université PARIS-SACLAY

école normale supérieure paris–saclay

NNT : 2017SACLN019

# Thèse de doctorat de l'Université Paris-Saclay préparée à l'École normale supérieure de Cachan (École normale supérieure Paris-Saclay)

Ecole doctorale n°580
Sciences et Technologies de l'Information et de la Communication
Spécialité de doctorat : Informatique

par

M. Lucca Hirschi

## Vérification automatique de la protection de la vie privée: entre théorie et pratique

Thèse présentée et soutenue à Cachan, le 21 avril 2017.

Composition du Jury :

| | | | |
|---|---|---|---|
| M. | David Baelde | Maître de conférences<br>ENS Paris-Saclay | (Co-directeur de thèse) |
| M. | Gilles Barthe | Professeur<br>IMDEA Software Institute | (Rapporteur) |
| M. | Bruno Blanchet | Directeur de recherche<br>INRIA | (Rapporteur) |
| M. | Cas Cremers | Professeur<br>Université d'Oxford | (Examinateur) |
| Mme. | Stéphanie Delaune | Chargée de recherche<br>CNRS | (Directrice de thèse) |
| M. | Thomas Jensen | Directeur de recherche<br>INRIA | (Président du jury) |
| Mme. | Catuscia Palamidessi | Directrice de recherche<br>INRIA | (Examinatrice) |

# Résumé en Français

La société de l'information dans laquelle nous vivons repose notamment sur notre capacité à échanger des informations de façon sécurisée. Ces échanges sécurisés sont réalisés au moyen de *protocoles cryptographiques*. Ils explicitent comment les différents agents communicants doivent se comporter et exploitent des primitives cryptographiques (e.g. chiffrement, signature) pour protéger les échanges. Étant donné leur prédominance et leur importance, il est crucial de s'assurer que ces protocoles accomplissent réellement leurs objectifs. Parmi ces objectifs, on trouve de plus en plus de propriétés en lien avec la *vie privée* (e.g. anonymat, non-traçabilité). Malheureusement, les protocoles développés et déployés souffrent souvent de défauts de conception entraînant un cycle interminable entre découverte d'attaques et amélioration des protocoles.

Pour en sortir, nous prônons l'analyse mécanisée de ces protocoles par des *méthodes formelles* qui, via leurs fondements mathématiques, permettent une analyse rigoureuse apportant des garanties fortes sur la sécurité attendue. Parmi celles-ci, la vérification dans le *modèle symbolique* offre de grandes opportunités d'automatisation. La plupart des propriétés en lien avec la vie privée sont alors modélisées par l'équivalence entre deux systèmes. Toutefois, vérifier cette équivalence est indécidable dans le cas général. Deux approches ont alors émergé. Premièrement, pour un nombre borné de sessions d'un protocole, il est possible de symboliquement explorer toutes ses exécutions possibles et d'en déduire des procédures de décision pour l'équivalence. Deuxièmement, il existe des méthodes de semi-décision du problème dans le cas général qui exploitent des abstractions, notamment en considérant une forme forte d'équivalence.

Nous avons identifié un problème majeur pour chacune des deux méthodes de l'état de l'art qui limitent grandement leur impact en pratique. Premièrement, les méthodes et outils qui explorent symboliquement les exécutions souffrent de l'explosion combinatoire du nombre d'états, causée par la nature concurrente des systèmes étudiés. Deuxièmenent, dans le cas non borné, la forme forte d'équivalence considérée se trouve être trop imprécise pour vérifier certaines propriétés telle que la non traçabilité, rendant cette approche inopérante pour ces propriétés.

Dans cette thèse, nous proposons une solution à chacun des problèmes. Ces solutions prennent la forme de contributions théoriques, mais nous nous efforçons de les mettre en pratique via des implémentations afin de confirmer leurs impacts pratiques qui se révèlent importants.

Tout d'abord, nous développons des *méthodes de réduction d'ordres partiels* pour réduire

drastiquement le nombre d'états à explorer tout en s'assurant que l'on ne perd pas d'attaques. Nos méthodes sont conçues pour le cadre exigeant de la sécurité et sont prouvées correctes et complètes vis-à-vis de l'équivalence. Nous montrons comment elles peuvent s'intégrer naturellement dans les outils existants. Nous prouvons la correction de cette intégration dans un outil et proposons une implémentation complète. Finalement, nous mesurons le gain significatif en efficacité ainsi obtenu et nous en déduisons que nos méthodes permettent l'analyse d'un plus grand nombre de protocoles.

