldots ; [\![\langle A_n, A_0, l_{route}\rangle]\!]_{\mathsf{sk}(A_n)}].$$

Note that a link $[\![\langle A_n, A_0, l_{route}\rangle]\!]_{\mathsf{sk}(A_i)}$ both depends on the list l_{route} and on its i-th element.

For each of these two languages, we show that it is possible to bound the size of a minimal attack (bounding in particular the size of the lists used in membership tests), relying on the characterization we have obtained in Chapter 2 for solutions of constraint systems. As a consequence, we obtain two new NP decision procedures for two classes of languages that encompass most of recursive tests involved in secured routing protocols and chain certificates.

We capture the recursivity tests that have to be performed with language constraints, that are formally defined in Section 4.1. Furthermore, we also consider a particular class of solutions, where the deducible terms are obtained by composition, and we apply this notion to lists. This enables us in the following sections to prove our decidability results. We define *link-based recursive languages*, which encompass in particular certificate chains, in Section 4.2, and we prove that deciding whether a constraint system with constraints in such a language has a solution is decidable in NP. In Section 4.3, we define *mapping based languages* and we prove a similar result of decidability regarding this class of languages.

4.1 Definitions

We give a global definition of language constraints in order to define the type of problems that we want to consider.

Definition 4.1.1 (language constraint). *Let $\mathcal{L}$ be a language (i.e. a set of terms). An $\mathcal{L}$-*language constraint *associated to some constraint system $(\mathcal{C}, \mathcal{I})$ is a formula of the form $(u_1 \in \mathcal{L})? \wedge \ldots \wedge (u_k \in \mathcal{L})?$ where each u_i is a term such that $vars(u_i) \subseteq vars(\mathcal{C})$ and $st(u_i) \cap \mathcal{I} = \emptyset$.*

A solution *of a constraint system $(\mathcal{C}, \mathcal{I})$ and of an $\mathcal{L}$-language constraint $\phi = (u_1 \in \mathcal{L})? \wedge \ldots \wedge (u_k \in \mathcal{L})?$ is a ground substitution θ such that θ is a solution of $(\mathcal{C}, \mathcal{I})$ and $u_i\theta \in \mathcal{L}$ for any $1 \leq i \leq k$. We denote by $st(\phi)$ the set $\{st(u_i) \mid 1 \leq i \leq k\}$.*

We will use also the notion of *constructive solution* on constraint systems in solved form, which is weaker than the notion of non-confusing solution.

Definition 4.1.2 (constructive solution). *Let $(\mathcal{C},\mathcal{I})$ be a constraint system in solved form. A substitution θ is a* constructive solution *of $(\mathcal{C},\mathcal{I})$ if for every deducibility constraint $T \overset{?}{\vdash} x$ in $\mathcal{C}$, we have that $Sat_v(T)\theta \cup \mathcal{I} \vdash x\theta$ using composition rules only.*

A non-confusing solution of a solved system is a constructive solution, while the converse does not always hold. This notion will be used in proofs in the following sections as we will transform solutions, preserving the constructive property but not necessarily the non-confusing property.

In this chapter, we have to pay extra attention to lists, as they are fundamental constructors in our different languages. In particular, we are going to build smaller lists and we will have to preserve deducibility. In order to achieve that, we show in Lemma 4.1.1 that if we can deduce a list using composition rules only, then we can deduce each of its elements, still using only composition rules.

Lemma 4.1.1. *Let $(\mathcal{C},\mathcal{I})$ be a constraint system in solved form, θ be a constructive solution of $(\mathcal{C},\mathcal{I})$, and $T \in \mathsf{lhs}(\mathcal{C})$. Let $u = [m_1;\ldots;m_n]$ be a list such that $Sat_v(T)\theta \cup \mathcal{I} \vdash u$ using composition rules only. Then for every $k \leq n$, there is a proof of $Sat_v(T)\theta \cup \mathcal{I} \vdash m_k$ using composition rules only.*

Proof. First, we can write $\mathcal{C} = T_1 \overset{?}{\vdash} x_1 \wedge \cdots \wedge T_p \overset{?}{\vdash} x_p$. For $1 \leq i \leq p$, let $\mathcal{S}_i = Sat_v(T_i) \cup \mathcal{I}$. Consider Δ a proof of $\mathcal{S}_i\theta \vdash u$ using only composition rules. We show by induction on $(i, |\Delta|)$ that for every element m_k of u, there exists a proof Δ_k of $\mathcal{S}_i\theta \vdash m_k$ that uses composition rules only. We distinguish cases depending on the last rule of Δ.

The last rule is an axiom. Then $u \in \mathcal{S}_i\theta$ and there is $t \in \mathcal{S}_i$ such that $u = t\theta$. As u is a list, there are terms $e_1, \ldots, e_m, t' \in \mathcal{S}_i$ such that $t = e_1 :: \ldots :: e_m :: t'$ and $t \in \{[]\} \cup \mathcal{X}$. For $1 \leq k \leq m$, $e_k\theta = m_k$, so for $1 \leq k \leq m$, there is a proof of $\mathcal{S}_i\theta \vdash m_k$ which is an axiom.
Now, either $t' = []$ or $t' = x_j \in \mathcal{X}$, with $j < i$. In the first case, we easily conclude. In the second case, as θ is a constructive solution of $(\mathcal{C},\mathcal{I})$, there is a proof Δ' of $\mathcal{S}_i\theta \vdash x_j\theta$ with $j < i$, and we apply the induction hypothesis.

The last rule is a composition rule.

$$\frac{\mathcal{S}_i\theta \vdash m_1 \quad \mathcal{S}_i\theta \vdash [m_2;\ldots;m_n]}{\mathcal{S}_i\theta \vdash u = [m_1;\ldots;m_n]}\text{(List Constr.)}$$

Δ uses composition rules only, so there is a proof of $\mathcal{S}_i\theta \vdash m_1$ using composition rules only, and a proof Δ' using only composition rules of $\mathcal{S}_i\theta \vdash [m_2;\ldots;m_n]$, smaller than Δ. By induction hypothesis, for every element m_k of l, there exists a proof Δ_k of $\mathcal{S}_i\theta \vdash m_k$ that uses composition rules only, and we conclude.

□

Then, we show that we can restrict our attention to solutions θ of $(\mathcal{C},\mathcal{I}) \wedge \phi$ such that $\theta(x)$ is either a constant, a name, or a subterm of $\phi\theta$. This result will also be useful for proving our decidability results.

Lemma 4.1.2. *Let $\mathcal{L}$ be a language, i.e. a set of terms. Let $(\mathcal{C}, \mathcal{I})$ be a constraint system in solved form and ϕ be an $\mathcal{L}$-language constraint associated to $(\mathcal{C}, \mathcal{I})$. Let θ be a constructive solution of $(\mathcal{C}, \mathcal{I})$ and ϕ. Let N_0 be a name of* Base *sort in $\mathcal{I}$, and θ' be a substitution such that:*

$$\begin{cases} x\theta' = x\theta & \text{if } x\theta \in st(\phi\theta) \\ x\theta' = [] & \text{if } x \in \mathcal{X}_{\mathsf{List}} \text{ and } x\theta \notin st(\phi\theta) \\ x\theta' = N_0 & \text{if } x \notin \mathcal{X}_{\mathsf{List}} \text{ and } x\theta \notin st(\phi\theta) \end{cases}$$

The substitution θ' is a constructive solution of $(\mathcal{C}, \mathcal{I})$ and ϕ.

Proof. Write $\mathcal{C} = T_1 \overset{?}{\vdash} x_1 \wedge \cdots \wedge T_n \overset{?}{\vdash} x_n$. As θ is constructive, for every $i \leq n$, there is a proof of $Sat_v(T_i)\theta \cup \mathcal{I} \vdash x_i\theta$ using composition rules only. We show that, for every $i \leq n$, there is a proof of $Sat_v(T_i)\theta' \cup \mathcal{I} \vdash x_i\theta'$ using composition rules only. We distinguish between cases:

- If $x_i\theta \in st(\phi\theta)$, then $x_i\theta' = x_i\theta$ and there exist terms $t^i_1, \ldots, t^i_{k_i} \in Sat_v(T_i)$ and a proof tree Δ_i such that Δ_i is a proof of $\{t^i_1, \ldots, t^i_{k_i}\}\theta \vdash x_i\theta$ using only composition rules. Consequently, for every $j \leq k_i$, we have that $t^i_j\theta \in st(x_i\theta)$. So, for every variable $y \in var(t^i_j)$, we have that $y\theta \in st(x_i\theta)$, and so $t^i_j\theta = t^i_j\theta'$. As a conclusion, Δ_i is a proof of $\{t^i_1, \ldots, t^i_{k_i}\}\theta' \vdash x_i\theta'$ using only composition rules. We deduce that there is a proof of $Sat_v(T_i)\theta' \cup \mathcal{I} \vdash x_i\theta'$ using composition rules only.
- If $x_i\theta \notin st(x_i\theta)$ and $x_i \in \mathcal{X}_{\mathsf{List}}$, $x_i\theta' = []$. Thus there is a proof of $Sat_v(T_i)\theta' \cup \mathcal{I} \vdash x_i\theta'$ using composition rules only.
- If $x_i\theta \notin st(x_i\theta)$ and $x_i \notin \mathcal{X}_{\mathsf{List}}$, then $x_i\theta' = N_0 \in \mathcal{I}$. We immediately deduce that there is a proof of $Sat_v(T_i)\theta' \cup \mathcal{I} \vdash x_i\theta'$ using composition rules only.

Furthermore, for every $x \in var(\phi)$, we have that $x\theta = x\theta'$. Consequently, we have that $\phi\theta' = \phi\theta$. Thus, we have that θ' is a solution of ϕ. □

The following definitions will be useful in the proofs of decidability.

Definition 4.1.3 (Tail of a list)**.** *The tail* $\mathsf{tail}(l)$ *of a list l is defined recursively as follows:*

$$\begin{cases} \mathsf{tail}([]) & = & [] & \\ \mathsf{tail}(x) & = & x & x \in \mathcal{X}_{\mathsf{List}} \\ \mathsf{tail}(u :: t) & = & \mathsf{tail}(t) & \end{cases}$$

Definition 4.1.4 (Size of lists)**.** *Let l be a list of terms, we define its size $\|l\|_l$ as the number of elements it contains. More precisely, it is defined recursively as*

$$\begin{cases} \|[]\|_l & = & 0 & \\ \|x\|_l & = & 1 & x \in \mathcal{X}_{\mathsf{List}} \\ \|u :: l\|_l & = & \|l\|_l + 1 & \end{cases}$$

A swapping replaces part of a term by another one. We will use this operation to obtain small solutions to language constraints.

Definition 4.1.5 (Swapping)**.** *Let $u_1, \dots, u_n, v_1, \dots, v_n$ be ground terms such that for every $i \neq j$, $u_i \neq u_j$. The* swapping *$\delta = \{u_1 \mapsto v_1, \dots, u_n \mapsto v_n\}$ is defined inductively over a term t in a top-down manner:*

- *if there exists i such that $t = u_i$, then $t\delta = v_i$*
- *if for every $i, t \neq u_i$ and $t = \mathsf{f}(t_1, \dots, t_k)$, then $t\delta = \mathsf{f}(t_1\delta, \dots, t_k\delta)$.*

We denote by id the empty swapping.

4.2 Link-based recursive languages

We define a class of languages that encompasses for example certificate chains, and we show that when considering constraint systems together with constraints in this class of languages, deciding the existence of a solution is in NP.

4.2.1 Definition and Examples

A chain of certificates is typically formed by a list of links such that consecutive links follow a certain relation. For example, the chain of public key certificates

$$[[\![\langle A_1, \mathsf{pub}(A_1)\rangle]\!]_{\mathsf{sk}(A_2)}; [\![\langle A_2, \mathsf{pub}(A_2)\rangle]\!]_{\mathsf{sk}(A_3)}; [\![\langle A_3, \mathsf{pub}(A_3)\rangle]\!]_{\mathsf{sk}(S)}]$$

is based on the link $[\![\langle x, \mathsf{pub}(y)\rangle]\!]_{\mathsf{sk}(z)}$. We provide a generic definition that captures such link-based recursive language.

Definition 4.2.1 (link-based recursive language)**.** *Let m be a term built over variables of sort Base. A* link-based recursive language *$\mathcal{L}$ is defined by three terms w_0, w_1, w_2 such that $w_i = \mathsf{m}\theta_i^1 :: \dots :: \mathsf{m}\theta_i^{k_i} :: x_i^{\mathsf{m}}$ for $i = 0, 1, 2$, and w_2 is a strict subterm of w_1.*

Once w_0, w_1, w_2 are given, the language is recursively defined as follows. A ground term t belongs to the language $\mathcal{L}$ if either $t = w_0\sigma$ for some σ such that $x_0^{\mathsf{m}}\sigma = []$, or there exists σ such that $t = w_1\sigma$, and $w_2\sigma \in \mathcal{L}$.

Intuitively, w_0 is the basic valid chain while w_1 encodes the desired dependence between the links and w_2 allows for a recursive call.

Example 4.2.1. *As defined in [HFP98], X.509 public key certificates consist in chains of signatures of the form:*

$$[[\![\langle A_1, \mathsf{pub}(A_1)\rangle]\!]_{\mathsf{sk}(A_2)}; [\![\langle A_2, \mathsf{pub}(A_2)\rangle]\!]_{\mathsf{sk}(A_3)}; \cdots; [\![\langle A_n, \mathsf{pub}(A_n)\rangle]\!]_{\mathsf{sk}(S)}]$$

where S is some trusted server and each agent A_{i+1} certifies the public key $\mathsf{pub}(A_i)$ of agent A_i. These chained lists are all built from the term $\mathsf{m} = [\![\langle x, \mathsf{pub}(y)\rangle]\!]_{\mathsf{sk}(z)}$ with $x, y, z \in \mathcal{X}_{\mathsf{Base}}$. The set of valid chains of signatures can be formally expressed as the m-link-based recursive language $\mathcal{L}_{\mathsf{cert}}$ defined by:

$$\begin{cases} w_0 = [\![\langle x, \mathsf{pub}(x)\rangle]\!]_{\mathsf{sk}(S)} :: x_0^{\mathsf{m}}, \\ w_1 = [\![\langle x, \mathsf{pub}(x)\rangle]\!]_{\mathsf{sk}(y)} :: [\![\langle y, \mathsf{pub}(y)\rangle]\!]_{\mathsf{sk}(z)} :: x_1^{\mathsf{m}}, \\ w_2 = [\![\langle y, \mathsf{pub}(y)\rangle]\!]_{\mathsf{sk}(z)} :: x_1^{\mathsf{m}}. \end{cases}$$

Similarly, link-based recursive languages can also describe delegation rights certificates in the context of distributed access-rights management. In [Aur99] for example, the certificate chains delegating authorization for operation O are of the form:

$$[[\![\langle A_1, \mathsf{pub}(A_1), O\rangle]\!]_{\mathsf{sk}(A_2)}; [\![\langle A_2, \mathsf{pub}(A_2), O\rangle]\!]_{\mathsf{sk}(A_3)}; \dots; [\![\langle A_n, \mathsf{pub}(A_n), O\rangle]\!]_{\mathsf{sk}(S)}]$$

where S has authority over operation O and each agent A_{i+1} delegates the rights for operation O to agent A_i. These chained lists are all built from the term $\mathsf{m} = [\![\langle x, \mathsf{pub}(y), O\rangle]\!]_{\mathsf{sk}(z)}$ with $x, y, z \in \mathcal{X}_{\mathsf{Base}}$.

Example 4.2.2. *In the recursive authentication protocol [Pau97], a certificate list consists in a chain of encryptions of the form:*

$$[\mathsf{senc}(\langle K_{ab}, B, N_a\rangle, K_a); \mathsf{senc}(\langle K_{ab}, A, N_b\rangle, K_b);$$
$$\mathsf{senc}(\langle K_{bc}, C, N_b\rangle, K_b); \mathsf{senc}(\langle K_{bc}, B, N_c\rangle, K_c); \dots; \mathsf{senc}(\langle K_{ds}, S, N_d\rangle, K_d)]$$

where S is a trusted server distributing session keys K_{ab}, K_{bc}, ..., K_{ds} to each pair of successive agents via these certificates. These chained lists are all built from the term $\mathsf{m} = \mathsf{senc}(\langle y_1, y_2, y_3\rangle, z)$ with $y_1, y_2, y_3, z \in \mathcal{X}_{\mathsf{Base}}$. The set of valid chains of encryptions in this protocol can be formally expressed as the m-link-based recursive language $\mathcal{L}_{\mathsf{RA}}$ defined by:

$$\begin{cases} w_0 = \mathsf{senc}(\langle z, S, x\rangle, x_k) :: x_0^{\mathsf{m}} \\ w_1 = \mathsf{senc}(\langle z, x_a, x\rangle, x_{k_b}) :: \mathsf{senc}(\langle z, x_b, y\rangle, x_{k_a}) :: \mathsf{senc}(\langle z', x_c, y\rangle, x_{k_a}) :: x_1^{\mathsf{m}} \\ w_2 = \mathsf{senc}(\langle z', x_c, y\rangle, x_{k_a}) :: x_1^{\mathsf{m}}. \end{cases}$$

4.2.2 A procedure considering link-based recursive tests

We propose a procedure for checking for secrecy preservation for a protocol with link-based recursive tests in NP, for a bounded number of sessions.