Ensuite, pour résoudre le problème de précision dans le cas non-borné, nous proposons une nouvelle démarche qui consiste à assurer la *vie privée via des conditions suffisantes*. Nous définissons deux propriétés qui impliquent systématiquement la non-traçabilité et l'anonymat et qui sont facilement vérifiables via les outils existants (e.g. ProVerif). Nous avons implémenté un nouvel outil qui met en pratique cette méthode résolvant le problème de précision de l'état de l'art pour une large classe de protocoles. Cette nouvelle approche a permis les premières analyses de plusieurs protocoles industriels incluant des protocoles largement déployés, ainsi que la découverte de nouvelles attaques.

# Abstract

The information society we belong to heavily relies on secure information exchanges. To exchange information securely, one should use *security protocols* that specify how communicating agents should behave notably by using cryptographic primitives (*e.g.* encryption, signature). Given their ubiquitous and critical nature, we need to reach an extremely high level of confidence that they actually meet their goals. Those goals can be various and depend on the usage context but, more and more often, they include *privacy properties* (*e.g.* anonymity, unlinkability). Unfortunately, designed and deployed security protocols are often flawed and critical attacks are regularly disclosed, even on protocols of utmost importance, leading to the never-ending cycle between attack and fix.

To break the present stalemate, we advocate the use of *formal methods* providing rigorous, mathematical frameworks and techniques to analyse security protocols. One such method allowing for a very high level of automation consists in analysing security protocols in the *symbolic model* and modelling privacy properties as equivalences between two systems. Unfortunately, deciding such equivalences is actually undecidable in the general case. To circumvent undecidability, two main approaches have emerged. First, for a bounded number of agents and sessions of the security protocol to analyse, it is possible to symbolically explore all possible executions yielding decision procedures for equivalence between systems. Second, for the general case, one can semi-decide the problem leveraging dedicated abstractions, notably relying on a strong form of equivalence (*i.e.* diff-equivalence).

The two approaches, *i.e.* decision for the bounded case or semi-decision for the unbounded case, suffer from two different problems that significantly limit their practical impact. First, (symbolically) exploring all possible executions leads to the so-called *states space explosion problem* caused by the concurrency nature of security protocols. Concerning the unbounded case, diff-equivalence is actually too imprecise to meaningfully analyse some privacy properties such as unlinkability, nullifying methods and tools relying on it for such cases.

In the present thesis, we address those two problems, going back and forth between theory and practice. Practical aspects motivate our work but our solutions actually take the form of theoretical developments. Moreover, we make the effort to confirm practical relevance of our solutions by putting them into practice (implementations) on real-world case studies (analysis

of real-world security protocols).

First, we have developed new *partial order reduction techniques* in order to dramatically reduce the number of states to explore without loosing any attack. We design them to be compatible with equivalence verification and such that they can be nicely integrated in frameworks on which existing procedures and tools are based. We formally prove the soundness of such an integration in a tool and provide a full implementation. We are thus able to provide benchmarks showing dramatic speedups brought by our techniques and conclude that more protocols can henceforth be analysed.

Second, to solve the precision issue for the unbounded case, we propose a new methodology based on the idea to ensure *privacy via sufficient conditions*. We present two conditions that always imply unlinkability and anonymity that can be verified using existing tools (*e.g.* ProVerif). We implement a tool that puts this methodology into practice, hence solving the precision issue for a large class of protocols. This novel approach allows us to conduct the first formal analysis of some real-world protocols (some of them being widely deployed) and to discover some novel attacks.

# Remerciements

Je veux tout d'abord remercier mes directeurs de thèse David Baelde et Stéphanie Delaune qui m'ont donné l'opportunité de commencer cette thèse et par qui j'ai appris tant de choses. C'est d'abord grâce à vous que ces quelques années au LSV ont été si enrichissantes.

Je tiens aussi à remercier les deux rapporteurs de ma thèse Bruno Blanchet et Gilles Barthe pour avoir lu et commenté mon manuscrit ainsi que Cas Cremers, Thomas Jensen et Catuscia Palamidessi pour avoir accepté de faire partie de mon jury de thèse.

J'ai eu la chance de travailler dans un milieu où l'échange et le partage ont un rôle primordial. Merci donc à tous ceux qui ont pris part à ce bouillonnement d'idées si stimulant que ça soit à Cachan ou ailleurs. Notamment, merci aux doctorants et autres membres du LSV de faire en sorte que ce bouillonement ne s'éteigne jamais.

Je veux également remercier mes amis qui m'entourent : que l'on se soit connu à Avignon, Lyon, Paris, ou ailleurs, merci à vous tous d'être toujours là. Merci à ma famille qui m'a enraciné dans ce qu'il y a de plus important. Merci à Marie-Charlotte pour m'avoir supporté ces quelques derniers mois et pour son amour ces quelques dernières années.