The goal of this section is to prove that checking for secrecy preservation for a protocol with link-based recursive tests is NP, for a bounded number of sessions (Theorem 4.2.1). To achieve this goal, we will show that we can bound in advance the length of the recursive lists.

We write $names(u)$ for the set of names occurring in u. This notation is extended as expected to sets of terms, constraint systems, ... Let S be a set, we denote by $\#S$ the cardinal of S.

Let l be a list (not necessarily ground), we denote by $\| l \|_l$ its length. By convention, $\| x \|_l = 1$ when x is a variable.

Theorem 4.2.1. *Let $\mathcal{L}$ be a link-based recursive language. Let $(\mathcal{C}, \mathcal{I})$ be a constraint system and ϕ be an $\mathcal{L}$-language constraint associated to $(\mathcal{C}, \mathcal{I})$. Deciding whether $(\mathcal{C}, \mathcal{I})$ and ϕ has a solution is in NP.*

The proof of Theorem 4.2.1 involves three main steps. First, thanks to Theorem 2.3.1, it is sufficient to decide in polynomial (DAG) size whether $(\mathcal{C}, \mathcal{I})$ with language constraint ϕ has a non-confusing solution when $(\mathcal{C}, \mathcal{I})$ is a solved constraint system. Then, we show that we can (polynomially) bound the size of the lists in ϕ. This relies partly on Proposition 2.4.1, as it shows that a non-confusing solution is a constructive solution.

A m-*link* is a ground instance of m. A m-*sublink* is a subterm of such an instance. When the term m is clear from the context, we may simply say link and sublink instead of m-link and m-sublink.

We first show that we can replace a list with another list built with some of its elements while preserving the deducibility of sublinks.

Lemma 4.2.2. *Let $\mathcal{C}$ be a deduction constraint system in solved form, θ a substitution such that θ is a constructive solution of $(\mathcal{C},\mathcal{I})$. Suppose that there exist $x_l \in \mathcal{X}_{\mathsf{List}}$ and* m*-links $m_1,\dots,m_p$ such that $x_l\theta = [m_1;\dots;m_p]$. Define a substitution θ' such that if $x \neq x_l$, $x\theta' = x\theta$. Let $T \in \mathsf{lhs}(\mathcal{C})$. A sublink deducible from $Sat(T)\theta \cup \mathcal{I}$ by composition rules only is still deducible from $Sat(T)\theta' \cup \mathcal{I}$ by composition rules only.*

Proof. Let $T \in \mathsf{lhs}(\mathcal{C})$, u be a sublink such that u is deducible from $Sat(T)\theta \cup \mathcal{I}$ by using composition rules only. Hence, there exist a prooftree Δ and terms $t_1,\dots,t_q \in Sat_v(T) \cup \mathcal{I}$ such that Δ is a proof of $\{t_1,\dots,t_q\}\theta \vdash u$ using composition rules only. As u is a sublink, for every $1 \leq i \leq q$, we have that $t_i\theta$ is a sublink. We show that for every $1 \leq i \leq q$, we have that $t_i\theta = t_i\theta'$. As $t_i\theta$ is an m-sublink, we know that $x_l \notin var(t_i)$. Indeed, $x_l \in \mathcal{X}_{\mathsf{List}}$, and by definition of m, we have that $vars(\mathsf{m}) \subseteq \mathcal{X}_{\mathsf{Base}}$. Consequently, for each $1 \leq i \leq q$, we have that $t_i\theta' = t_i\theta$, and so Δ is a proof of $\{t_1,\dots,t_q\}\theta' \vdash u$ using composition rules only. Hence, we have that u is deducible from $Sat(T)\theta' \cup \mathcal{I}$ by using composition rules only. □

We prove in Proposition 4.2.4 that we can consider only small solutions. Indeed, we first show that there is a constructive solution that uses a bounded number of distinct names. Thus there is a finite number of instances of m used in recursive calls, allowing us to cut the lists while preserving the membership to the recursive language.

Proposition 4.2.4. *Let $\mathcal{L}$ be a* m*-link-based recursive language defined by w_0, w_1, and w_2. Let $(\mathcal{C},\mathcal{I})$ be a constraint system in solved form and $\phi = l_1 \overset{?}{\in} \mathcal{L} \wedge \ldots \wedge l_n \overset{?}{\in} \mathcal{L}$ be a $\mathcal{L}$-language constraint associated to $(\mathcal{C},\mathcal{I})$. Let θ be a constructive solution of $(\mathcal{C},\mathcal{I})$ and ϕ. Then there exists a constructive solution θ' of $(\mathcal{C},\mathcal{I})$ and ϕ such that, for every $1 \leq i \leq n$, we have that $\|\mathsf{tail}(l_i)\theta'\|_l \leq M = k_0 + (k_1 - k_2) \times N^{k_1}$ where $N = (\#names(\mathcal{C},\phi,w_0,w_1,w_2)+1)^{\#vars(\mathsf{m})}$.*

Proof. First, note that if θ is a solution of $(\mathcal{C},\mathcal{I})$ and ϕ, and δ is a renaming of all names in $\mathcal{I}$ that do not occur in w_0, w_1, w_2 by the same name N_0 of Base sort, then $\theta\delta$ is a solution of $(\mathcal{C},\mathcal{I})$ and ϕ. We can thus consider (constructive) solutions containing at most $names(\{w_0,w_1,w_2\})+1$ names of $\mathcal{I}$.

Thanks to Lemma 4.1.2, we can furthermore assume that for every x, either $x\theta \in st(\phi\theta)$, or $x\theta \in \{N_0, []\}$ (more precisely, $x\theta = []$ if $x \in \mathcal{X}_{\mathsf{List}}$ and $x\theta = N_0$ otherwise).

Write $\phi = l_1 \in \mathcal{L} \wedge \ldots \wedge l_n \in \mathcal{L}$. Consider a smallest constructive solution θ, where the size of θ is given by

$$|\theta| = \sum_{1\leq j\leq n} \|\mathsf{tail}(l_j)\theta\|_l.$$

Either for every $j \leq n$, we have that $\|\mathsf{tail}(l_j)\theta\|_l \leq M$ and we conclude directly, or there exists $j_0 \leq n$ such that $\|\mathsf{tail}(l_{j_0})\theta\|_l > M$. In the second case, we define $x_{j_0} = \mathsf{tail}(l_{j_0})$, and we show that we can build θ' a constructive solution of $(\mathcal{C},\mathcal{I})$ and ϕ smaller than θ, which leads to a contradiction and allows us to conclude. More precisely, we build θ' such that $x\theta' = x\theta$ for $x \neq x_{j_0}$ and $\|x_{j_0}\theta'\|_l < \|x_{j_0}\theta\|_l$.

We build θ' smaller than θ. Intuitively, N^{k_1} is the number of possible instantiations of the pattern-matching prefix of w_1 (i.e. $\mathsf{m}\theta_1^1,\dots,\mathsf{m}\theta_1^{k_1}$). Remember that k_0, k_1 and k_2 are respectively the number of links in w_0, w_1 and w_2 in the definition of the language $\mathcal{L}$.

We consider the successive recursive calls made in order to prove that $l_j\theta \in \mathcal{L}$. By definition of $\mathcal{L}$, a term t belongs to the language $\mathcal{L}$ if

- either there exists a substitution σ with $x_0^{\mathsf{m}}\sigma = []$ such that $t = w_0\sigma$,
- or there exists a substitution σ such that $t = w_1\sigma$, and $w_2\sigma \in \mathcal{L}$.

Consequently, $l_{j_0}\theta \in \mathcal{L}$ if and only if there is a sequence of terms $t_0, \dots, t_p$ and substitutions $\sigma_0, \dots, \sigma_p$ such that

- $l_{j_0}\theta = t_0$,
- for every $0 \leq i < p$, $t_i = w_1\sigma_i$ and $t_{i+1} = w_2\sigma_i$,
- $x_0^{\mathsf{m}}\sigma_p = []$ and $t_p = w_0\sigma_p$.

For every $0 \leq i < p$, define substitution $\overline{\sigma}_i$ such that $dom(\overline{\sigma}_i) = dom(\sigma_i) \smallsetminus \{x_1^{\mathsf{m}}\}$, and for every $x \in dom(\overline{\sigma}_i)$, $x\overline{\sigma}_i = x\sigma_i$. We want to find $i_1 < i_2$ such that $\overline{\sigma}_{i_1} = \overline{\sigma}_{i_2}$ in order to cut the list while still having a sequence of correct recursive calls. This is possible if the sequence of recursive calls is long enough, as there are only a finite number of possible instantiations of $\overline{\sigma}_i$. Indeed, note that $dom(\overline{\sigma}_i) = vars(\mathsf{m}\theta_1^1, \dots, \mathsf{m}\theta_{k_1}^1) \subseteq \mathcal{X}_{\mathsf{Base}}$ and $\#dom(\overline{\sigma}_i) \leq \#vars(\mathsf{m}) \times k_1$. The number of possible values for $\overline{\sigma}_i$ is N^{k_1}.

For every $0 \leq i < p$, $\|t_i\|_l = \|w_1\sigma_i\|_l = k_1 + \|x_1^{\mathsf{m}}\sigma_i\|_l$ and $\|t_{i+1}\|_l = \|w_2\sigma_i\|_l = k_2 + \|x_2^{\mathsf{m}}\sigma_i\|_l$. By definition of a link-based recursive language, w_2 is a subterm of w_1 and so $x_1^{\mathsf{m}} = x_2^{\mathsf{m}}$. We deduce that $\|t_{i+1}\|_l = \|x_1^{\mathsf{m}}\sigma_i\|_l + k_2 = \|t_i\|_l + k_2 - k_1$. Moreover, $\|t_p\|_l = k_0$. Hence, $\|t_0\|_l = k_0 + (k_1 - k_2) \times p$. As $\|x_{j_0}\theta\|_l > k_0 + (k_1 - k_2) \times N^{k_1}$, we get that $p > N^{k_1}$, and necessarily there are $i_1 < i_2$ such that $\overline{\sigma}_{i_1} = \overline{\sigma}_{i_2}$, and $\|x_1^{\mathsf{m}}\sigma_{i_1}\|_l < \|x_{j_0}\theta\|_l$. As $i_1 < i_2$, we have that $\|x_1^{\mathsf{m}}\sigma_{i_2}\|_l < \|x_1^{\mathsf{m}}\sigma_{i_1}\|_l$.

We are going to replace $x_1^{\mathsf{m}}\sigma_{i_1}$ with $x_1^{\mathsf{m}}\sigma_{i_2}$ in $x_{j_0}\theta$.
We define the swapping $\delta = \{x_1^{\mathsf{m}}\sigma_{i_1} \mapsto x_1^{\mathsf{m}}\sigma_{i_2}\}$. We show that $(l_{j_0}\theta)\delta \in \mathcal{L}$.
For $0 \leq i \leq i_1$, let $\sigma_i' = \sigma_i\delta$ and $t_i' = t_i\delta$. Define also, for $1 \leq k \leq p - i_2$, $\sigma_{i_1+k}' = \sigma_{i_2+k}$ and $t_{i_1+k}' = t_{i_2+k}$. First, $\sigma_{i_2} = \sigma_{i_1}'$. Indeed, $\sigma_{i_1} = \overline{\sigma}_{i_1} \uplus \{x_1^{\mathsf{m}} \mapsto x_1^{\mathsf{m}}\sigma_{i_1}\}$. Hence, $\sigma_{i_1}' = \sigma_{i_1}\delta = \overline{\sigma}_{i_1} \uplus \{x_1^{\mathsf{m}} \mapsto x_1^{\mathsf{m}}\sigma_{i_1}\}\delta = \overline{\sigma}_{i_2} \uplus \{x_1^{\mathsf{m}} \mapsto x_1^{\mathsf{m}}\sigma_{i_2}\} = \sigma_{i_2}$ (thanks to the choice of i_1, i_2 and δ).

- $t_0' = t_0\delta = (l_{j_0}\theta)\delta$.
- for $0 \leq i < i_1$, $t_i' = t_i\delta = (w_1\sigma_i)\delta = w_1(\sigma_i\delta) = w_1\sigma_i'$ and $t_{i+1}' = t_{i+1}\delta = (w_2\sigma_i)\delta = w_2(\sigma_i\delta) = w_2\sigma_i'$,
- $t_{i_1}' = t_{i_1}\delta = (w_1\sigma_{i_1})\delta = w_1(\sigma_{i_1}\delta) = w_1\sigma_{i_1}'$ and $t_{i_1+1}' = t_{i_2+1} = w_2\sigma_{i_2} = w_2\sigma_{i_1}'$,
- for $1 \leq k < p - i_2$, $t_{i_1+k}' = t_{i_2+k} = w_1\sigma_{i_2+k} = w_1\sigma_{i_1+k}'$ and $t_{i_1+k+1}' = t_{i_2+k+1} = w_2\sigma_{i_2+k} = w_2\sigma_{i_1+k}'$,
- $x_0^{\mathsf{m}}\sigma_{p-i_2+i_1}' = x_0^{\mathsf{m}}\sigma_p = []$ and $t_{p-i_2+i_1}' = t_p = w_0\sigma_p = w_0\sigma_{p-i_2+i_1}'$.

We conclude that $(l_{j_0}\theta)\delta \in \mathcal{L}$ and $\|(l_{j_0}\theta)\delta\|_l < \|l_{j_0}\theta\|_l$. Intuitively, we can replace $x_1^{\mathsf{m}}\sigma_{i_1}$ by $x_1^{\mathsf{m}}\sigma_{i_2}$ in $l_{j_0}\theta$ and still get a correct sequence of recursive calls.

Let θ' be a substitution such that $x\theta' = x\theta$ when $x \neq x_{j_0}$ and $x_{j_0}\theta' = (x_{j_0}\theta)\delta$.

We show that θ' is a solution of ϕ. We have $\phi = l_1 \in \mathcal{L} \wedge \dots \wedge l_n \in \mathcal{L}$. For every j, we write $l_j = m_1^j :: \dots :: m_{p_j}^j :: \mathsf{tail}(l_j)$, with m_k^j of sort Msg and $\mathsf{tail}(l_j) \in \mathcal{X}_{\mathsf{List}} \cup \{[]\}$ of sort List. If $\mathsf{tail}(l_j) \neq x_{j_0}$, $l_j\theta' = l_j\theta$, and so $l_j\theta' \in \mathcal{L}$. If $\mathsf{tail}(l_j) = x_{j_0}$, as θ is a solution of ϕ, $l_j\theta \in \mathcal{L}$. So there is a sequence of terms $u_0, \dots, u_m$ and substitutions $\tau_0, \dots, \tau_m$ such that

- $l_j\theta = u_0$,
- for every $0 \leq i < m$, $u_i = w_1\tau_i$ and $u_{i+1} = w_2\tau_i$,
- $x_0^{\mathsf{m}}\tau_m = []$ and $u_m = w_0\tau_m$.

As $\mathsf{tail}(l_j) = x_{j_0}$, we have that $\sigma_i = \tau_{m-p+i}$ for $i \geq i_1$. Consequently, we can show that $l_j\theta' = l_j(\theta\delta) \in \mathcal{L}$ in the following way: let $j_1 = m-p+i_1$ and $j_2 = m-p+i_2$. For $0 \leq i \leq j_1$, let $\tau'_i = \tau_i\delta$ and $u'_i = u_i\delta$. Define also, for $1 \leq k \leq m - j_2$, $\tau'_{j_1+k} = \tau_{j_2+k}$ and $u'_{j_1+k} = u_{j_2+k}$. First, $\tau_{j_2} = \sigma_{i_2} = \sigma_{i_1}\delta = \tau_{j_1}\delta = \tau'_{j_1}$.

- $u'_0 = u_0\delta = (l_j\theta)\delta$.
- for $0 \leq i < j_1$, $u'_i = u_i\delta = (w_1\tau_i)\delta = w_1(\tau_i\delta) = w_1\tau'_i$ and $t'_{i+1} = t_{i+1}\delta = (w_2\tau_i)\delta = w_2(\tau_i\delta) = w_2\tau'_i$,
- $u'_{j_1} = u_{j_1}\delta = (w_1\tau_{j_1})\delta = w_1(\tau_{j_1}\delta) = w_1\tau'_{j_1}$
  $u'_{j_1+1} = u_{j_2+1} = w_2\tau_{j_2} = w_2\tau'_{j_1}$,
- for every $1 \leq k \leq m - j_2 + 1$, $u'_{j_1+k} = u_{j_2+k} = w_1\tau_{j_2+k} = w_1\tau'_{j_1+k}$ and $u'_{j_1+k+1} = u_{j_2+k+1} = w_2\tau_{j_2+k} = w_2\tau'_{j_1+k}$,
- $x_0\tau'_{p-j_2+j_1} = x_0\tau_p = []$ and $u'_{p-j_2+j_1} = u_p = w_0\tau_p = w_0\tau'_{p-j_2+j_1}$.

Terms $u'_0, \ldots, u'_{p-j_2+j_1}$ and substitutions $\tau'_0, \ldots, \tau'_{p-j_2+j_1}$ characterize a series of recursive calls that allows to prove that $l_j\theta' \in \mathcal{L}$.
We get that for every $1 \leq j \leq n$, $l_j\theta' \in \mathcal{L}$, so θ' is a solution of ϕ.