# Contents

Chapter 1

# General Introduction

## 1.1 General Context

Historically, *cryptography* exclusively belonged to the military and diplomatic domain: from the ancient Greeks using scytales[1] to the Enigma machine notably used by nazi Germany before and during World War II and broken by Alan Turing *et al.*

**Figure 1.1** A scytale and an Enigma machine

The context has changed during the last century with the rise of the *information society* replacing the previously in place industrial society. Since the – commonly termed – *information revolution*, information is at the heart of our society, be it in our economy (*e.g.* tertiary industry gradually replacing primary industry, emergence and rise of quaternary industry[2]), political or cultural activities. As the main tool to secure exchanges and storage of information, cryptography has, very naturally, flood into our everyday life. It is now so ubiquitous that it appears to be one of the most critical piece of technology in our society's foundations alongside other Information Technologies (ITs).

**Security Protocols.** More specifically, to secure exchanges of information, one should use *security protocols* (also called *cryptographic protocols*). As all communication protocols, security

[1] A scytale (see Figure 1.1) is a rod around which a parchment is wrapped to read and write the message (in the axis of the rod). The diameter of the rod constitutes the key to cipher and decipher. Indeed, when wrapped around a rod of a different diameter, the message is unreadable.

[2] This commonly refers to the knowledge-based part of the economy.

protocols specify how different agents may exchange information through a communication channel (*e.g.* the Internet, the air) via a set of rules. However, security protocols additionally make use of *cryptographic primitives*, such as encryption or signature for example, to protect exchanged messages. To illustrate the wide range of contexts where security protocols are critical, we now give a short selection of concrete examples (see also illustrations in Figure 1.2).


**Figure 1.2** Some modern manifestations of cryptography

Obviously, one can find a tremendous amount of security protocols aiming at securing the Internet. For instance, the Transport Layer Security (TLS) protocol which is mainly used by websites (*e.g.* bank websites) to secure all communications (*i.e.* HTTPS traffic) between their servers and web browsers, which then display a green lock in the address bar. As announced recently [EFF16], more than half of page loads in Firefox and in Chrome are now secured with TLS making this protocol one of the most popular ones. Security protocols are also intensely used to secure services accessible through the Internet. For instance, Single Sign-On (SSO) protocols enable users of companies such as Facebook or Google to sign in just once to their respective websites and yet are able to access other (often third-party) services without signing in again. Such a protocol is notably used by Google to share users' credentials across all Google Applications.

However, security protocols are not confined to the Internet. For instance, mobile phones also heavily rely on them. The 3rd Generation Partnership Project (3GPP) has notably standardised stacks of security protocols protecting 2G, 3G and 4G wireless mobile telecommunication technologies. For example, 3GPP designed the Authentication and Key Agreement (AKA) protocol – equipping all 3G and 4G-capable mobile phones – which aims at establishing a secure channel

between a mobile phone and an antenna (*i.e.* serving network) for subsequent communications (*e.g.* calls, SMSs).

More recently, came to the market a wide range of wireless devices such as radio-frequency identification (RFID) tags embedded in goods in supermarkets, RFID chips in credit cards enabling wireless payments, wireless keys for opening cars or garage doors, RFID chips embedded in e-passport for improving its security. Not to mention the rise of the Internet of Things (IoT), from the smart lights and heating systems at home, to cars connected to the Internet. Examples of security protocols used in those devices are countless. Let us go a bit more into the details of two examples mentioned above. First, the International Civil Aviation Organisation has designed the Basic Access Control (BAC) protocol which now equips all biometric passports in the world. Similarly to AKA, the purpose of the BAC protocol is to establish a secure link between an e-passport and an authorised reading device (*e.g.* in customs offices at airports). Second, distance bounding protocols are used *e.g.* in wireless keys and rely on mechanisms to measure the delay between an emission and a reception to bound the physical distance between the key and the door to open.

Maybe more surprisingly, we also find security protocols at the heart of e-voting systems and notably in end-to-end voter verifiable systems. Such systems should obviously protect the vote privacy but also must achieve an antagonist goal: every manipulation should be verifiable by all voters in order to completely relocate the trust from the different agents and the architecture to the protocol itself. One example of such protocols is Helios [ADMP+09] which has been used in real elections by the International Association of Cryptologic Research [BVQ10], the Princeton University [hel], and the Catholic University of Louvain [PAdM10]. A more recent example is Du-Vote [GRCC15] that has been designed to be used securely even from an infected laptop thanks to an additional hardware token.

**Security Goals.** As shown by the previous examples, security protocols are used in a wide range of contexts. Naturally, they also have to achieve various requirements that we call *security goals*. For instance, when a customer using his bank account online sees the green lock in his web browser's address bar, he expects that the TLS protocol ensures that all his HTTP traffic (*e.g.* credentials, account balances) remains private from anybody but the bank account's web server (excluding *e.g.* his Internet service provider, the host of the Wi-Fi hot-spot, anybody sniffing Wi-Fi traffic around). This high-level requirement actually involves at least two security goals. First, the traffic encrypted by TLS should remain *secret*. Secrecy is maybe the most common security goal but ensuring it is already not that trivial. For instance, a bad management of encryption keys may be exploited by an attacker to break the secrecy of some messages. Furthermore, secrecy is not sufficient in general. Indeed, if able to impersonate the bank's web server, an attacker could fake the bank's website to get customer's credentials. Hence, the customer's web browsers must make sure that the server it is communicating with is indeed the bank's web server. Conversely, the bank's web server has to be convinced that it is indeed communicating with the expected customer. Otherwise, an attacker could access to *e.g.* customers' account balances. This security goal is called *mutual authentication*: each party is able to correctly authenticate

the other party. Similarly for the wireless keys example, the underlying distance bounding protocols shall achieve authentication of the key: *i.e.* the door opens only when requested by genuine authorised keys.

More recently, privacy-related security goals (we shall call them *privacy goals*) are becoming more and more predominant and critical. This could be explained by a combination of different factors. First, as a consequence of the information revolution, the gigantic quantity of private information created routinely (deliberately or not) and its continuing circulation (*e.g.* with advent of wireless devices) understandably raises privacy concerns. Second, citizens and customers are now more and more aware of privacy issues maybe because of the above, the countless recent hacks, revelations about state-scale surveillance and GAFAM's[3] monopoly. Last but not least, and surely with causal links with the previous, regulatory requirements about privacy protection are becoming increasingly strict. For instance, the general data protection regulation (GDPR), taking effect on May 25 2018 in EU and its partner states, will impose strong privacy requirements on companies that control sensitive personal information [GDP]. In France, the "commission nationale de l'informatique et des libertés" (CNIL) (*i.e.* the national data protection authority) will have authority, once the GDPR takes effect, to impose fines to companies that can reach 20 millions of euros or 4% of their annual worldwide turnovers.

Let us illustrate some privacy goals previous examples shall meet. For instance, the AKA protocol implemented in mobile phones must not enable ill-intentioned people to trace specific mobile phones by wirelessly communicating with them[4]. Similarly, we expect to not be traceable just because we carry on an e-passport implementing the BAC protocol. This privacy goal is called *unlinkability* (or *untraceability*). Another example of privacy goal naturally arises in e-voting systems, where *vote privacy* is one of the most essential property such systems must achieve. We will discuss later on other privacy goals such as anonymity, coercion-resistance, etc.

**Critical Attacks.** Given the sensitivity of the exchanged data, security protocols' societal and economical[5] significance, we need to reach an extremely high level of confidence that security protocols actually meet their security goals. Given this context and despite the appropriate care taken in designing those protocols, it is astonishing to regularly observe the contrary: highly critical attacks on security protocols of utmost importance are regularly found, disclosed and sometimes exploited. Let us discuss some of them which have broken the aforementioned security protocols.

The TLS protocol has been repeatedly broken. The recently disclosed LogJam [ABD+15, log] attacks notably rely on a downgrade attack: a Man-In-The-Middle (MITM) attacker is able to force a client and a web server to negotiate the use of weak cipher suites which can then be broken online. Indeed, many web servers continue to propose weak cipher suites designed in the

[3]GAFAM (or Big Five) refers to the most influential international technology companies: Google, Apple, Facebook, Amazon and Microsoft.

[4]There are examples of companies (*e.g.* Navizon [nava]) selling tools to trace people exploiting traceability attacks (see the survey [DCL14]). Such tools have for example been used by supermarkets to trace customers through their shelves [navb, Tur17].

[5]For instance, when researchers disclosed an attack [car] to remotely take control over Jeep Cheroke cars, Fiat Chrysler had to recall 1.54 million vehicles to fix the issue.

90s when regulation standards in the U.S. were limiting the size of the keys. As a result, such an attacker can completely impersonate a web server to a client (*e.g.* fake a bank website despite the reassuring green lock). Another attack on deployed TLS versions, called 3SHAKE [BLF+14, 3sh], has been disclosed one year earlier. It enables an attacker to impersonate a client to a server breaking also the mutual authentication security goal. This could be used to steal credentials from users and use them later *e.g.* on a bank website. A similar attack has been found [CHSvdM16] in proposals for the next versions of TLS (*i.e.* TLS 1.3) which is currently being designed by the TLS working group.