We show that θ' is a constructive solution of $(\mathcal{C}, \mathcal{I})$. Consider $T \overset{?}{\vdash} x$ a constraint in $\mathcal{C}$.

- If $x\theta \notin st(\phi\theta)$, then we have assumed that $x\theta \in \{N_0, []\}$. As $x\theta \notin st(\phi\theta)$, then $x \neq x_{j_0}$, and $x\theta' = x\theta$. We deduce that $T\theta \cup \mathcal{I} \vdash x\theta'$ using composition rules only.
- If $x\theta \in st(\phi\theta)$, and $x \in \mathcal{X}_{\mathsf{Msg}}$, then $x\theta$ is a sublink and $x\theta' = x\theta$. We know that θ is a constructive solution of $(\mathcal{C}, \mathcal{I})$. Then thanks to Lemma 4.2.2, we have that $Sat(T)\theta' \cup \mathcal{I} \vdash x\theta = x\theta'$ using composition rules only.
- If $x\theta \in st(\phi\theta)$ and $x \in \mathcal{X}_{\mathsf{List}}$, then $x\theta$ is a list of links, *i.e* there exists links $m_1, \ldots, m_p$ such that $x\theta = [m_1; \ldots; m_p]$. As θ is constructive, there exist a context prooftree Δ $t_1, \ldots, t_q \in Sat(T) \cup \mathcal{I}$ such that Δ is a proof of $\{t_1\theta, \ldots, t_q\theta\} \vdash x\theta$ using composition rules only. We can assume that for every $1 \leq j \leq q$, we have that $t_j \notin \mathcal{X}$. Consequently, for every $1 \leq i \leq p$, we have that $Sat(T)\theta \cup \mathcal{I} \vdash m_i$ by using composition rules only. By applying Lemma 4.2.2, $Sat(T)\theta' \cup \mathcal{I} \vdash m_i$ using composition rules only. So, if $x\theta' = x\theta$, $Sat(T)\theta' \cup \mathcal{I} \vdash x\theta'$ using composition rules only. Otherwise, $x_{j_0}\theta'$ is built with links of $x_{j_0}\theta$, and we still get that $Sat(T)\theta' \cup \mathcal{I} \vdash x_{j_0}\theta'$ using composition rules only.

Consequently, we have that $Sat(T)\theta' \cup \mathcal{I} \vdash x\theta'$ using composition rules only for every $T \overset{?}{\vdash} x \in \mathcal{C}$, so θ' is a constructive solution of $(\mathcal{C}, \mathcal{I})$.

In conclusion, θ' is a constructive solution of $(\mathcal{C}, \mathcal{I})$ and ϕ, smaller than θ. We conclude by contradiction. $\square$

The third step of the proof of Theorem 4.2.1 consists in showing that we can restrict our attention to solutions θ such that $x\theta$ is either a constant or a subterm of $\phi\theta$, by using Lemma 4.1.2. This lemma is a generic lemma that shows how any solution can be transformed by projecting some variables on constants. It will be reused in the next section.

Theorem 4.2.1. *Let $\mathcal{L}$ be a link-based recursive language. Let $(\mathcal{C}, \mathcal{I})$ be a constraint system and ϕ be an $\mathcal{L}$-language constraint associated to $(\mathcal{C}, \mathcal{I})$. Deciding whether $(\mathcal{C}, \mathcal{I})$ and ϕ has a solution is in NP.*

We want to decide whether $(\mathcal{C}, \mathcal{I})$ with constraint language ϕ has a solution. Our decision procedure works as follows:

Step 1. We guess a sequence of transformation rules in $\mathcal{S}$ from $(\mathcal{C}, \mathcal{I})$ to $(\mathcal{C}', \mathcal{I})$ where $(\mathcal{C}', \mathcal{I})$ is a constraint system in solved form. We have that:

$$(\mathcal{C}, \mathcal{I}) \rightsquigarrow^*_\sigma (\mathcal{C}', \mathcal{I}) \in \mathcal{S} \text{ with } (\mathcal{C}', \mathcal{I}) \text{ in solved form.}$$

Step 2. Assume that $\mathcal{L}$ is defined by w_0, w_1, and w_2. Let N_0 be a name of Base sort in $\mathcal{I}$, $S = names(\mathcal{C}, \phi, w_0, w_1, w_2)$ and let $N = (\#S + 1)^{\#var(\mathsf{m})}$.

- Guess the values of variables of sort Base in $\{N_0\} \cup S$.
- Guess the values of variables of sort Msg in the sublinks built over $\{N_0\} \cup S$.
- Guess the values of variables of sort List among lists of sublinks in $\{N_0\} \cup S$ of length at most $k_0 + (k_1 - k_2) \times N^{k_1}$.

This gives us a substitution θ', we check whether θ' is a solution of $(\mathcal{C}', \mathcal{I})$ and $\phi\sigma$.

Proof. Thanks to Theorem 2.3.1, there exists a solution θ of $(\mathcal{C}, \mathcal{I})$ and ϕ if, and only if, there exist a constraint system $(\mathcal{C}', \mathcal{I})$ in solved form and substitutions σ, θ' such that $(\mathcal{C}, \mathcal{I}) \rightsquigarrow^*_\sigma (\mathcal{C}', \mathcal{I})$ by a derivation in $\mathcal{S}$ and θ' is a *non-confusing* solution of $(\mathcal{C}', \mathcal{I})$ and $\phi\sigma$. Furthermore, the length of this derivation is polynomially bounded in the size of $\mathcal{C}$. We can guess such a derivation, and are now left to decide the existence of a non confusing solution to a constraint system in solved form. First, thanks to Proposition 2.4.1, we can actually consider constructive solutions only.

Thanks to Proposition 4.2.4, we can assume that if θ' is a constructive solution of $(\mathcal{C}', \mathcal{I})$ and $l_1 \overset{?}{\in} \mathcal{L} \wedge \ldots \wedge l_n \overset{?}{\in} \mathcal{L}$, then $\|\mathsf{tail}(l_i)\theta'\|_l \leq k_0 + (k_1 - k_2) \times N^{k_1}$ for every $i \in \{1, \ldots, n\}$. Then, thanks to Lemma 4.2.2, if $(\mathcal{C}', \mathcal{I})$ is a constraint system in solved form and θ' is a constructive solution of $(\mathcal{C}, \mathcal{I}) \wedge \phi\sigma$, then θ'' is a solution of $(\mathcal{C}', \mathcal{I})$ and $\phi\sigma$ where

- $x\theta'' = x\theta'$ if $x\theta' \in st(\phi\theta')$,
- $x\theta'' = []$ if $x\theta' \notin st(\phi\theta')$ and $x \in \mathcal{X}_{\mathsf{List}}$,
- $x\theta'' = N_0$ if $x\theta' \notin st(\phi\theta')$ and $x \notin \mathcal{X}_{\mathsf{List}}$

For $x \notin \mathcal{X}_{\mathsf{List}}$, either $x\theta'' \in st(\phi\theta'')$, which means that $x\theta''$ is a sublink, or $x\theta'' = N_0$. So we can guess which variables of $\mathcal{X}_{\mathsf{Msg}} \cup \mathcal{X}_{\mathsf{Base}}$ are instantiated by sublinks, and guess the sublinks. Instantiate the other variables in $\mathcal{X}_{\mathsf{Msg}} \cup \mathcal{X}_{\mathsf{Base}}$ with N_0. If $x \in \mathcal{X}_{\mathsf{List}}$ then either $x\theta'' = []$ or $x\theta'' \in st(\phi\theta'')$, and so it is a list of links of length at most $k_0 + (k_1 - k_2) \times N^{k_1}$.

□

4.3 Routing protocols

Routing protocols typically perform recursive checks to ensure the validity of a given route. However, link-based recursive languages do not suffice to express these checks. Indeed, in routing protocols, nodes aim at establishing and certifying a successful route (*i.e.* a list of names of nodes) between two given nodes that wish to communicate. Each node on the route typically contributes to the routing protocol by certifying that the proposed route is correct, to the best of its knowledge. Thus each contribution contains a list of names (the route). Then the final node receives a list of contributions and needs to check that each contribution contains the same list of names, which has also to be consistent with the whole received message. For example, in the case of the SMNDP protocol [FGML09], the source node has to check that the received message is of the form:

$$[[\![\langle D, S, l_{\mathsf{route}}\rangle]\!]_{\mathsf{sk}(A_n)}; \dots; [\![\langle D, S, l_{\mathsf{route}}\rangle]\!]_{\mathsf{sk}(A_1)}; [\![\langle D, S, l_{\mathsf{route}}\rangle]\!]_{\mathsf{sk}(D)}]$$

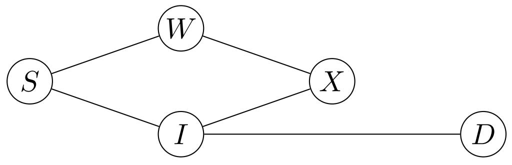
where $l_{route} = [D; A_1; \dots; A_n]$.

4.3.1 Example: the SMNDP protocol

The aim of the SMNDP protocol [FGML09] is to find a path from a source node S towards a destination node D. In the first phase of the protocol, nodes broadcast the route request to their neighbors, adding their name to the current path. When the request reaches the destination, D signs the route and sends the reply back over the network.

More formally, if D receives a request message $\langle \mathsf{req}, S, D, Id, l\rangle$, it computes signature $s_0 = [\![D, S, D :: l]\!]_{\mathsf{sk}(D)}$ and sends back the reply $\langle \mathsf{rep}, D, S, D :: l, [s_0]\rangle$. All nodes along the route have then to certify the route by adding their own signature. More precisely, during the reply phase, an intermediate node A_i receiving a message $\langle \mathsf{rep}, D, S, l_{\mathsf{route}}, [s_{i-1}, \dots, s_0]\rangle$ would compute the signature $s_i = [\![D, S, l_{\mathsf{route}}]\!]_{\mathsf{sk}(A_i)}$ and send the message $\langle \mathsf{rep}, D, S, l_{\mathsf{route}}, [s_i, \dots, s_0]\rangle$. The list of signatures expected by S built over the list $l_{\mathsf{route}} = [D, A_1, \dots, A_n]$ is the list $l_{sign} = [s_n, \dots, s_0]$ where $s_0 = [\![D, S, l_{\mathsf{route}}]\!]_{\mathsf{sk}(D)}$ and $s_i = [\![D, S, l_{\mathsf{route}}]\!]_{\mathsf{sk}(A_i)}$ for $1 \leq i \leq n$. We will denote by $\mathcal{L}_{\mathsf{SMNDP}}$ the set of messages of the form $\langle\langle S, D\rangle, \langle l_{\mathsf{route}}, l_{sign}\rangle\rangle$.

Consider the following network configuration, where S is the source node, D is the destination node, X is an intermediate (honest) node, W is a node who has been compromised (*i.e.* the intruder knows the secret key $\mathsf{sk}(W)$), and I is the malicious node.



An execution of the protocol where D is ready to answer a request and the source is ready to input the final message can be represented by the following constraint system:

$$\mathcal{C} = \begin{cases} T_0 \cup \{u_0, u_1\} & \Vdash & v_1 \\ T_0 \cup \{u_0, u_1, u_2\} & \Vdash & v_2 \end{cases}$$

with $T_0 = \{S, D, X, I, W, \mathsf{sk}(I), \mathsf{sk}(W)\}$ the initial knowledge of the intruder
$u_0 = \langle \mathsf{req}, S, D, Id, [] \rangle$,
$u_1 = \langle \mathsf{req}, S, D, Id, [X, W] \rangle$,
$u_2 = \langle \mathsf{rep}, D, S, D :: x_l, [[\![\langle D, S, D :: x_l \rangle]\!]_{\mathsf{sk}(D)}] \rangle$,
$v_1 = \langle \mathsf{req}, S, D, x_{id}, x_l \rangle$,
$v_2 = \langle \mathsf{rep}, D, S, D :: x_{\mathsf{route}}, x_{sign} \rangle$

Let $\mathcal{I}$ be a non-empty set of names such that $st(\mathcal{C}) \cap \mathcal{I} = \emptyset$. We have that $(\mathcal{C}, \mathcal{I})$ is a constraint system. A solution to $(\mathcal{C}, \mathcal{I}) \wedge \langle \langle S, D \rangle, \langle D :: x_{\mathsf{route}}, x_{sign} \rangle \rangle \overset{?}{\in} \mathcal{L}_{\mathsf{SMNDP}}$ is *e.g.* the substitution $\theta = \{x_{id} \mapsto Id, x_l \mapsto [I; W], x_{\mathsf{route}} \mapsto [I; W], x_{sign} \mapsto l_{sign}\}$ where:

- $l_{\mathsf{route}} = [D, I, W]$, and
- $l_{sign} = [[\![\langle D, S, l_{\mathsf{route}} \rangle]\!]_{\mathsf{sk}(W)}; [\![\langle D, S, l_{\mathsf{route}} \rangle]\!]_{\mathsf{sk}(I)}; [\![\langle D, S, l_{\mathsf{route}} \rangle]\!]_{\mathsf{sk}(D)}]$.

This solution reflects an attack (discovered in [AY07]) where the attacker sends to the destination node D the message $\langle \mathsf{req}, S, D, Id, l \rangle$ with a false list $l = [I, W]$. Then D answers accordingly by $\langle \mathsf{rep}, D, S, l_{\mathsf{route}}, [[\![\langle D, S, l_{\mathsf{route}} \rangle]\!]_{\mathsf{sk}(D)}] \rangle$. The intruder concludes the attack by sending to S the message $\langle \mathsf{rep}, D, S, l_{\mathsf{route}}, l_{sign} \rangle$. This yields S accepting W, I, D as a route to D, while it is not a valid route.

4.3.2 Definition of Mapping-based languages

An interesting property in the case of routing protocols is that (valid) messages are uniquely determined by the list of nodes $[A_1; \ldots; A_n]$ in addition to some parameters (*e.g.* the source and destination nodes in the case of SMNDP). We propose a generic definition that captures any such language based on a list of names.

Definition 4.3.1 (mapping-based language). *Let* b *be a term that contains no name and no* :: *symbol, and such that:*

$$\{w_1, w_1^p, \ldots, w_m^p\} \subseteq vars(\mathsf{b}) \subseteq \{w_1, w_2, w_3, w_1^p, \ldots, w_m^p\}.$$

The variables $w_1^p, \ldots, w_m^p$ *are the parameters of the language, whereas* w_1, w_2, *and* w_3 *are special variables. Let* $\mathcal{P} = \langle P_1, \ldots, P_m \rangle$ *be a tuple of names and* $\sigma_{\mathcal{P}} = \{w_1^p \mapsto P_1, \ldots, w_m^p \mapsto P_m\}$. *Let* $l = [A_1; \ldots; A_n]$ *be a list of names, the links are defined over* l *recursively in the following manner :*

$$\mathsf{m}_{\mathcal{P}}(i, l) = (\mathsf{b}\sigma_{\mathcal{P}})\{w_1 \mapsto l, w_2 \mapsto A_i, w_3 \mapsto [\mathsf{m}_{\mathcal{P}}(i-1, l); \ldots; \mathsf{m}_{\mathcal{P}}(1, l)]\}$$

The mapping-based *language (defined by* b*) is the following one:*

$$\mathcal{L} = \{\langle \mathcal{P}, \langle l, l' \rangle \rangle \mid \mathcal{P} = \langle P_1, \ldots, P_m \rangle \text{ is a tuple of names, } l = [A_1; \ldots; A_n] \text{ a list of names, } n \in \mathbb{N}, \text{ and } l' = [\mathsf{m}_{\mathcal{P}}(n, l); \ldots; \mathsf{m}_{\mathcal{P}}(1, l)]\}.$$

A mapping-based language is defined by a base shape b. The special variables w_2 and w_3 are optional and may not occur in b. Each element of the language is a triple $\langle \mathcal{P}, \langle l, l' \rangle \rangle$ where l' is a list of links entirely determined by the tuple $\mathcal{P} = \langle P_1, \ldots, P_m \rangle$ and the list l of arbitrary length n. In the list l', each link contains the same parameters $P_1, \ldots, P_m$ (*e.g.* the source and destination nodes), the list l of n names $[A_1; \ldots; A_n]$ and possibly the current name A_i and the list of previous links, following the base shape b.
We illustrate this definition with two examples of routing protocols.

Example 4.3.1 (SMNDP protocol [FGML09]). *Recall that in SMNDP, the list of signatures expected by the source node S built over the list $l = [A_1, \ldots, A_n]$ is the list $[s_n, \ldots, s_1]$, where $s_i = [\![\langle D, S, l\rangle]\!]_{\mathsf{sk}(A_i)}$. This language has two parameters, the name of the source w_1^p and the name of the destination w_2^p. The language can be formally described with $\mathsf{b} = [\![\langle w_2^p, w_1^p, w_1\rangle]\!]_{\mathsf{sk}(w_2)}$.*

Example 4.3.2 (endairA protocol [BV04]). *The difference between SMNDP and endairA lies in the fact that during the reply phase, the intermediate nodes compute a signature over the partial signature list that they receive. In the endairA protocol, the list of signatures expected by the source node S built over the list of nodes $l = [A_1, \ldots, A_n]$ is the list $l'_s = [s_n, \ldots, s_1]$, where $s_i = [\![\langle D, S, l, [s_{i-1}; \ldots; s_1]\rangle]\!]_{\mathsf{sk}(A_i)}$.*

This language has two parameters, the name of the source w_1^p and the name of the destination w_2^p. The language can be formally described with $\mathsf{b} = [\![\langle w_2^p, w_1^p, w_1, w_3\rangle]\!]_{\mathsf{sk}(w_2)}$.