Transport layer protocols are not the only components that put the Internet at risk. Indeed, web-services themselves are also often flawed. For instance, the SAML-based SSO protocol deployed in Google Applications has also been broken [ACC+08]. Researchers have shown how a malicious third-party service could use credentials whose purpose was supposed to be limited to this service to impersonate users to other services (*e.g.* Google Gmail).

The situation does not improve when it comes to mobile phones security. The AKA protocol (still) suffers from several different traceability attacks: from the well-known IMSI catcher attacks [SSB+16, vdBVdR15] to a more elaborate attack [ACRR10] where leaks of error messages can be exploited by an attacker to trace people who carry mobile phones.

RFID devices also often suffer from traceability attacks. For instance, researchers [ACRR10] have shown that the BAC protocol could be exploited by ill-intentioned people to trace citizens who carry their e-passport by wirelessly communicating with the RFID chips they contain without the victims being able to notice anything. In the next section, we will go into more details of this protocol and its attacks. Distance bounding protocols have also been broken. We could for example cite several distance hijacking attacks [CRSv12] (*i.e.* when an attacker is able to pretend to be physically close by exploiting honest users) on several distance bounding protocols.

Finally, despite the highly sensitive nature of e-voting protocols, attacks are also found there. Hence, several critical attacks [K+16] have been disclosed on the Du-Vote e-voting system and Helios has been shown [CS13] to fail to entirely meet vote privacy, one of its most essential security goal.

All those attacks, as many others we do not cite, break the protocols' specifications themselves and do not rely on the attacker being able to break or exploit flaws in cryptographic primitives (*e.g.* fake a signature, decrypt a ciphertext without knowing the decryption key). Such attacks neither rely on bad hardware or bad implementation of the protocol's specification. They rather rely on possible interactions of the different parties (but not the ones one would expect from normal usages of the protocol). We say that they are *logical attacks*. Interestingly, any implementation and any deployment on any hardware of a protocol whose specification suffers from a logical attack would be broken in practice as well. With this in mind, we now state the following natural question:

> How to explain that such critical logical attacks are regularly found on specifications of security protocols of utmost importance despite the high care given to their design ?

An important part of the answer is that designing protocols which actually achieve the required security goals is a tremendously complex task. In this thesis, we are interested in improving this current distressing situation by enhancing rigorous mathematical frameworks to analyse security protocols and ease their design.

The rest of the introduction is organised as follows. In the next section, we explain more precisely what are security protocols and logical attacks. Thanks to those additional details, we will then be able to answer the above question in Section 1.3 by giving the reasons why the security setting is particularly complex. This will motivate the need for rigorous, formal methods to model and analyse security protocols. Next, the state-of-the-art of such methods will be given in Section 1.4. Finally, we show that this state-of-the-art suffers from two main limitations in Section 1.5 before explaining how our contributions address them in Section 1.6. We conclude with the outline of the rest of this thesis in Section 1.7.

## 1.2 Security Protocols

A security protocol can be seen as a set of rules that participants have to follow in order to achieve specific security goals. Those rules specify emissions and receptions of messages as well as tests performed by the participants of the protocols called *agents*. Security protocols use as basic building blocks cryptographic primitives such as symmetric and asymmetric encryptions, signatures, and hash functions. For a long time, it was believed that designing a strong encryption scheme was sufficient to ensure secure message exchanges. Starting from the 80's, researchers understood that even with perfect encryption schemes, message exchanges were still not necessarily secure: indeed protocols are still subject to logical attacks such as the ones discussed before. We will conclude this section with in-depth descriptions of two such logical attacks, but we first briefly explain the most standard cryptographic primitives together with their fundamental properties.

### 1.2.1 Cryptographic Primitives

We list below the most standard cryptographic primitives. Obviously this list is not exhaustive, and modern protocols often rely on less standard cryptographic primitives, such as blind signature, homomorphic encryption, trapdoor bit commitment, zero-knowledge proofs, etc.

**Symmetric encryption.** Symmetric cryptography refers to encryption methods in which both the sender and the receiver share the same key. For instance, the Data Encryption Standard (DES) and the Advanced Encryption Standard (AES) are symmetric encryption schemes which have been designated cryptography standards by the US government in 1976 and 2002 respectively. The scytale (see Figure 1.1) is an example of primitive symmetric encryption scheme.

A significant disadvantage of symmetric ciphers is the key management necessary to use them securely. Each distinct pair of agents must share a different key. Therefore, the number of required keys increases as the square of the number of agents, which very quickly requires complex key management schemes to keep them all straight and secret. The difficulty of securely

establishing a secret key between two agents, when a secure channel does not already exist between them, also presents a chicken-and-egg problem which is a considerable practical obstacle for the use of symmetric cryptography in the real world. This is why the recourse to asymmetric cryptography is so popular for key establishment protocols that aim to establish a fresh symmetric key between two parties.

**Asymmetric encryption.** In 1976, Diffie and Hellman proposed the notion of public key cryptography, in which two different but mathematically related keys are used – a *public key* and a *private key*. A public key system is constructed, in such a way that calculation of the private key is computationally infeasible from the public key, even though they are necessarily related. In public key cryptosystems, the public key may be freely distributed, while its associated private key must remain secret. The public key is typically used for encryption, while the private key is used for decryption. Diffie and Hellman showed that public key cryptography was possible[6] by presenting the Diffie-Hellman key exchange protocol [DH76] in 1976. In 1978, Rivest, Shamir, and Adleman invented RSA, another public key cryptosystem [RSA78] which has established itself as the main standard.

**Digital signature.** Over the same period, signature schemes have also been proposed. A digital signature is a mathematical scheme for demonstrating the authenticity of a digital message or of a document. It gives the recipient a reason to believe that the message was created by a known sender, that the sender cannot deny having sent the message (authentication and non-repudiation), and that the message was not altered while in transit (integrity). Digital signatures are commonly used for software distribution, key management, financial transactions, etc.

**Hash function.** A hash function takes a message of any length as input, and outputs a short, fixed length digest of this message. Hash functions have many security applications, notably in digital signatures, message authentication codes (MACs), and other forms of authentication. They can also be used as checksums to detect accidental data corruption. For good hash functions, an attacker cannot find two messages that produce the same hash. MACs are much like hash functions, except that a secret key can be used to authenticate the hash value upon receipt.

### 1.2.2 The Example of the BAC Protocol

For the purpose of illustration, we go into the details of the already mentioned BAC protocol used in e-passports. An e-passport is a paper passport with a RFID chip that stores the critical information printed on the passport in order to increase its security (*e.g.* anti-cloning, integrity). The International Civil Aviation Organisation (ICAO) standard specifies the communication protocols that are used to access this information [ICA04]. We do not plan to describe all the protocols that are specified in the standard. Instead, we shall focus only on the BAC protocol following the modelling proposed in [ACRR10].

[6] Diffie and Hellman have been rewarded by the ACM Turing Award in 2016 for having laid the foundations of asymmetric encryption.

The information stored in the chip is organised in data groups ($dg_1$ to $dg_{19}$). For example, $dg_5$ contains a JPEG copy of the displayed picture, and $dg_7$ contains the displayed signature. The verification key $\mathsf{vk}(sk_P)$ of the e-passport, together with its certificate $\mathsf{sign}(\mathsf{vk}(sk_P), sk_{DS})$ issued by the Document Signer Authority, is stored in $dg_{15}$. The corresponding signing key $sk_P$ is stored in a tamper resistant memory, and cannot be read or copied. For authentication purposes, a hash of all the data groups, together with a signature on this hash value issued by the Document Signer Authority, are stored in a separate file, the Security Object Document:

$$sod \stackrel{\mathsf{def}}{=} \langle \mathsf{sign}(\mathsf{h}(dg_1, \ldots, dg_{19}), sk_{DS}),\ \mathsf{h}(dg_1, \ldots, dg_{19}) \rangle.$$

The ICAO standard specifies several protocols through which this information can be accessed. In particular, read access to the data on the e-passport is protected by the BAC protocol.


**Figure 1.3** Password of low entropy in some e-passport

The BAC protocol is a password-based authenticated key exchange (PAKE) protocol whose security relies on two master keys, namely $ke$ and $km$, which are derived from a password of low entropy (see Figure 1.3) optically retrieved from the e-passport by the reader before executing the protocol. Through the BAC protocol, the reader and the e-passport agree on a key seed $xkseed$ that is then used to generate an encryption session key as well as a MAC session key for the next protocols. Following the description given in Figure 1.4, the protocol works as follows:

1. The reader sends a request $\mathsf{get\_Challenge}$ to the e-passport that will answer by generating a nonce, *i.e.* a fresh random number.

2. Once the reader receives this nonce $n_T$, it will generate its own nonce $n_R$, as well as a key $k_R$ that will be used later on to derive session keys. The reader encrypts the nonce $n_T$, its own nonce $n_R$ as well as the key $k_R$ with the (long-term) symmetric encryption key $ke$. This message $\mathsf{senc}(\langle n_R, \langle n_T, k_R \rangle \rangle, ke)$ is sent to the e-passport together with the associated MAC (with key $km$) to ensure that the encryption will be correctly transmitted to the e-passport.