4.3.3 Decision procedure

We propose a procedure for checking for secrecy preservation for a protocol with mapping-based tests in NP, for a bounded number of sessions.

The goal of this section is to prove that checking for secrecy preservation for a protocol with mapping-based recursive tests is NP, for a bounded number of sessions (Theorem 4.3.4). To achieve this goal, we will show that we can bound in advance the length of the recursive lists.

Let u be a term. We denote by $\|u\|_{dag}$ the size of u in DAG representation (*i.e.* number of distinct subterms in u).

Let t be a term such that $vars(t) = \{w_1, w_2, w_3\}$, the variables of t are the special variables in the definition of b, and let v_1, v_2, v_3 be ground terms. For the sake of clarity, we will write $t\lfloor v_1, v_2, v_3\rfloor$ for $t\{w_1 \mapsto v_1, w_2 \mapsto v_2, w_3 \mapsto v_3\}$.

In a mapping-based language, the links contain enough information to define precisely the language to which they belong.

Lemma 4.3.1. *Let θ be a solution of $\phi = u_1 \overset{?}{\in} \mathcal{L} \wedge \cdots \wedge u_p \overset{?}{\in} \mathcal{L}$ where $u_j = \langle \mathcal{P}_j, \langle l_j, l_j\rangle\rangle$, non-confusing with respect to $st(\phi)$,* i.e. *such that $t_1\theta = t_2\theta$ implies $t_1 = t_2$ for any term $t_1, t_2 \in st(\phi)$. Let $1 \le i, j \le n$. If $l'_i\theta$ and $l'_j\theta$ share a link, i.e. $\mathsf{m}_{\mathcal{P}_i\theta}(i', l_i\theta) = \mathsf{m}_{\mathcal{P}_j\theta}(j', l_j\theta)$ for some i', j', then $u_i = u_j$.*

Proof. Indeed, suppose that there exist i', j' such that $\mathsf{m}_{\mathcal{P}_i\theta}(i', l_i\theta) = \mathsf{m}_{\mathcal{P}_j\theta}(j', l_j\theta)$. Write $l_i\theta = [a_1; \ldots; a_p]$ and $l_j\theta = [c_1; \ldots; c_q]$. Write $\mathcal{P}_i\theta = \langle p_1^i, \ldots, p_m^i\rangle$ and $\mathcal{P}_j\theta = \langle p_1^j, \ldots, p_m^j\rangle$. Recall that

$$\begin{aligned} \mathsf{m}_{\mathcal{P}_i\theta}(i', l_i\theta) &= \mathsf{b}_i\lfloor l_i\theta, a_{i'}, [\mathsf{m}_{\mathcal{P}_i\theta}(i'-1, l_i\theta); \ldots; \mathsf{m}_{\mathcal{P}_i\theta}(1, l_i\theta)]\rfloor \\ \mathsf{m}_{\mathcal{P}_j\theta}(j', l_j\theta) &= \mathsf{b}_j\lfloor l_j\theta, c_{j'}, [\mathsf{m}_{\mathcal{P}_j\theta}(j'-1, l_j\theta); \ldots; \mathsf{m}_{\mathcal{P}_j\theta}(1, l_j\theta)]\rfloor \end{aligned}$$

where $\mathsf{b}_i = \mathsf{b}\{w_1^p \mapsto p_1^i, \ldots, w_m^p \mapsto p_m^i\}$, $\mathsf{b}_j = \mathsf{b}\{w_1^p \mapsto p_1^j, \ldots, w_m^p \mapsto p_m^j\}$, and $a_{i'}, b_{j'}$, $[\mathsf{m}_{\mathcal{P}_i}(i'-1, l_i\theta); \ldots; \mathsf{m}_{\mathcal{P}_i}(1, l_i\theta)]$, $[\mathsf{m}_{\mathcal{P}_j}(j'-1, l_j\theta); \ldots; \mathsf{m}_{\mathcal{P}_j}(1, l_j\theta)]$ are optional parameters. As $\mathsf{m}_{\mathcal{P}_i\theta}(i', l_i\theta) = \mathsf{m}_{\mathcal{P}_j\theta}(j', l_j\theta)$, we have that

$$\begin{gathered} \mathsf{b}_i\lfloor l_i\theta, a_{i'}, [\mathsf{m}_{\mathcal{P}_i\theta}(i'-1, l_i\theta); \ldots; \mathsf{m}_{\mathcal{P}_i\theta}(1, l_i\theta)]\rfloor \\ = \\ \mathsf{b}_j\lfloor l_j\theta, b_{j'}, [\mathsf{m}_{\mathcal{P}_j\theta}(j'-1, l_j\theta); \ldots; \mathsf{m}_{\mathcal{P}_j\theta}(1, l_j\theta)]\rfloor \end{gathered}$$

We deduce that $l_i\theta = l_j\theta$, and also $p^i_k = p^j_k$ for all $1 \leq k \leq m$ (these parameters are compulsory), *i.e.* $\mathcal{P}_i\theta = \mathcal{P}_j\theta$. It follows that $l'_i\theta = l'_j\theta$, hence $u_i\theta = u_j\theta$. As θ is non-confusing w.r.t. $st(\phi)$, it follows that $u_i = u_j$. □

We prove in Proposition 4.3.3 that we can consider only small constructive solutions. Intuitively, a constraint is of the form $\langle\langle p^j_1, \dots, p^j_m\rangle, \langle \ell_j, \ell'_j\rangle\rangle \in \mathcal{L}$, where the beginning of ℓ_j constrains the end of ℓ'_j and reciprocally, so we can cut somewhere in the middle the large solutions.

Proposition 4.3.3. *Let $\mathcal{L}$ be a mapping-based language. Let $(\mathcal{C}, \mathcal{I})$ be a deduction constraint system in solved form, ψ be an $\mathcal{L}$-language constraint associated to $(\mathcal{C}, \mathcal{I})$, and τ be a constructive solution of $(\mathcal{C}, \mathcal{I})$ and ψ. We further assume that ψ is of the form $u_1 \overset{?}{\in} \mathcal{L} \wedge \dots \wedge u_p \overset{?}{\in} \mathcal{L}$ where $u_j = \langle\langle p^j_1, \dots, p^j_m\rangle, \langle \ell_j, \ell'_j\rangle\rangle$.*

Let $M = \#st(\mathcal{C}, \psi) + \max_{1 \leq j \leq p} \|\ell'_j\|_l + 2 \times \#var(\mathcal{C}) \times \max_{t \in st(\mathcal{C},\psi)} \|t\|_{dag}$. There exists a constructive solution τ' of $(\mathcal{C}, \mathcal{I})$ and ψ such that, for every j, we have that $\|\mathsf{tail}(\ell_j)\tau'\|_l \leq M$.

Proof. Consider a smallest constructive solution τ of $(\mathcal{C}, \mathcal{I})$ and ψ, where the size of a solution is given by:

$$|\tau| = \sum_{1 \leq j \leq p} \|\mathsf{tail}(\ell_j)\tau\|_l$$

Either $\|\mathsf{tail}(\ell_j)\tau\|_l \leq M$ for all j and we conclude. Otherwise, there exists j_0 such that $\|\mathsf{tail}(\ell_{j_0})\tau\|_l > M$. In that case, we show that we can build τ' smaller than τ, a constructive solution of $(\mathcal{C}, \mathcal{I})$ and ψ, which is in contradiction with τ smallest solution, and we conclude.

We wish to write $\tau = \theta \circ \sigma$ with θ non-confusing w.r.t. $st(\mathcal{C}, \psi)$, i.e. such that $t_1\theta = t_2\theta$ implies $t_1 = t_2$ for any $t_1, t_2 \in st(\mathcal{C}, \psi)$. We define

$$\sigma = mgu\{t_1 = t_2 \mid t_1\tau = t_2\tau, t_1 \neq t_2, t_1, t_2 \in st(\mathcal{C}, \psi)\}.$$

We have that $st(\mathcal{C}\sigma, \psi\sigma) \subseteq st(\mathcal{C}, \psi)\sigma$, thanks to Lemma **??**, as σ is an mgu of terms in $st(\mathcal{C}, \psi)$. Furthermore, by Definition, $st(\psi) \cap \mathcal{I} = \emptyset$ and $st(\mathcal{C}) \cap \mathcal{I} = \emptyset$, so we deduce that $(\mathcal{C}\sigma, \mathcal{I})$ is a constraint system and $\psi\sigma$ is an $\mathcal{L}$-language constraint associated with $(\mathcal{C}\sigma, \mathcal{I})$. Lastly, since σ is more general than τ, there is a substitution θ such that $\tau = \theta \circ \sigma$ and θ is a solution of $(\mathcal{C}\sigma, \mathcal{I})$ and $\psi\sigma$.

We now show that θ is non-confusing w.r.t. $st(\mathcal{C}\sigma, \psi\sigma)$. Let $t_1, t_2 \in st(\mathcal{C}\sigma, \psi\sigma)$ be two terms such that $t_1\theta = t_2\theta$. We apply Lemma **??**: there are two terms $u_1, u_2 \in st(\mathcal{C}, \psi)$ such that $t_1 = u_1\sigma$ and $t_2 = u_2\sigma$. As $t_1\theta = t_2\theta$, we deduce that $u_1\tau = u_2\tau$. It follows that $u_1, u_2 \in st(\mathcal{C}, \psi)$ with $u_1\tau = u_2\tau$. Either $u_1 \neq u_2$, and thus by Definition of σ, we have that $u_1\sigma = u_2\sigma$, or $u_1 = u_2$. In both cases, we deduce that $t_1 = t_2$, so θ is non-confusing w.r.t. $st(\mathcal{C}\sigma, \psi\sigma)$.

For every j, $l_j = \ell_j\sigma$, $l'_j = \ell'_j\sigma$, $u'_j = u_j\sigma$ and $\phi = \psi\sigma$, *i.e.*

$$\phi = u'_1 \overset{?}{\in} \mathcal{L} \wedge \dots \wedge u'_p \overset{?}{\in} \mathcal{L} \text{ with } u'_j = \langle\langle p^j_1\sigma, \dots, p^j_m\sigma\rangle, \langle l_j, l'_j\rangle\rangle.$$

We can assume that the elements of ϕ are distinct, *i.e.* for every i, j, if $u'_i = u'_j$ then $i = j$. Furthermore, let $\mathcal{P}_j = \langle p^j_1\tau, \dots, p^j_m\tau\rangle$. Lemma 4.3.1 then allows us to express the following statement:

$$\text{if } \mathsf{m}_{\mathcal{P}_j}(i, l_j\theta) = \mathsf{m}_{\mathcal{P}_{j'}}(i', l_{j'}\theta) \text{ for some } i,\ i', \text{ then } u_j = u_{j'}, \text{ and thus } j = j'.$$

We show that $\|\mathsf{tail}(l_{j_0})\theta\|_l > M'$ *where* $M' = \#st(\mathcal{C}\sigma, \phi) + \max_{1\leq j\leq p} \|l'_j\|_l$.

We have chosen j_0 such that $\|\mathsf{tail}(\ell_{j_0})\tau\|_l > M$. Let $M_1 = \#var(\mathcal{C}) \times \max_{t\in st(\mathcal{C},\psi)} \|t\|_{dag}$. Thanks to Lemma **??**, for every $x \in var(\mathcal{C})$, $\|x\sigma\|_{dag} \leq M_1$.

We first show that $M > M' + M_1$. We have that $l'_j = \ell'_j\sigma$, so $\|l'_j\|_l = \|\ell'_j\|_l + \|\mathsf{tail}(\ell'_j)\sigma\|_l$. Furthermore, it is clear that $\|u :: l\|_{dag} \geq \|l\|_{dag} + 1$, and thus for every l, $\|l\|_{dag} \geq \|l\|_l$. Consequently, for every j, we have that:

$$\begin{array}{rcl} \max_{1\leq j\leq p} \|\ell'_j\|_l & \geq & \|\ell'_j\|_l \\ & \geq & \|l'_j\|_l - \|\mathsf{tail}(\ell'_j)\sigma\|_l \\ & \geq & \|l'_j\|_l - \|\mathsf{tail}(\ell'_j)\sigma\|_{dag} \\ & \geq & \|l'_j\|_l - M_1 \end{array}$$

As this inequality is true for every j, it follows that

$$\max_{1\leq j\leq p} \|\ell'_j\|_l \geq \max_{1\leq j\leq p} \|l'_j\|_l - M_1$$

Furthermore, we have $\#st(\mathcal{C}\sigma, \phi) = \#st(\mathcal{C}, \psi)$. We deduce that $M = \max_{1\leq j\leq p} \|\ell'_j\|_l + \#st(\mathcal{C}, \psi) + 2M_1 = \max_{1\leq j\leq p} \|\ell'_j\|_l + \#st(\mathcal{C}\sigma, \phi) + 2M_1$, and so $M \geq \max_{1\leq j\leq p} \|l'_j\|_l + \#st(\mathcal{C}\sigma, \phi) + M_1 = M' + M_1$.

We now conclude by showing that $\|\mathsf{tail}(l_{j_0})\theta\|_l > M'$. We have that $l_{j_0} = \ell_{j_0}\sigma$, so $\|\mathsf{tail}(\ell_{j_0})\tau\|_l = \|\mathsf{tail}(l_{j_0})\theta\|_l + \|\mathsf{tail}(\ell_{j_0})\sigma\|_l$. Consequently, we have that:

$$\begin{array}{rcl} \|\mathsf{tail}(l_{j_0})\theta\|_l & = & \|\mathsf{tail}(\ell_{j_0})\tau\|_l - \|\mathsf{tail}(\ell_{j_0})\sigma\|_l \\ & > & M - \|\mathsf{tail}(\ell_{j_0})\sigma\|_{dag} \\ & \geq & M - M_1 \geq M'. \end{array}$$

We build θ' *smaller than* θ *and we define* $\tau' = \theta' \circ \sigma$

Let j_1 be such that $\|\mathsf{tail}(l_{j_1})\theta\|_l \geq \|\mathsf{tail}(l_j)\theta\|_l$ for every $j \leq p$. Necessarily, $\mathsf{tail}(l_{j_1}) = x_1$ for some $x_1 \in \mathcal{X}_{\mathsf{List}}$ and $\|x_1\theta\|_l > M'$. By reordering the constraints, we can assume that there exists $1 \leq q \leq p$ such that

- for every $1 \leq j \leq q$, we have that $\mathsf{tail}(l_j) = x_1$, and
- for $q < j \leq p$, we have that $\mathsf{tail}(l_j) \neq x_1$ (and thus, $\|\mathsf{tail}(l_j)\theta\|_l \leq \|x_1\theta\|_l$).

We want to change the value of $x_1\theta$, while preserving language memberships.

For each constraint $\langle\langle p_1^j\sigma, \ldots, p_m^j\sigma\rangle, \langle l_j, l'_j\rangle\rangle \overset{?}{\in} \mathcal{L}$, l_j provides constraints on the last elements of the list $l'_j\theta$, while l'_j provides constraints on the last elements of the list $l_j\theta$. For $1 \leq j \leq q$, l_j thus constrains the last elements of $x_1\theta$. We have to keep those elements to preserve language membership.

For $j \leq q$, we write:

$$\begin{array}{ll} l_j = c^j :: \ldots :: c^j_{k_j} :: x_1 & \\ l_j\theta = c^j_1\theta :: \ldots :: c^j_{k_j}\theta :: x_1\theta = [a^j_1; \ldots; a^j_{n_j}] & (x_1\theta = [a^j_{k_j+1}; \ldots; a^j_{n_j}]) \end{array}$$

$$\begin{array}{ll} l'_j = v^j_1 :: \ldots :: v^j_{k'_j} :: y_j & \\ l'_j\theta = v^j_1\theta :: \ldots :: v^j_{k'_j}\theta :: y_j\theta = [m^j_{n_j}; \ldots; m^j_1] & (y_j\theta = [m^j_{n_j-k'_j}; \ldots; m^j_1]) \end{array}$$

Let $k_s = \max_{1\leq j\leq q} k'_j$. The k_s last elements of $x_1\theta$ are determined, and thus the k_s first elements of each l'_j are also determined.
For every $j \leq m$, let $t_j = [m^j_{n_j-k_s}; \dots; m^j_1]$. t_j represents the part of $l_j\theta$ which is not constrained by the k_s last elements of $x_1\theta$ and that can be modified while preserving language membership constraints. For every $1 \leq j \leq q$, we have that $\|t_j\|_l + k_s = \|l'_j\theta\|_l = \|l_j\theta\|_l = \|x_1\theta\|_l + k_j$.
As $m^j_i = \mathsf{m}_{\mathcal{P}_j}(i, l_j\theta)$, we obtain thanks to statement $(*)$ that if $m^j_i = m^{j'}_{i'}$, then $j = j'$. Hence, the sublists of $t_1, \dots, t_m$ are distinct. We know that $\|x_1\theta\|_l > M'$, and thus

$$\|t_j\|_l - k_j = \|x_1\theta\|_l - k_s > \#st(\mathcal{C}\sigma, \phi).$$

As a consequence, there exists k_v such that $k_s \leq k_v \leq \#st(\mathcal{C}\sigma,\phi) + k_s$ and for all $1 \leq j \leq q$, $[m^j_{n_j-k_v}; \dots; m^j_1] \notin (st(\mathcal{C}\sigma, \phi))\theta$.
We define $w_0 = [a^1_{n_1-k_v+1}, \dots, a^1_{n_1}]$ as the last k_v elements of $x_1\theta$. Let $\delta_0 = \{x_1\theta \mapsto w_0\}$ be the associated swapping.
Define, for $1 \leq j \leq q$,

$$\begin{cases} w_j = [m^j_{n_j-k_v}; \dots; m^j_1] \\ w'_j = [m^j_{k_j}; \dots; m^j_1]\delta_0 \end{cases}$$

Note that $w_j \notin (st(\mathcal{C}\sigma, \phi))\theta$ since we chose k_v to ensure this. Intuitively, w'_j represents the part of $l'_j\theta$ that is constrained by l_j and thus has to be kept in our new solution. Let $\delta_j = \{w_j \mapsto w'_j, x_1\theta \mapsto w_0\}$ for $1 \leq j \leq q$. For $j > q$, we define $\delta_j = id$.
Let $\delta = \{x_1\theta \mapsto w_0, w_1 \mapsto w'_1, \dots, w_q \mapsto w'_q\}$. We choose $\theta' = \theta\delta$, and $\tau' = \theta' \circ \sigma$.
Note that $\|x_1\theta'\|_l = k_v \leq M' < \|x_1\theta\|_l$. By definition, $x_1 = \mathsf{tail}(\ell_{j_1}\sigma)$, so we have that $\|\mathsf{tail}(\ell_{j_1})\tau'\|_l < \|\mathsf{tail}(\ell_{j_1})\tau\|_l$. Furthermore, for every x, $\|x\tau'\|_l \leq \|x\tau\|_l$, since for every term u, $\|u\delta\|_l \leq \|u\|_l$. We deduce that $|\tau'| < |\tau|$.
We have thus built a substitution $\tau' = \theta' \circ \sigma$ smaller than τ. It remains to show that τ' is a constructive solution of $(\mathcal{C}, \mathcal{I})$ and ψ. We will first show that θ' is a solution of ϕ and then that τ' is a constructive solution of $(\mathcal{C}, \mathcal{I})$. We first have two claims that will help us with the proof.

Claim A: *$(t\theta)\delta = t(\theta\delta)$ for $t \in st(\mathcal{C}\sigma, \phi)$.*
We show by induction on t that for every term $t \in st(\mathcal{C}\sigma, \phi)$, $(t\theta)\delta = t(\theta\delta)$.

- if $t \in \mathcal{X}$, then the result is straightforward, since for every $x \in \mathcal{X}$, $(x\theta)\delta = x(\theta\delta)$
- if $t = f(t_1, \dots, t_k)$, we reason by case distinction over the value of $f(t_1, \dots, t_k)\theta$:
  - If $f(t_1, \dots, t_k)\theta = x_1\theta$, as θ is non-confusing with respect to $st(\mathcal{C}\sigma, \phi)$, then $x_1 = f(t_1, \dots, t_k)$, and this is in contradiction with $x \in \mathcal{X}$.
  - If there exists i such that $f(t_1, \dots, t_k)\theta = w_i$, then $w_i \in st(\mathcal{C}\sigma, \phi)\theta$, which yields a contradiction.
  - We are thus in a case where $f(t_1\theta, \dots, t_k\theta)\delta = f((t_1\theta)\delta, \dots, (t_k\theta)\delta)$. By induction hypothesis, $(t_i\theta)\delta = t_i(\theta\delta)$, and so

$$\begin{array}{rcl} (t\theta)\delta & = & (f(t_1, \dots, t_k)\theta)\delta \\ & = & f((t_1\theta)\delta, \dots, (t_k\theta)\delta) \\ & = & f(t_1(\theta\delta), \dots, t_k(\theta\delta)) \\ & = & f(t_1, \dots, t_k)(\theta\delta) \\ & = & t(\theta\delta) \end{array}$$

Claim B: *for* $1 \leq j \leq p$ *and* $1 \leq i \leq n_j$, $\mathsf{m}_{\mathcal{P}_j}(i, l_j\theta)\delta = \mathsf{m}_{\mathcal{P}_j}(i, l_j\theta)\delta_j$ *and* $[\mathsf{m}_{\mathcal{P}_j}(i, l_j\theta); \ldots; \mathsf{m}_{\mathcal{P}_j}(1, l_j\theta)]\delta = [\mathsf{m}_{\mathcal{P}_j}(i, l_j\theta); \ldots; \mathsf{m}_{\mathcal{P}_j}(1, l_j\theta)]\delta_j$

First, we show that for every $1 \leq j \leq p$, $(l_j\theta)\delta = (l_j\theta)\delta_0 = (l_j\theta)\delta_j$. We have that $l_j\theta = [a_1^j; \ldots; a_{n_j}^j]$ with $a_k^j \in \mathcal{N}$. For every j', k', $a_k^j \neq m_{k'}^{j'}$, so $(l_j\theta)\delta = (l_j\theta)\delta_0$.

For $1 \leq j \leq q$, by a similar reasoning we also get that $(l_j\theta)\delta_j = (l_j\theta)\delta_0$.

For $q < j \leq p$, $l_j = c_1^j :: \ldots :: c_{k_j}^j :: \mathsf{tail}(l_j)$ with $\|\mathsf{tail}(l_j)\theta\|_l \leq \|x_1\theta\|_l$. By applying the definition of a swapping, and as θ is non-confusing,

$$(l_j\theta)\delta_0 = (c_1^j\theta)\delta_0 :: \ldots :: (c_{k_j}^j\theta)\delta_0 :: (\mathsf{tail}(l_j)\theta)\delta_0.$$

Furthermore, $(\mathsf{tail}(l_j)\theta)\delta_0 = \mathsf{tail}(l_j)\theta$ as $\|\mathsf{tail}(l_j)\theta\|_l \leq \|x_1\theta\|_l$ (and $\mathsf{tail}(l_j)\theta \neq x_1\theta$ as $\mathsf{tail}(l_j) \neq x_1$ and θ is non-confusing). We also have that for every $1 \leq k \leq k_j$, $(c_k^j\theta)\delta_0 = c_k^j\theta$ since $c_k^j\theta$ is a name. Consequently, $(l_j\theta)\delta_0 = (l_j\theta) = (l_j\theta)\delta_j$ since $\delta_j = id$ for $q < j \leq p$.

To sum up, for every $1 \leq j \leq p$, we have that $(l_j\theta)\delta = (l_j\theta)\delta_0 = (l_j\theta)\delta_j$.

Let $\mathsf{b}_j = \mathsf{b}\{w_1^p \mapsto p_1^j\tau, \ldots, w_m^p \mapsto p_m^j\tau\}$. We show by induction on i that

- $\mathsf{m}_{\mathcal{P}_j}(i, l_j\theta)\delta = \mathsf{m}_{\mathcal{P}_j}(i, l_j\theta)\delta_j$, and
- $[\mathsf{m}_{\mathcal{P}_j}(i, l_j\theta); \ldots; \mathsf{m}_{\mathcal{P}_j}(1, l_j\theta)]\delta = [\mathsf{m}_{\mathcal{P}_j}(i, l_j\theta); \ldots; \mathsf{m}_{\mathcal{P}_j}(1, l_j\theta)]\delta_j$.

- If $i = 1$, we have that $\mathsf{m}_{\mathcal{P}_j}(1, l_j\theta) = \mathsf{b}_j\lfloor l_j\theta, a_1^j, []\rfloor$.

  We apply the swapping δ (resp. δ_j) on $\mathsf{m}_{\mathcal{P}_j}(1, l_j\theta)$. As b_j is a term which does not contain the list constructor and δ is a swapping of non-empty lists, we have that:

$$\begin{array}{rcl} \mathsf{m}_{\mathcal{P}_j}(1, l_j\theta)\delta & = & \mathsf{b}_j\lfloor (l_j\theta)\delta, a_1^j\delta, []\rfloor \\ \mathsf{m}_{\mathcal{P}_j}(1, l_j\theta)\delta_j & = & \mathsf{b}_j\lfloor (l_j\theta)\delta_j, a_1^j\delta_j, []\rfloor. \end{array}$$

  We have shown previously that $(l_j\theta)\delta = (l_j\theta)\delta_j$. Furthermore, since a_i^j is a name, we have that $a_1^j\delta = a_1^j = a_1^j\delta_j$, and so we deduce that:

$$\mathsf{m}_{\mathcal{P}_j}(1, l_j\theta)\delta = \mathsf{m}_{\mathcal{P}_j}(1, l_j\theta)\delta_j.$$

  We can have $\mathsf{m}_{\mathcal{P}_j}(1, l_j\theta) = \mathsf{m}_{\mathcal{P}_{j'}}(i, l_{j'}\theta)$ only if $j = j'$ (*cf.* statement $(*)$), so for any $j \neq j'$, $w_{j'} \notin st([\mathsf{m}_{\mathcal{P}_j}(1, l_j\theta)])$, and $[\mathsf{m}_{\mathcal{P}_j}(1, l_j\theta)]\delta = [\mathsf{m}_{\mathcal{P}_j}(1, l_j\theta)]\delta_j$.

- If $i > 1$, we have that

$$\mathsf{m}_{\mathcal{P}_j}(i, l_j\theta) = \mathsf{b}_j\lfloor l_j\theta, a_i^j, [\mathsf{m}_{\mathcal{P}_j}(i-1, l_j\theta); \ldots; \mathsf{m}_{\mathcal{P}_j}(1, l_j\theta)]\rfloor.$$

  We apply the swapping δ (resp. δ_j) on $\mathsf{m}_{\mathcal{P}_j}(1, l_j\theta)$. As b_j is a term which does not contain the list constructor and δ is a swapping of non-empty lists, we have that:

$$\begin{array}{rcl} \mathsf{m}_{\mathcal{P}_j}(i, l_j\theta)\delta & = & \mathsf{b}_j\lfloor (l_j\theta)\delta, a_i^j\delta, [\mathsf{m}_{\mathcal{P}_j}(i-1, l_j\theta); \ldots; \mathsf{m}_{\mathcal{P}_j}(1, l_j\theta)]\delta\rfloor \\ \mathsf{m}_{\mathcal{P}_j}(i, l_j\theta)\delta_j & = & \mathsf{b}_j\lfloor (l_j\theta)\delta_j, a_i^j\delta_j, [\mathsf{m}_{\mathcal{P}_j}(i-1, l_j\theta); \ldots; \mathsf{m}_{\mathcal{P}_j}(1, l_j\theta)]\delta_j\rfloor. \end{array}$$

We have shown previously that $(l_j\theta)\delta = (l_j\theta)\delta_j$. Furthermore, since a_i^j is a name, we have that $a_1^j\delta = a_1^j = a_1^j\delta_j$. Lastly, thanks to our induction hypothesis, we have that

$$[\mathsf{m}_{\mathcal{P}_j\theta}(i-1, l_j\theta); \ldots ; \mathsf{m}_{\mathcal{P}_j\theta}(1, l_j\theta)]\delta = [\mathsf{m}_{\mathcal{P}_j\theta}(i-1, l_j\theta); \ldots ; \mathsf{m}_{\mathcal{P}_j\theta}(1, l_j\theta)]\delta_j.$$

This allows us to conclude that $\mathsf{m}_{\mathcal{P}_j}(i, l_j\theta)\delta = \mathsf{m}_{\mathcal{P}_j}(i, l_j\theta)\delta_j$.

We can have $\mathsf{m}_{\mathcal{P}_j}(i, l_j\theta) = \mathsf{m}_{\mathcal{P}_{j'}}(i', l_{j'}\theta)$ only if $j = j'$ by $(*)$. Consequently, for any $j \neq j'$, we deduce that $v_{j'} \notin st([\mathsf{m}_{\mathcal{P}_j}(i, l_j\theta); \ldots ; \mathsf{m}_{\mathcal{P}_j}(1, l_j\theta)])$, and so $[\mathsf{m}_{\mathcal{P}_j}(i, l_j\theta); \ldots ; \mathsf{m}_{\mathcal{P}_j}(1, l_j\theta)]\delta = [\mathsf{m}_{\mathcal{P}_j}(i, l_j\theta); \ldots ; \mathsf{m}_{\mathcal{P}_j}(1, l_j\theta)]\delta_j$.

Show that θ' is a solution of ϕ. First, as $\langle p_1^j\sigma, \ldots, p_m^j\sigma\rangle\theta$ is a tuple of names, we have that $\langle p_1^j\sigma, \ldots, p_m^j\sigma\rangle\theta = \langle p_1^j\sigma, \ldots, p_m^j\sigma\rangle\theta'$. We will write

$$\mathcal{P}_j = \langle p_1^j\tau, \ldots, p_m^j\tau\rangle = \langle p_1^j\tau', \ldots, p_m^j\tau'\rangle.$$

We show that for every $1 \leq j \leq p$, $\langle \mathcal{P}_j, \langle l_j\theta', l'_j\theta'\rangle\rangle \in \mathcal{L}$ by distinguishing between cases:

- First, show that if $1 \leq j \leq q$ ($\mathsf{tail}(l_j) = x_1$), $\langle \mathcal{P}_j, \langle l_j\theta', l'_j\theta'\rangle\rangle \in \mathcal{L}$.

$$\begin{array}{ll} l_j\theta' = c_1\theta' :: \ldots :: c_{k_j}\theta' :: v_0 = [b_1; \ldots ; b_k] & (v_0 = [b_{k_j+1}; \ldots ; b_k]) \\ l'_j\theta' = v_1\theta' :: \ldots :: v_{k'_j}\theta' :: y_j\theta' = [r_k; \ldots ; r_1] & (y_j\theta' = [r_{k-k'_j}; \ldots ; r_1]) \end{array}$$

  where $k = k_j + k_v$. (for more clarity, as we only consider the j-th constraint, we will write $c_i = c_i^j$, $v_i = v_i^j, m_i = m_i^j$, $a_i = a_i^j$ and $n = n_j$).

  For $i \leq k_j$, it is clear that $b_i = a_i$, and for $i > k_j$, $b_i = a_{n-k+i}$.

  We have that $l'_j\theta = [m_n; \ldots ; m_1]$. By Claim A, $l'_j\theta' = (l'_j\theta)\delta$, and by Claim B, $(l'_j\theta)\delta = (l'_j\theta)\delta_j$. We compute $l'_j\theta'$:

$$\begin{array}{rcl} l'_j\theta' & = & [m_n; \ldots ; m_1]\delta_j \\ & = & (m_n :: \ldots :: m_{n-k_v+1} ::)\delta_j \\ & = & [m_n\delta_j; \ldots ; m_{n-k_v+1}\delta_j; m_{k_j}\delta_j; \ldots ; m_1\delta_j] \end{array}$$

  As $l'_j\theta' = [r_k; \ldots ; r_1]$, we deduce the values of r_i depending on i:

$$\begin{cases} \text{For } i \leq k_j, & r_i = m_i\delta_j \\ \text{For } i > k_j, & r_i = m_{n-k+i}\delta_j \end{cases}$$

  By definition of $\mathcal{L}$, $\langle \mathcal{P}_j, \langle l_j\theta', l'\rangle\rangle \in \mathcal{L}$ iff $l' = [\mathsf{m}_{\mathcal{P}_j}(k, l_j\theta'); \ldots ; \mathsf{m}_{\mathcal{P}_j}(1, l_j\theta')]$ where $\mathsf{m}_{\mathcal{P}_j}(i, l_j\theta') = \mathsf{b}_j\lfloor l_j\theta', b_i, [\mathsf{m}_{\mathcal{P}_j}(i-1, l_j\theta'); \ldots ; \mathsf{m}_{\mathcal{P}_j}(1, l_j\theta')]\rfloor$ and $\mathsf{b}_j = \mathsf{b}\{w_1^p \mapsto p_1^j\tau', \ldots, w_m^p \mapsto p_m^j\tau'\}$. We show by induction over i that $r_i = \mathsf{m}_{\mathcal{P}_j}(i, l_j\theta')$:

  - if $i \leq k_j$, $r_i = m_i\delta_j$.
    As $\langle \mathcal{P}_j, \langle l_j\theta, l'_j\theta\rangle\rangle \in \mathcal{L}$, $m_i = \mathsf{b}_j\lfloor l_j\theta, c_i\theta, [m_{i-1}; \ldots ; m_1]\rfloor$. As b_j does not contain the list constructor :: and δ_j is a swapping of (non empty) lists, by definition of a swapping, we have that

$$\mathsf{b}_j\lfloor l_j\theta, c_i\theta, [m_{i-1}; \ldots ; m_1]\rfloor\delta_j = \mathsf{b}_j\lfloor (l_j\theta)\delta_j, (c_i\theta)\delta_j, [m_{i-1}; \ldots ; m_1]\delta_j\rfloor$$