3. The e-passport performs some checks not shown in Figure 1.4. In particular, it checks that the MAC has been computed using the right key $km$, and that the nonce $n_T$ it has generated at the first step of the protocol is part of the plaintext inside the encryption.

Once these checks have been performed, the e-passport sends to the reader a message similar to the one it has received using its own contribution $k_T$.

4. Again, the reader will perform the necessary checks before accepting the message, and in case of success two session keys will be generated from the value $xkseed$ obtained by applying the exclusive or operator on $k_R$ and $k_T$.

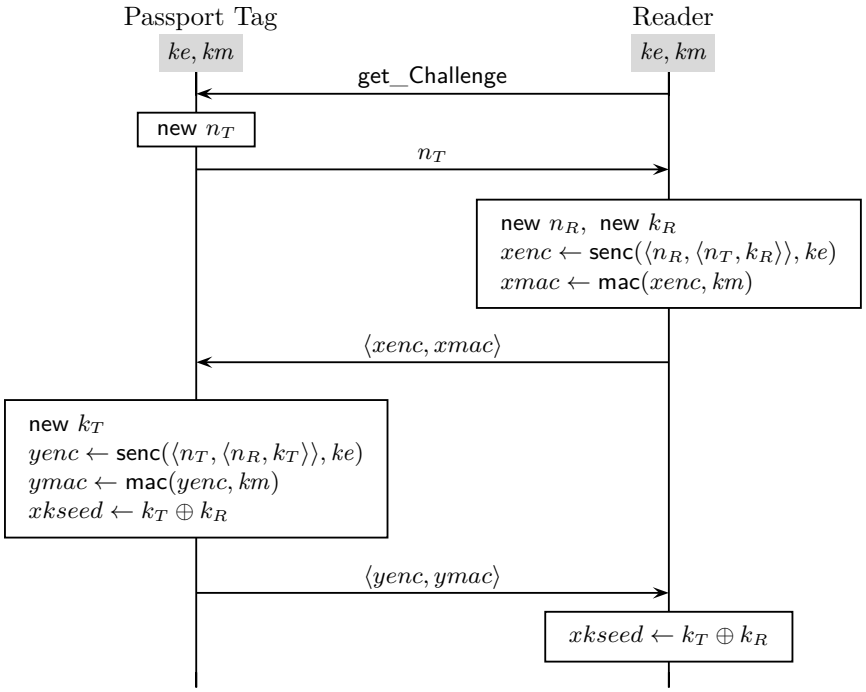


**Figure 1.4** Basic Access Control protocol

These two session keys are used to provide confidentiality, integrity, and authentication in subsequent communications. In particular, they are used to encrypt and MAC the messages exchanged during the execution of the Passive and Active Authentication protocols in order to ensure that only parties with physical access to the e-passport[7] can read the data. The aim of establishing fresh session keys (instead of reusing $ke$ and $km$ at each session) is to notably make the e-passport *unlinkable*, a property that will be discussed in Section 1.2.3.

### 1.2.3 Some Logical Attacks on the BAC Protocol

Now, we describe two possible attacks. These attacks are purely logical in the sense that they do not require to break any cryptographic primitives.

[7]They can thus retrieve the low-entropy password.

**Authentication issue (for a weakened version).** First, we would like to pinpoint the fact that the order in which the nonces $n_R$ and $n_T$ have been placed inside both encryptions is relevant. The careful reader will have noticed that the nonces have been swapped: the reader encrypts $\langle n_R, \langle n_T, k_R\rangle\rangle$ whereas the e-passport encrypts $\langle n_T, \langle n_R, k_T\rangle\rangle$. The purpose of this design is to avoid a *replay attack*. Indeed, without such a swap, a malicious user (who does not know the keys $ke$ and $km$) would be able to simply replay the message he received from the reader without decrypting it and performing the checks. Such a message will be accepted by the reader and pass all checks performed by the reader. This means that the reader will end its session thinking (s)he has talked with the e-passport identified by $ke$ and $km$, whereas this e-passport will not have really taken part to the protocol. Moreover, the key seed computed at the end will be $k_R \oplus k_R = 0$ and thus very different from the one that is supposed to be computed during a normal execution.

**Unlinkability issue.** Following the specification provided by the ICAO, each nation has implemented its own version of the BAC protocol. Unfortunately, as the specification is not completely comprehensive, each nation's e-passport has subtle differences. In particular, the standard specifies that the e-passport must reply with an error message to every ill formed or incorrect message from the reader, but it does not specify what the error message should be.