We have that $(c_i\theta)\delta = c_i\theta = a_i b_i$ and $(l_j\theta)\delta_j = (l_j\theta)\delta = l_j\theta'$, thanks to Claims (A) and (B). Furthermore, $[m_{i-1}; \dots; m_1]$ is a strict subterm of w_j, so

$$[m_{i-1}; \dots; m_1]\delta_j = [m_{i-1}\delta_j; \dots; m_1\delta_j].$$

Recall that $m_h\delta_j = r_h$ for $h \leq i-1 \leq k_j$. By induction hypothesis, we have that for every $h < i, r_h = m_h\delta_j = \mathsf{m}_{\mathcal{P}_j}(h, l_j\theta')$. So,

$$\begin{array}{rcl} r_i & = & m_i\delta_j \\ & = & \mathsf{b}_j\lfloor l_j\theta', b_i, [\mathsf{m}_{\mathcal{P}_j}(i-1, l_j\theta'); \dots; \mathsf{m}_{\mathcal{P}_j}(1, l_j\theta')]\rfloor \\ & = & \mathsf{m}_{\mathcal{P}_j}(i, l_j\theta'). \end{array}$$

– if $i > k_j$, we have that $r_i = m_{n-k+i}\delta_j$ and $b_i = a_{n-k+i}$. Let $i' = n-k+i$. As $\langle \mathcal{P}_j, \langle l_j\theta, l'_j\theta\rangle\rangle \in \mathcal{L}$, $m_{i'} = \mathsf{b}_j\lfloor l_j\theta, a_{i'}, [m_{i'-1}; \dots; m_1]\rfloor$. As b_j does not contain the list constructor and δ_j is a swapping of (non empty) lists, by definition of a swapping, we have that

$$\mathsf{b}_j\lfloor l_j\theta, a_{i'}, [m_{i'-1}; \dots; m_1]\rfloor\delta_j = \mathsf{b}_j\lfloor (l_j\theta)\delta_j, a_{i'}\delta_j, [m_{i'-1}; \dots; m_1]\delta_j\rfloor$$

We have that $a_{i'}\delta_j = a_{i'} = b_i$ and $(l_j\theta)\delta_j = (l_j\theta)\delta = l_j\theta'$ thanks to Claims (A) and (B). We have $[m_{n-k+i-1}; \dots; m_1] = m_{n-k+i-1} :: \dots :: m_{n+1-k_v} :: w_j$, and by applying the swapping:

$$\begin{array}{rcl} [m_{n-k+i-1}; \dots; m_1]\delta_j & = & m_{n-k+i-1}\delta_j :: \dots :: m_{n+1-k_v}\delta_j :: w_j\delta_j \\ & = & [m_{n-k+i-1}\delta_j; \dots; m_{n+1-k_v}\delta_j; m_{k_j}\delta_j; \dots; m_1\delta_j] \end{array}$$

For $h \leq k_j$, we know that $r_h = m_h\delta_j$. For $k_j < h < i$, $r_h = m_{n-k+h}\delta_j$. By applying the induction hypothesis, for all $h < i$, $r_h = \mathsf{m}_{\mathcal{P}_j}(h, l_j\theta')$. As $k = k_v + k_j$, we deduce that

$$\begin{array}{rcl} [m_{n-k+i-1}; \dots; m_1]\delta_j & = & [r_{i-1}; \dots; r_{k-k_v+1}; r_{k_j}; \dots; r_1] \\ & = & [r_{i-1}; \dots; r_1] \\ & = & [\mathsf{m}_{\mathcal{P}_j}(i-1, l_1\theta'); \dots; \mathsf{m}_{\mathcal{P}_j}(1, l_1\theta')] \end{array}$$

So $r_i = \mathsf{b}_j\lfloor l_j\theta', b_i, [\mathsf{m}_{\mathcal{P}_j}(i-1, l_j\theta'); \dots; \mathsf{m}_{\mathcal{P}_j}(1, l_j\theta')]\rfloor = \mathsf{m}_{\mathcal{P}_j}(i, l_j\theta')$.

We have thus shown that $\langle \mathcal{P}_j, \langle l_j\theta', l'_j\theta'\rangle\rangle \in \mathcal{L}$.

- Second, show that for $j > q$, $\langle \mathcal{P}_j, \langle l_j\theta', l'_j\theta'\rangle\rangle \in \mathcal{L}$.

  Thanks to Claim A, $l_j\theta' = (l_j\theta)\delta$, and by applying Claim B, $(l_j\theta)\delta = (l_j\theta)\delta_j$, and similarly $l'_j\theta' = (l'_j\theta)\delta_j$. As $\delta_j = id$, it follows that $(l_j\theta', l'_j\theta') = (l_j\theta, l'_j\theta)$. Hence, $\langle \mathcal{P}_j, \langle l_j\theta', l'_j\theta'\rangle\rangle = \langle \mathcal{P}_j, \langle l_j\theta, l'_j\theta\rangle\rangle \in \mathcal{L}$.

We have thus shown that θ' is a solution of $\phi = \psi\sigma$. As a consequence, $\tau' = \theta' \circ \sigma$ is a solution of ψ.

Show that τ' is a constructive solution of $(\mathcal{C}, \mathcal{I})$. Write $\mathcal{C} = T_1 \Vdash z_1 \wedge \dots \wedge T_n \Vdash z_n$. We will show by induction on i that $Sat T_i\tau' \cup \mathcal{I} \vdash z_i\tau'$ using composition rules only.

Claim C will help us with the proof, but first we need to show that when $x_1\theta$ is deducible using composition rules only, then $x_1\theta' = w_0$ is also deducible using composition rules only.

Let i_0 be minimal such that $SatT_{i_0}\tau \cup \mathcal{I} \vdash x_1\theta$ using composition rules only. Otherwise, let $i_0 = n+1$. We show that, if $i_0 \leq n$, $SatT_{i_0}\tau' \cup \mathcal{I} \vdash w_0$ using composition rules only.
Recall that $x_1\theta = [a^1_{k_1+1};\ldots;a^1_{n_1}]$ and $w_0 = [a^1_{n_1-k_v+1};\ldots;a^1_{n_1}]$ with a^1_k of sort Base. Let $n_1 - k_v + 1 \leq k \leq n_1$. As τ is a constructive solution of $(\mathcal{C},\mathcal{I})$ in solved form and $x_1\theta$ is a list such that $SatT_{i_0}\tau \cup \mathcal{I} \vdash x_1\theta$ using composition rules only, we can apply Lemma 4.1.1: we have that $SatT_{i_0}\tau \cup \mathcal{I} \vdash a^1_k$ using composition rules only. As a^1_k is of sort Base, it follows that $a^1_k \in SatT_{i_0}\tau \cup \mathcal{I}$, and so $a^1_k = a^1_k\delta \in SatT_{i_0}\tau' \cup \mathcal{I}$. Consequently, there is a proof of $SatT_{i_0}\tau \cup \mathcal{I} \vdash w_0$ using composition rules only.
Consider $j \leq q$ (*i.e.* $\mathsf{tail}(l_j) = x_1$). If there exists i such that $SatT_i\tau \cup \mathcal{I} \vdash w_j$ using only composition rules, let i_j be minimal such that $SatT_{i_j}\tau \cup \mathcal{I} \vdash w_j$ using composition rules only. Otherwise, let $i_j = n+1$. We show that, for $i_j \leq n$, $SatT_{i_j}\tau' \cup \mathcal{I} \vdash w'_j$ using composition rules only.

***Claim C**: $Sat(T_{i_j})\tau' \cup \mathcal{I} \vdash w'_j$ using composition rules only.*
Recall that $w_j = [m_{n-k_v};\ldots;m_1]$ and $w'_j = [m_{k_j}\delta_0;\ldots;m_1\delta_0]$ (with $n-k_v > k_j$). We show that for every $1 \leq k \leq k_j$, $SatT_{i_j}\tau' \cup \mathcal{I} \vdash m_k\delta_0$ using composition rules only.

- If the case where $i_j \geq i_0$. Let $1 \leq k \leq k_j$. As τ is a constructive solution of $(\mathcal{C},\mathcal{I})$ in solved form and w_j is a list such that $Sat(T_{i_j})\tau \cup \mathcal{I} \vdash w_j$ using composition rules only, we can apply Lemma 4.1.1: we have that $Sat(T_{i_j})\tau \cup \mathcal{I} \vdash m_k$ using composition rules only. As a consequence, there is a context D_k minimal in size and terms $t_1,\ldots,t_q \in Sat(T_{i_j}) \cup \mathcal{I}$ such that $D_k\lfloor t_1\tau,\ldots,t_q\tau, x_1\theta\rfloor = m_k$.

  We want to show that $D_k\lfloor t_1\tau',\ldots,t_q\tau', x_1\theta'\rfloor = m_k\delta_0$.

  First, thanks to Claim B, $m_k\delta = m_k\delta_j$. Furthermore, m_k is a strict subterm of w_j, so $m_k\delta_j = m_k\delta_0$. Thus, for every subterm u of m_k, $u\delta = u\delta_0$. We deduce that
  $$D_k\lfloor t_1\tau,\ldots,t_q\tau, x_1\theta\rfloor\delta = D_k\lfloor t_1\tau,\ldots,t_q\tau, x_1\theta\rfloor\delta_0.$$
  As D_k is a minimal context,
  $$(D_k\lfloor t_1\tau,\ldots,t_q\tau, x_1\theta\rfloor)\delta_0 = D_k\lfloor (t_1\tau)\delta_0,\ldots,(t_q\tau)\delta_0,(x_1\theta)\delta_0\rfloor.$$
  Indeed, suppose by contradiction that there exists a non empty subcontext E of D_k such that $E\lfloor t_1\tau,\ldots,t_q\tau, x_1\theta\rfloor = x_1\theta$: then D_k is not minimal. Furthermore, $(t_i\tau)\delta = ((t_i\sigma)\theta)\delta = t_i\sigma(\theta\delta) = t_i\tau'$ thanks to Claim A.

  We have that $(t_i\tau)\delta_0 = (t_i\tau)\delta$, as $t_i \in st(m_k)$ and $m_k\delta = m_k\delta_0$. We deduce that $(t_i\tau)\delta_0 = t_i\tau'$. We also have that $D_k\lfloor (t_1\tau)\delta_0,\ldots,(t_q\tau)\delta_0,(x_1\theta)\delta_0\rfloor = m_k\delta_0$. We deduce that $D_k\lfloor t_1\tau',\ldots,t_q\tau', x_1\theta'\rfloor = m_k\delta$. Furthermore, since $i_j \geq i_0$ and $Sat(T_{i_0})\tau \cup \mathcal{I} \vdash w_0 = x_1\theta'$ using composition rules only, we conclude that there is a constructive proof of $Sat(T_{i_j})\tau \cup \mathcal{I} \vdash m_k\delta_0$.

- In the case where $i_j < i_0$, then similarly there is a context D_k minimal in size and terms $t_1,\ldots,t_q \in Sat(T_{i_j}) \cup \mathcal{I}$ such that $D_k\lfloor t_1\tau,\ldots,t_q\tau\rfloor = m_k$. By a similar reasoning, we have that
  $$\begin{aligned} D_k\lfloor t_1\tau,\ldots,t_q\tau\rfloor\delta &= D_k\lfloor t_1\tau,\ldots,t_q\tau\rfloor\delta_0 \\ &= D_k\lfloor (t_1\tau)\delta_0,\ldots,(t_q\tau)\delta_0\rfloor \end{aligned}$$
  Indeed, suppose by contradiction that there exists a non empty subcontext E of D_k such that $E\lfloor t_1\tau,\ldots,t_q\tau\rfloor = x_1\theta$: then i_0 is not minimal. We conclude in a similar manner that there is a constructive proof of $Sat(T_{i_j})\tau \cup \mathcal{I} \vdash m_k\delta_0$.

For every $1 \leq k \leq k_j$, there is a constructive proof of $Sat(T_{i_j})\tau' \cup \mathcal{I} \vdash m_k\delta_0$, so there is a proof of $Sat(T_{i_j})\tau' \cup \mathcal{I} \vdash w'_j$ using composition rules only.

Now, we show that for every i, $Sat(T_i)\tau' \cup \mathcal{I} \vdash z_i\tau'$ using composition rules only.
Let $S_i = \begin{cases} \{w_j | i_j \leq i\} \cup \{x_1\theta\} \text{ if } i \leq i_0 \\ \{w_j | i_j \leq i\} \end{cases}$
Note that if $w_k \notin S_i$, then there is no constructive proof of $Sat(T_i)\tau \cup \mathcal{I} \vdash w_k$. If $w_k \in S_i$, thanks to Claim C there is a constructive proof of $Sat(T_{i_k})\tau \cup \mathcal{I} \vdash w'_k$ with $i_k \leq i$, so there is a constructive proof of $Sat(T_i)\tau \cup \mathcal{I} \vdash w'_k$. The same holds for $x_0\theta$ and w_0.

As τ is a constructive solution, there is a minimal context C_i and terms $t_1, \ldots, t_p \in Sat(T_i) \cup \mathcal{I}$ such that $C_i\lfloor t_1\tau, \ldots, t_p\tau, S_i\rfloor = z_i\tau$. We apply the swapping δ: $C_i\lfloor t_1\tau, \ldots, t_p\tau, S_i\rfloor\delta = (z_i\tau)\delta$. Thanks to Claim A, we have that $(z_i\tau)\delta = (z_i\sigma\theta)\delta = (z_i\sigma)(\theta\delta)$, and $(z_i\sigma)(\theta\delta) = (z_i\sigma)\theta' = z_i\tau'$. Furthermore, $C_i\lfloor t_1\tau, \ldots, t_p\tau, S_i\rfloor\delta = C_i\lfloor (t_1\tau)\delta, \ldots, (t_p\tau)\delta, S_i\delta\rfloor$, as C_i is minimal and for all k such that $w_k \notin S_i$, then there is no constructive proof of $Sat(T_i)\tau \cup \mathcal{I} \vdash w_k$. Now, for every k, $(t_k\tau)\delta = ((t_k\sigma)\theta)\delta = t_k\sigma(\theta\delta) = t_k\tau'$ thanks to Claim A. Consequently, $C_i\lfloor t_1\tau', \ldots, t_p\tau', S_i\delta\rfloor = z_i\tau'$. For every $w_k \in S_i$, there is a constructive proof of $Sat(T_i)\tau \cup \mathcal{I} \vdash w'_k = w_k\delta$. Furthermore, if $x_1\theta \in S_i$, then $i \geq i_0$ and consequently there is a constructive proof of $Sat(T_i)\tau \cup \mathcal{I} \vdash w_0$. We deduce that there is a constructive proof of $Sat(T_i)\tau' \cup \mathcal{I} \vdash z_i\tau'$.

As a conclusion, τ' is a constructive solution of $(\mathcal{C}, \mathcal{I})$ and ψ smaller than τ. □

Theorem 4.3.4. *Let $\mathcal{L}$ be a mapping-based language. Let $(\mathcal{C}, \mathcal{I})$ be a constraint system and ϕ be an $\mathcal{L}$-language constraint associated to $(\mathcal{C}, \mathcal{I})$.*
Deciding whether $(\mathcal{C}, \mathcal{I}) \wedge \phi$ has a solution is in NP.

The proof of Theorem 4.3.4 involves three main steps. First, thanks to Theorem 2.3.1, it is sufficient to decide in polynomial (DAG) size whether $(\mathcal{C}, \mathcal{I})$ with language constraint ϕ has a non-confusing solution when $(\mathcal{C}, \mathcal{I})$ is a solved constraint system. Due to Proposition 2.4.1, we deduce that it is sufficient to show that deciding whether $(\mathcal{C}, \mathcal{I}) \wedge \phi$ has a constructive solution is in NP, where $(\mathcal{C}, \mathcal{I})$ is a solved constraint system.

The second and key step of the proof consists in bounding the size of a constructive solution. Note that the requirement on the form of ϕ is not a restriction since any substitution satisfying ϕ will necessarily have this shape.

For each constraint $\langle\langle p_1^j, \ldots, p_m^j\rangle, \langle l_j, l'_j\rangle\rangle \overset{?}{\in} \mathcal{L}$, the list l_j provides constraints on the last elements of the list l'_j, while l'_j provides constraints on the last elements of the list l_j. The main idea of the proof of Proposition 4.3.3 is to show that it is possible to cut the middle of the list l_j, modifying the list l'_j accordingly. This is however not straightforward as we have to show that the new substitution is still a solution of the constraint system $(\mathcal{C}, \mathcal{I})$. In particular, cutting part of the list might destroy some interesting equalities, used to deduce terms. Such cases are actually avoided by considering constructive solutions and by cutting at some position in the lists such that none of the elements are subterms of the constraint, which can be ensured by combinatorial arguments.

Proposition 4.3.3 allows us to bound the size of $l_j\theta$ for a minimal solution θ, which in turn bounds the size of $l'_j\theta$. The last step of the proof of Theorem 4.3.4 consists in showing that any $x\theta$ is bounded by the size of the lists or can be replaced by a constant, by applying Lemma 4.1.2.
We want to decide whether $(\mathcal{C}, \mathcal{I}) \wedge \phi$ has a solution.

Write $\mathcal{C} = T_1 \Vdash u_1 \wedge \cdots \wedge T_n \Vdash u_n$ and $\phi = v_1 \overset{?}{\in} \mathcal{L} \wedge \cdots \wedge v_p \overset{?}{\in} \mathcal{L}$. Our decision procedure works as follows:

Step 1. First, we may assume without loss of generality that for each $j \leq p$, the term v_j is of the following form: $v_j = \langle\langle p_1^i, \ldots, p_m^i\rangle, \langle \ell_j, \ell'_j\rangle\rangle$.

Step 2. We guess a sequence of transformation rules in strategy $\mathcal{S}$ from $(\mathcal{C}, \mathcal{I})$ to $(\mathcal{C}', \mathcal{I})$ where $(\mathcal{C}', \mathcal{I})$ is a constraint system in solved form. We have that:

$$(\mathcal{C}, \mathcal{I}) \rightsquigarrow_\sigma^* (\mathcal{C}', \mathcal{I}) \in \mathcal{S} \text{ with } (\mathcal{C}', \mathcal{I}) \text{ in solved form.}$$

Let $\phi' = \langle\langle p_1^1\sigma, \ldots, p_m^1\sigma\rangle, \langle l_1, l'_1\rangle\rangle \overset{?}{\in} \mathcal{L} \wedge \cdots \wedge \langle\langle p_1^p\sigma, \ldots, p_m^p\sigma\rangle, \langle l_p, l'_p\rangle\rangle \overset{?}{\in} \mathcal{L}$ where $l_j = \ell_j\sigma$ and $l'_j = \ell'_j\sigma$.

Step 3. Let $L = \max_{1\leq j\leq p} \|l_j\|_{dag}$.

Let $M = \#st(\mathcal{C}', \phi') + \max_{1\leq j\leq p} \|l'_j\|_l + 2 \times \#var(\mathcal{C}') \times \max_{t\in st(\mathcal{C}',\phi')} \|t\|_{dag}$.

Let $\mathcal{I}_0 \subseteq \mathcal{I}$ of size $p \times 2(M+L) \times (\|\mathsf{b}\|_{dag} + m + 2 + 2M + 2L) + 1$. Guess the values of variables in terms built over $names(\mathcal{C}, \phi) \cup \mathcal{I}_0$ of size at most $2(M+L) \times (\|\mathsf{b}\|_{dag} + m + 2 + 2M + 2L) + 2m + 1$. This gives us a substitution θ', and we check whether θ' is a solution of $(\mathcal{C}', \mathcal{I})$ and ϕ'.

Proof.

We show that these steps allow us to guess a solution of $(\mathcal{C}, \mathcal{I})$ and ϕ in polynomial time.

Step 1. We can write $v_j = \langle\langle p_1^j, \ldots, p_m^j\rangle, \langle \ell_j, \ell'_j\rangle\rangle$. Indeed, if θ is a solution of the constraint $v_j \overset{?}{\in} \mathcal{L}$, then by definition of $\mathcal{L}$, there is a tuple of names $\langle p_1, \ldots, p_m\rangle$ and ground lists l, l' such that $v_j\theta = \langle\langle p_1, \ldots, p_m\rangle, \langle l, l'\rangle\rangle$. We can thus compute $\sigma_j = mgu\{v_j = \langle\langle x_1, \ldots, x_m\rangle, \langle y_1, y_2\rangle\rangle\}$ where $x_1, \ldots, x_m, y_1, y_2$ are fresh variables, with $x_1, \ldots, x_m$ of Base sort and y_1, y_2 of List sort. Then, we can apply substitution σ_j to $\mathcal{C}$. The DAG size of $\mathcal{C}$ grows at most by $2 \times (2+m)$ for each transformation.

Step 2. We can apply Theorem 2.3.1: there exists a solution θ of $(\mathcal{C}, \mathcal{I})$ if, and only if, there exist a deduction constraint system $(\mathcal{C}', \mathcal{I})$ in solved form and substitutions σ, θ' such that $(\mathcal{C}, \mathcal{I}) \rightsquigarrow_\sigma^* (\mathcal{C}', \mathcal{I})$ by a derivation in strategy $\mathcal{S}$, $\theta = \theta' \circ \sigma$, and θ' is a non-confusing solution of $(\mathcal{C}', \mathcal{I})$. Moreover, we have that θ' is a solution of $\phi\sigma$. The length of this derivation is polynomially bounded in the DAG size of $\mathcal{C}$ and the DAG size of $\mathcal{C}'$ is also polynomially bounded by the DAG size of $\mathcal{C}$. We can guess such a derivation, and are now left to decide the existence of a non-confusing solution θ' to $(\mathcal{C}', \mathcal{I})$ and $\phi\sigma$.

Thanks to Proposition 2.4.1, a non-confusing solution of $(\mathcal{C}', \mathcal{I})$ is in particular a constructive solution.

Step 3. We want to decide whether there exists a constructive solution θ' to the constraint system $(\mathcal{C}', \mathcal{I})$ and $\phi' = \phi\sigma$ where $\phi' = v'_1 \overset{?}{\in} \mathcal{L} \wedge \cdots \wedge v'_p \overset{?}{\in} \mathcal{L}$, with

- $v'_j = \langle\langle p_1^j, \ldots, p_m^j\rangle, \langle l_j, l'_j\rangle\rangle$, and
- $(\mathcal{C}', \mathcal{I})$ is in solved form.

Thanks to Lemma 4.3.3, if such a solution θ' exists, we can assume that

$$\|\mathsf{tail}(l_j)\theta'\|_l \leq M = \#st(\mathcal{C}', \phi') + \max_{1\leq j\leq p} \|l'_j\|_l + 2 \times \#var(\mathcal{C}') \times \max_{t\in st(\mathcal{C}',\phi')} \|t\|_{dag}.$$

for every $1 \leq j \leq p$.

As $l_j\theta'$ is a list of names, we deduce that $\|l_j\theta'\|_{dag} \leq 2 \times \|l_j\theta'\|_l \leq 2 \times (M + L)$. We now bound the size of $l'_j\theta'$ (in DAG representation), using the language constraint $\langle\langle p^j_1, \ldots, p^j_m\rangle, \langle l_j, l'_j\rangle\rangle \overset{?}{\in} \mathcal{L}$. Indeed, by definition of the language $\mathcal{L}$, if $l_j\theta' = [a_1; \ldots; a_n]$, then $l'_j\theta' = [m_n; \ldots; m_1]$ with $m_i = \mathsf{b}_j\lfloor l_j\theta', a_i, [m_{i-1}; \ldots; m_1]\rfloor$ where $\mathsf{b}_j = \mathsf{b}\{w^p_1 \mapsto p^j_1\theta', \ldots, w^p_m \mapsto p^j_m\theta'\}$. For every $i \leq n$, we can bound the size of $[m_{i+1}, \ldots, m_1]$ with respect to the size of $[m_i, \ldots, m_1]$:

$$\|[m_{i+1}, \ldots, m_1]\|_{dag} \leq 1 + \|[m_i, \ldots, m_1]\|_{dag} + \|\mathsf{b}\|_{dag} + m + \|l_j\theta'\|_{dag} + 1.$$

Consequently,

$$\begin{aligned}\|l'_j\theta'\|_{dag} &\leq n \times (\|\mathsf{b}\|_{dag} + m + 2 + \|l_j\theta'\|_{dag}) \\ &\leq \|l_j\theta'\|_{dag} \times (\|\mathsf{b}\|_{dag} + m + 2 + \|l_j\theta'\|_{dag}) \\ &\leq 2(M + L) \times (\|\mathsf{b}\|_{dag} + m + 2 + 2M + 2L).\end{aligned}$$

Then, thanks to Lemma 4.1.2, we can assume that for every variable x, either $x\theta' \in st(\phi'\theta')$, or $x\theta' \in \{N_0, []\}$ with $N_0 \in \mathcal{I}$. Thus, we can guess the values of $x\theta'$ by considering only a finite subset of names of $\mathcal{I}$ of size

$$p \times 2(M + L) \times (\|\mathsf{b}\|_{dag} + m + 2 + 2M + 2L) + 1$$

Moreover, for every variable $x \in dom(\theta')$, we have that:

$$\begin{aligned}\|x\theta'\|_{dag} &\leq \max\{\|v'_j\theta'\|_{dag} \mid 1 \leq j \leq p\} \\ &\leq \max\{\|l'_j\theta'\|_{dag} \mid 1 \leq j \leq p\} + 2m + 1 \\ &\leq 2(M + L) \times (\|\mathsf{b}\|_{dag} + m + 2 + 2M + 2L) + 2m + 1\end{aligned}$$

We can check whether a given substitution is a solution of $(\mathcal{C}', \mathcal{I}) \wedge \phi$ in polynomial time. In order to conclude, It only remains to show that M and L are polynomial in the size of $\mathcal{C}, \phi$. We have that

- $M = \#st(\mathcal{C}', \phi') + \max_{1\leq j\leq p} \|\ell'_j\sigma\|_l + 2 \times \#var(\mathcal{C}') \times \max_{t\in st(\mathcal{C}',\phi')} \|t\|_{dag}$, and
- $L = \max_{1\leq j\leq p} \|\ell_j\sigma\|_{dag}$.

First, we have that $\#st(\mathcal{C}', \phi') = \#st(\mathcal{C}, \phi)$. Then, for every term $u \in st(\mathcal{C}, \phi)$, we get that $\|u\sigma\|_{dag} \leq \|u\|_{dag} + \#var(\mathcal{C}) \times \max_{t\in st(\mathcal{C},\phi)} \|t\|_{dag}$. This allows us to conclude. □

4.4 Conclusion and future prospects

We have provided two new NP decision procedures for (automatically) analysing confidentiality of security protocols with recursive tests, for a bounded number of sessions. The classes of recursive languages we can consider both encompass chained-based lists of certificates and most of the recursive tests performed in the context of routing protocols.

Analyzing more recursivity tests. The way we have modeled both SMNDP and endairA is not totally accurate. During the execution of both these protocols, the intermediate nodes perform recursive checks too, on the partially built list of signatures. If the test fails, they drop the message. We do not lose attacks with our modeling but we may discover false attacks as some messages that should have been dropped are transferred without any hindrance. We would have to add a language for partially-built signature lists. Defining such a language is easy, but not so to see whether the decidability result still holds. Our proof makes use of the fact that if two signature lists share a link, then they are equal, which would no longer be the case. To circumvent that, we could define a notion of family, which would intuitively regroup all the signature lists corresponding to a certain path.

Furthermore, in order to model Ariadne, where the recursivity test is different, we need to again define another new class of recursive languages. In this protocol, the list of signatures providing authentication is built during the request phase, and so both the list representing the path and the list authenticating the path grow along the way. It is possible that those two classes of languages could be linked, they both involve partial lists of signatures. The difference lies in the fact that in Ariadne, the list of nodes grows along the way, in contrast with endairA or SMNDP, where the list of nodes is fixed.

Analyzing routing protocols with recursive tests. The attack on SRP given in the previous chapter shows that the intermediate links intuitively must actively participate by authenticating the list in route discovery if we hope to prove correctness. Recursively built authentication is one way to achieve this participation.

Combining recursivity tests and route property modelisation, i.e. the results obtained in this chapter and in the previous one, is both logical and desirable in order to wrap up things neatly. Notice that adding neighborhood constraints and other local properties of lists to recursivity tests seems easily feasible. The crucial component to add is the property of route correctness, and as this property concerns the entire route it interferes with the recursivity test, which also concerns the entire route. In both chapters, we show decidability by showing a small attack property: if there is an attack, there is one where we can bound the size of lists involved in the attack. But the bounding relies on totally different techniques in the proofs. For recursivity, we cut lists in the middle. In the other case, we have to keep some nodes in the list and we discard more or less the other ones, modulo some constraints. Combining the two approaches seems possible, but it is not straightforward.

Chapter 5

Conclusion and Future Prospects

In this dissertation, we interested ourselves in ad hoc routing protocols. We proposed a model that takes into account the particularities of these networks, such as the network topology, the broadcast nature of communication and the particular tests performed in these protocols. We have obtained decidability results for source routing protocols without recursivity tests, and for protocols using only recursivity tests. A natural further direction for research would be to study source routing protocols with recursive tests. This possibility has already been discussed in Chapter 4 (see Section 4.4). Another natural direction is to study table routing protocols, that we are able to model up to a certain point. This possibility was discussed in Chapter 3 (see Section 3.5).

In this concluding chapter, we discuss other research directions. We believe it is possible to produce a tool that would automatically analyze routing protocols and either detect attacks or prove them secure. It would also be interesting to consider other security properties, and in particular anonymity, as there is a specific family of ad hoc routing protocols designed to guarantee anonymity. Finally, an important aspect of ad hoc networks we have not modeled is mobility, and we would like to tackle this issue.

5.1 Towards automation

For constraint systems with small attacks properties, tools have been developed that can find attacks without having to search all the space of possible solutions. For trace based properties in particular, there are efficient tools that can detect attacks or guarantee security. We want to build such a tool tailored for analyzing routing protocols. It is not possible to use existing tools, at least without any modification: we have to take into account the topology of the network, the broadcast primitive, an intruder localized at a precise node and particular security properties using the underlying graph. Such particularities are not accounted for in the existing tools.

A first possibility would be to implement our procedure. Of course, this would require to adapt it: at the moment it is unfit for automation, with too many guesses. However, the bounds we give are not tight, and some guesses could be more accurate. Instead of guessing a list and then checking that it satisfies all the constraints, we could build a list taking into account from the beginning some of the constraints and only check it against the remaining constraints. In particular, all the local checks could give a basis for the possible solutions.

Another approach would be to use existing tools and to find a way to model the particularities of routing protocols. The crucial point is to be able to take into account the underlying graph, and the modified model for the intruder.

It is possible to encode routing protocols for a fixed topology. [PPB10] uses AVISPA to analyze ARAN, but without taking the topology into account, letting the intruder control the network. However, this study shows that table routing protocols can be modeled in AVISPA and analyzed. Benetti, Merro and Viganò [vig10] use the AVISPA tool to automatically analyse some execution scenarios of the ARAN and endairA protocols, and find some attacks on ARAN. They consider two different network topologies for ARAN and discover attacks using AVISPA. [ABY11] uses a model checking approach using the tool SPIN in order to analyze ARAN in fixed topologies.

Checking particular scenarios of different routing protocols has already been done using existing tools. The process however is done for small networks, as the graph needs to be hard coded into the analysis tool. In order to analyze larger networks or networks with unknown topology, a promising approach consists in reducing the size of the graph for which there can be an attack, instead of focusing on the size of the solution.

In order to analyze all possible cases of topologies, [ABY11] uses a technique that consists in reducing the search space by building topology equivalence classes and testing the smallest topology in each class.

J. Degriek [Deg11] has already shown that looking for an attack can be reduced to looking for an attack on small graphs. Any attack can be transformed in an attack on a smaller graph, by distinguishing nodes that invalidate the routing property and regrouping the other ones. This approach has some limitations in the design of the protocol, and particularly makes use of only a fraction of the logic. Attacks on SRP and SMNDP were retrieved by implementing this method in Proverif. This approach allows to consider only a finite number of small graphs.

5.2 Anonymity

Some routing protocols, in addition to other security properties, wish to maintain anonymity of the participants to the protocol. These protocols are usually fairly complex, they make a rather heavy use of recursivity. The messages are like onions, with multiple layers of encryptions to be taken off in order to reach the destination. These protocols may involve recursive input/output steps, and our results in Chapter 4 can not be applied in this case. We proved decidability for classes of languages capturing recursivity tests only. It would be interesting to see if we could reuse some of the results as part of a work that would deal with recursive input/output steps with certain conditions to preserve decidability.

The difficulty with anonymous routing protocols does not lie only with modeling the protocol but also with modeling the security property. The anonymity property is delicate to express even informally, more so in a formal way. Some works recently tackled this issue for different applications: RFID tags [BCd10], voting protocols [DKR09] for instance. In order to capture the notion of anonymity, they use indistinguishability of two executions where two agents have exchanged their secret. Consequently, the notion of indistinguishability is not a trace-based property but an equivalence-based one. In order to prove this property, we have to consider two different sets of traces and compare them. There already exist some algorithms for deciding trace equivalence for deducibility constraints, and Proverif [Bla05] makes use of some of them, but they probably need to be adapted in order to be used in the context of anonymous routing protocols.

5.3 Challenges of mobility

It should be noted that we do not take mobility into account in the sense that the topology of the network does not change during our analysis. There are two main reasons for this limitation. First, many flaws can already be detected without any change in the network topology. Second, properties like the validity of a route are of course (temporarily) invalidated during a network topology modification. Therefore, such properties have to be analyzed once the network is stabilized, previous routing protocol executions being possibly included in the initial knowledge of the attacker.

An extension would be to model mobility during the execution of the protocol. This would allow us to consider changes in the network topology and to analyze the security of route updates.

In [Mer07], M. Merro proposes a process calculus to study the observational theory of mobile ad hoc networks, called CMN (Calculus for Mobile Networks). He establishes a bisimilarity relation that enables him to prove some structural properties of mobile networks, e.g. a node that does not send any message can not be observed.

J. Godskesen proposes in [God07] a Calculus for Mobile Ad Hoc Networks, CMAN. In this model, nodes may autonomously change their neighbor relationship and thereby change the network topology. He shows behavioural equivalences between processes and analyze ARAN. He proves that ARAN is not robust as adding an intruder leads to a process that is not equivalent to the process with which they began.

J. Godskesen and S. Nanz then worked together [GN09] to establish a realistic mobility model. They describe the movements of the nodes with a mobility function. A process calculus taking the time into account is set, and the mobility functions can be compared through bisimulation.

Models for mobile networks exist, but they do not include cryptography. Furthermore, adding mobility to the network requires to model an appropriate security property. Our decidability result holds only for the logic $\mathcal{L}_{\mathsf{route}}$, though the concrete and symbolic model hold for any logic.

Bibliography

[ABB$^+$05] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron, *The AVISPA Tool for the automated validation of internet security protocols and applications*, Proc. of the 17th International Conference on Computer Aided Verification, CAV'2005, LNCS, vol. 3576, Springer, 2005, pp. 281–285.

[ABV05] Gergely Ács, Levente Buttyán, and István Vajda, *Provable security of on-demand distance vector routing in wireless ad hoc networks*, Second European Workshop on Security and Privacy in Ad Hoc and Sensor Networks (ESAS 2005), 2005.

[ABV06] ______, *Modelling adversaries and security objectives for routing protocols in wireless sensor networks*, The Fourth ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN 2006), 2006.

[ABY11] Todd R. Andel, G. Back, and Alec Yasinsac, *Automating the security analysis process of secure ad hoc routing protocols*, Simulation Modelling Practice and Theory **19** (2011), no. 9, 2032–2049.

[AC06] M. Abadi and V. Cortier, *Deciding knowledge in security protocols under equational theories*, Theoretical Computer Science **387** (2006), no. 1-2, 2–32.

[ACC$^+$08] A. Armando, R. Carbone, L. Compagna, J. Cuéllar, and M. L. Tobarra, *Formal analysis of SAML 2.0 web browser single sign-on: breaking the SAML-based single sign-on for google apps*, Proc. of the 6th ACM Workshop on Formal Methods in Security Engineering (FMSE 2008), 2008, pp. 1–10.

[ACD09] Mathilde Arnaud, Véronique Cortier, and Stéphanie Delaune, *Modeling and verifying ad hoc routing protocol*, Preliminary Proceedings of the 4th International Workshop on Security and Rewriting Techniques (SecReT'09) (Port Jefferson, NY, USA) (Hubert Comon-Lundh and Catherine Meadows, eds.), July 2009, pp. 33–46.

[ACD10] ______, *Modeling and verifying ad hoc routing protocols*, Proc. of the 23rd IEEE Computer Security Foundations Symposium (CSF'10), IEEE Computer Society Press, 2010, pp. 59–74.

[ACD11] ______, *Deciding security for protocols with recursive tests*, Proceedings of the 23rd International Conference on Automated Deduction (CADE'11) (Wrocław, Poland) (Nikolaj Bjørner and Viorica Sofronie-Stokkermans, eds.), Lecture Notes in Artificial Intelligence, Springer, July 2011, pp. 49–63.

[AF01] M. Abadi and C. Fournet, *Mobile values, new names, and secure communication*, Proc. 28th Symposium on Principles of Programming Languages (POPL'01), ACM Press, 2001, pp. 104–115.

[AG97] M. Abadi and A. Gordon, *A calculus for cryptographic protocols: The spi calculus*, Proc. 4th Conference on Computer and Communications Security (CCS'97), ACM Press, 1997, pp. 36–47.

[AG00] N. Asokan and Philip Ginzboorg, *Key agreement in ad hoc networks*, Computer Communications **23** (2000), no. 17, 1627–1637.

[ALV02] R. Amadio, D. Lugiez, and V. Vanackère, *On the symbolic reduction of processes with cryptographic functions*, Theoretical Computer Science **290** (2002), no. 1, 695–740.

[And07] Todd R. Andel, *Formal Security Evaluation of Ad Hoc Routing Protocols*, Ph.D. thesis, Florida State University, 2007.

[AR00] M. Abadi and P. Rogaway, *Reconciling two views of cryptography: the computational soundness of formal encryption*, Proceedings of the 1rst IFIP International Conference on Theoretical Computer Science, LNCS, vol. 1872, 2000.

[Aur99] T. Aura, *Distributed access-rights management with delegation certificates*, Secure Internet Programming, LNCS, vol. 1603, 1999, pp. 211–235.

[AY07] Todd R. Andel and Alec Yasinsac, *Automated security analysis of ad hoc routing protocols*, Proc. of the Joint Workshop on Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis (FCS-ARSPA'07), 2007, pp. 9–26.

[BCd10] Mayla Bruso, Konstantinos Chatzikokolakis, and Jerry den Hartog, *Formal Verification of Privacy for RFID Systems*, Computer Security Foundations Symposium – CSF 2010 (Edinburgh, United Kingdom), IEEE, July 2010.

[BDC09] Sergiu Bursuc, Stéphanie Delaune, and Hubert Comon-Lundh, *Deducibility constraints*, Proceedings of the 13th Asian Computing Science Conference (ASIAN'09) (Seoul, Korea) (Anupam Datta, ed.), Lecture Notes in Computer Science, vol. 5913, Springer, December 2009, pp. 24–38.

[BEKXK04] Azzedine Boukerche, Khalil El-Khatib, Li Xu, and Larry Korba, *A novel solution for achieving anonymity in wireless ad hoc networks*, PE-WASUN '04: Proceedings of the 1st ACM international workshop on Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks (New York, NY, USA), ACM, 2004, pp. 30–38.

[BGP95] *A Border Gateway Protocol 4 (BGP-4)*, 1995.

[BHS10] Remco Boer, Cassandra Hensen, and Adriana Screpnic, *Online payments 2010 Increasingly a global game*, May 2010.

[Bla01] B. Blanchet, *An efficient cryptographic protocol verifier based on prolog rules*, Proc. of the 14th Computer Security Foundations Workshop (CSFW'01), IEEE Computer Society Press, 2001.

[Bla05] B. Blanchet, *An automatic security protocol verifier based on resolution theorem proving (invited tutorial)*, Proc. of the 20th International Conference on Automated Deduction (CADE-20), 2005.

[BV04] L. Buttyán and I. Vajda, *Towards Provable Security for Ad Hoc Routing Protocols*, Proc. of the 2nd ACM workshop on Security of ad hoc and sensor networks (SASN'04), ACM, 2004, pp. 94–105.

[BYLM06] Sebastien Berton, Hao Yin, Chuang Lin, and Geyong Min, *Secure, disjoint, multipath source routing protocol(sdmsr) for mobile ad-hoc networks*, Proceedings of the Fifth International Conference on Grid and Cooperative Computing (Washington, DC, USA), GCC '06, IEEE Computer Society, 2006, pp. 387–394.

[CH03] Srdjan Capkun and Jean-Pierre Hubaux, *Biss: Building secure routing out of an incomplete set of security associations*, 2003.

[CKRT03] Yannick Chevalier, Ralf Küsters, Michael Rusinowitch, and Mathieu Turuani, *Deciding the security of protocols with diffie-hellman exponentiation and products in exponents*, Proceedings of the Foundations of Software Technology and Theoretical Computer Science FSTTCS'03, Springer, 2003, pp. 124–135.

[CKW09] Véronique Cortier, Steve Kremer, and Bogdan Warinschi, *A Survey of Symbolic Methods in Computational Analysis of Cryptographic Systems*, Rapport de recherche RR-6912, INRIA, 2009.

[CLCZ10] H. Comon-Lundh, V. Cortier, and E. Zalinescu, *Deciding security properties for cryptographic protocols. Application to key cycles*, ACM Transactions on Computational Logic (TOCL) **11** (2010), no. 4, 496–520.

[CS03] Hubert Comon-Lundh and Vitaly Shmatikov, *Intruder deductions, constraint solving and insecurity decision in presence of exclusive or*, Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science (LICS'03) (Ottawa, Canada), IEEE Computer Society Press, June 2003, pp. 271–280.

[CTR09] N. Chridi, M. Turuani, and M. Rusinowitch, *Decidable analysis for a class of cryptographic group protocols with unbounded lists*, Proc. of the 22nd IEEE Computer Security Foundations Symposium (CSF'09), 2009, pp. 277–289.

[Deg11] Jan Degrieck, *Réduction de graphes pour l'analyse de protocoles de routage sécurisés*, 2011.

[DKR09] Stéphanie Delaune, Steve Kremer, and Mark D. Ryan, *Verifying privacy-type properties of electronic voting protocols*, Journal of Computer Security **17** (2009), no. 4, 435–487.

[DLLT08] Stéphanie Delaune, Pascal Lafourcade, David Lugiez, and Ralf Treinen, *Symbolic protocol analysis for monoidal equational theories*, Information and Computation **206** (2008), no. 2-4, 312–351.

[DLMS99] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov, *Undecidability of bounded security protocols*, Proc. Workshop on Formal Methods and Security Protocols, 1999.

[DR08] T. Dierks and E. Rescorla, *The transport layer security (tls) protocol version 1.2*, 2008.

[DY81] D. Dolev and A. C. Yao, *On the security of public key protocols*, Proceedings of the 22nd Annual Symposium on Foundations of Computer Science (Washington, DC, USA), IEEE Computer Society, 1981, pp. 350–357.

[FGML09] Tao Feng, Xian Guo, Jianfeng Ma, and Xinghua Li, *UC-Secure Source Routing Protocol*, 2009.

[GN09] Jens Chr. Godskesen and Sebastian Nanz, *Mobility models and behavioural equivalence for wireless networks*, COORDINATION '09: Proceedings of the 11th International Conference on Coordination Models and Languages (Berlin, Heidelberg), Springer-Verlag, 2009, pp. 106–122.

[God06] Jens Chr. Godskesen, *Formal verification of the ARAN protocol using the applied pi-calculus*, Proceedings of the Sixth International IFIP WG 1.7 Workshop on Issues in the Theory of Security, 2006, pp. 99–113.

[God07] ______, *A calculus for mobile ad hoc networks*, COORDINATION, 2007, pp. 132–150.

[HFP98] R. Housley, W. Ford, and W. Polk, *X.509 certificate and CRL profile*, 1998, IETF standard, RFC 2459.

[HJP03] Yih-Chun Hu, David B. Johnson, and Adrian Perrig, *Sead: secure efficient distance vector routing for mobile wireless ad hoc networks*, Ad Hoc Networks **1** (2003), no. 1, 175–192.

[HPJ05] Y.-C. Hu, A. Perrig, and D. Johnson, *Ariadne: A Secure On-Demand Routing Protocol for Ad Hoc Networks*, Wireless Networks **11** (2005), 21–38.

[HPJ06] Yih-Chun Hu, A. Perrig, and D. B. Johnson, *Wormhole attacks in wireless networks*, Selected Areas in Communications, IEEE Journal on **24** (2006), no. 2, 370–380.

[JK91] Jean-Pierre Jouannaud and Claude Kirchner, *Solving equations in abstract algebras: a rule-based survey of unification*, Computational Logic. Essays in honor of Alan Robinson (Jean-Louis Lassez and Gordon Plotkin, eds.), The MIT press, Cambridge (MA, USA), 1991, pp. 257–321.

[JMB01] D. Johnson, D. Maltz, and J. Broch, *DSR: The Dynamic Source Routing Protocol for multi-hop wireless ad hoc networks*, Ad Hoc Networking, 2001, pp. 139–172.

[KHG06] Jiejun Kong, Xiaoyan Hong, and Mario Gerla, *Modeling ad-hoc rushing attack in a negligibility-based security framework*, WiSe '06: Proceedings of the 5th ACM workshop on Wireless security (New York, NY, USA), ACM, 2006, pp. 55–64.

[KT07] R. Küsters and T. Truderung, *On the Automatic Analysis of Recursive Security Protocols with XOR*, Proc. of the 24th Symposium on Theoretical Aspects of Computer Science (STACS 2007), LNCS, vol. 4393, Springer, 2007, pp. 646–657.

[KT09] J. Kim and G. Tsudik, *Srdp: Secure route discovery for dynamic source routing in manets*, Ad Hoc Netw. **7** (2009), no. 6, 1097–1109.

[KW04] R. Küsters and T. Wilke, *Automata-based Analysis of Recursive Cryptographic Protocols*, Proc. of the 21st Symposium on Theoretical Aspects of Computer Science (STACS 2004), LNCS, vol. 2996, Springer-Verlag, 2004, pp. 382–393.

[Low96] G. Lowe, *Breaking and fixing the Needham-Schroeder public-key protocol using FDR*, Proc. of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96), LNCS, vol. 1055, Springer-Verlag, 1996, pp. 147–166.

[LPM$^+$05] L. Lazos, R. Poovendran, C. Meadows, P. Syverson, and L. W. Chang, *Preventing wormhole attacks on wireless ad hoc networks: a graph theoretic approach*, Wireless Communications and Networking Conference, vol. 2, 2005, pp. 1193–1199 Vol. 2.

[Mar03] John D. Marshall, *An analysis of the secure routing protocol for mobile ad hoc network route discovery: Using intuitive reasoning and formal verification to identify flaws*, Tech. report, Florida State University, 2003.

[Mer07] Massimo Merro, *An observational theory for mobile ad hoc networks*, Electron. Notes Theor. Comput. Sci. **173** (2007), 275–293.

[MS01] J. K. Millen and V. Shmatikov, *Constraint solving for bounded-process cryptographic protocol analysis*, Proc. of the 8th ACM Conference on Computer and Communications Security (CCS'01), ACM Press, 2001, pp. 166–175.

[NH06] Sebastian Nanz and Chris Hankin, *A Framework for Security Analysis of Mobile Wireless Networks*, Theoretical Computer Science **367** (2006), no. 1, 203–227.

[Pau89] Lawrence C. Paulson, *The foundation of a generic theorem prover*, Journal of Automated Reasoning **5** (1989).

[Pau97] ______, *Mechanized proofs for a recursive authentication protocol*, Proc. of the 10th IEEE Computer Security Foundations Workshop, IEEE Computer Society Press, 1997, pp. 84–95.

[Pau98] L. C. Paulson, *The inductive approach to verifying cryptographic protocols.*, Journal of Computer Security **6** (1998), no. 1-2, 85–128.

[PBR99] Charles E. Perkins and Elizabeth M. Belding-Royer, *Ad-hoc on-demand distance vector routing*, Proc. 2nd Workshop on Mobile Computing Systems and Applications (WMCSA '99), 1999, pp. 90–100.

[PH02] P. Papadimitratos and Z. Haas, *Secure routing for mobile ad hoc networks*, Proc. SCS Communication Networks and Distributed Systems Modelling Simulation Conference (CNDS), 2002.

[PPB10] Mihai-Lica Pura, Victor-Valeriu Patriciu, and Ion Bica, *Formal verification of secure ad hoc routing protocols using avispa: Aran case study*, Proceedings of the 4th conference on European computing conference (Stevens Point, Wisconsin, USA), ECC'10, World Scientific and Engineering Academy and Society (WSEAS), 2010, pp. 200–206.

[PPH08] Marcin Poturalski, Panos Papadimitratos, and Jean-Pierre Hubaux, *Towards Provable Secure Neighbor Discovery in Wireless Networks*, Proceedings of the 6th ACM workshop on Formal methods in security engineering, ACM, 2008, pp. 31–42.

[PPS$^+$08] P. Papadimitratos, M. Poturalski, P. Schaller, P. Lafourcade and D. Basin, S. Čapkun, and J.-P. Hubaux, *Secure Neighborhood Discovery: A Fundamental Element for Mobile Ad Hoc Networking*, IEEE Communications Magazine **46** (2008), no. 2, 132–139.

[PSW$^+$02] Adrian Perrig, Robert Szewczyk, Victor Wen, David Culler, and J. D. Tygar, *SPINS: Security protocols for sensor networks*, Wireless Networks **8** (2002), no. 5, 521–534.

[RT01] M. Rusinowitch and M. Turuani, *Protocol insecurity with finite number of sessions is NP-complete*, Proc. of the 14th Computer Security Foundations Workshop (CSFW'01), IEEE Computer Society Press, 2001, pp. 174–190.

[SDL$^+$02] Kimaya Sanzgiri, Bridget Dahill, Brian Neil Levine, Clay Shields, and Elizabeth M. Belding-Royer, *A secure routing protocol for ad hoc networks*, 10th IEEE International Conference of Network Protocols (ICNP), 2002.

[SKY05] Ronggong Song, Larry Korba, and George Yee, *Anondsr: efficient anonymous dynamic source routing for mobile ad-hoc networks*, SASN '05: Proceedings of the 3rd ACM workshop on Security of ad hoc and sensor networks, ACM, 2005, pp. 33–42.

[THG99] F. Javier Thayer, Jonathan C. Herzog, and Joshua D. Guttman, *Strand spaces: Proving security protocols correct.*, Journal of Computer Security **7** (1999), no. 1.

[Tru05] T. Truderung, *Selecting theories and recursive protocols*, Proc. of the 16th International Conference on Concurrency Theory (CONCUR'05), LNCS, vol. 3653, Springer, 2005, pp. 217–232.

[vig10] *Model checking ad hoc network routing protocols: ARAN vs. endairA*, IEEE Computer Society, 2010.

[YB03] Shahan Yang and John S. Baras, *Modeling vulnerabilities of ad hoc routing protocols*, Proc. 1st ACM Workshop on Security of ad hoc and Sensor Networks (SASN'03), 2003.

[ZA02] Manel Guerrero Zapata and N. Asokan, *Securing ad hoc routing protocols*, Proc. 1st ACM workshop on Wireless SEcurity (WiSE'02), ACM, 2002, pp. 1–10.

[ZWK$^+$04] Bo Zhu, Zhiguo Wan, Mohan S. Kankanhalli, Feng Bao, and Robert H. Deng, *Anonymous secure routing in mobile ad-hoc networks*, LCN, 2004, pp. 102–108.