For example, in the French implementation of e-passports issued before 2010, the e-passport replies different error messages [ACRR10] depending on whether the nonce in $xenc$ is not $n_T$ (*i.e.* nonce error) or $xmac$ is not a correct MAC w.r.t. the key $km$ (*i.e.* MAC error). An attacker who does not know the keys $ke$ and $km$ could then trace an e-passport in the following way:

1. He eavesdrops a session between an authentic reader and an e-passport $P_{\text{Alice}}$ (with keys $ke$ and $km$) and stores $m = \langle xenc, xmac\rangle$.
2. In a different session, he sends the message $m$ and waits for the e-passport's answer.
3. Then, we distinguish two cases:
   a) if he receives a nonce error then he knows that the e-passport succeeded to check the MAC and so this e-passport is $P_{\text{Alice}}$;
   b) if he receives a MAC error then he knows that the e-passport is not the one with keys $km$ (and $ke$), and therefore it is not $P_{\text{Alice}}$.

This attack makes it possible to detect when a particular e-passport comes into the range of a reader, which could be, for instance, placed by a doorway, in order to monitor when a target enters or leaves a particular building[8]. To avoid the information leakage of these error messages, the specification should prescribe that, in case of failure, the e-passport yields the same message in both situations (as it is done for instance in e-passports from the UK).

[8]This is not a paranoid scenario, remember the already mentioned tools which are sold *e.g.* by Navizon [nava] to trace people by exploiting traceability attacks.

**Discussion.** We may note that in presence of honest participants who follow the protocol rules, the protocol works well, and the attack scenarios described above are not possible. However, it is important to ensure that these protocols meet their security goals in any situation, especially in the presence of malicious agents that may want to take advantage of the protocol and therefore do not necessarily follow the instructions specified by the protocols. Making sure that security protocols are secure in such a hostile environment is an essential aspect which makes protocol design and verification very difficult tasks.

## 1.3 Formal Verification

Now that we better grasp what are security protocols and logical attacks, we go back to our initial question and try to answer it.

**Why is designing good security protocols so hard ?** Compared to critical or embedded software, that we routinely rely on (*e.g.* for flying planes), making sure that security protocols actually meet their goals is more involved for several reasons.

First, as illustrated by logical attacks from the previous section, it is necessary to make sure that security protocols are able to cope with malicious agents that do not necessarily follow the expected instructions. In particular, security protocols have to be resistant to MITM attackers who can actively exploit the insecure network on which agents are trying to exchange messages. Already in 1978, this was considered the most reasonable threat model on the network as acknowledged by Needham and Schroeder [NS78] (for the case of authentication protocols): *"We assume that the intruder can interpose a computer in all communication paths, and thus can alter or copy parts of messages, replay messages, or emit false material. While this may seem an extreme view, it is the only safe one when designing authentication protocols."* This complex threat model also explains why fuzzing or testing are not powerful enough for this setting: even a very unlikely corner case that will most likely be missed by those approaches could be exploited systematically by a clever attacker to break a security goal.

Second, security protocols are essentially distributed programs. Indeed, the different agent's programs are running concurrently. This makes the set of possible interactions very large since one needs to examine all possible interleavings of agents' actions. Indeed, expected security goals must hold for any interleaving. Otherwise, the attacker may force agents to follow the specific scheduling enabling the attack for instance by delaying some receptions of messages.

> To sum up, we face a very intricate problem here: we shall achieve rather complex security goals using security protocols that will be executed *concurrently* by all the different agents involved. Moreover, those agents will exchange messages through an *insecure network* that should be considered to be *entirely controlled by the attacker*.

The combination of all those features makes potential attacks very subtle and extremely hard to find. Furthermore, there are so many possible interactions with potential hidden corner cases to

analyse that security researchers and engineers will most likely miss some of them yielding potential disastrous attacks. To break the present stalemate, we advocate the use of formal methods providing rigorous, mathematical frameworks and techniques to analyse security protocols. The security community (academics but not only) now agrees upon the importance of formal methods also as a guide for designing security protocols. For instance, the TLS Working Group is following an "analysis-before-deployment" design paradigm for drafting TLS 1.3 [PvdM16]. We hope to find in the future more examples of such proactive standardisation processes. We also stress the importance of automation we shall obtain from those verification techniques. While the number of corner cases to examine is often too large to be examined by humans, we hope for automated methods to conduct exhaustive analyses. Automated methods are also motivated by the large number of security protocols and their variations to be verified.

Two main distinct approaches have emerged, starting with the early 1980's attempt of [DY83], to ground security analysis of security protocols on formal methods. These two approaches are known as the *computational approach* and the *symbolic approach*. They essentially capture the same threat model on the network (*i.e.* they both consider that the attacker entirely controls the network) but differ in the threat model on the cryptographic primitives (*i.e.* what an attacker can do with them).

### 1.3.1 Computational Approach

In the computational model, the attacker can do anything he wants – including trying to break cryptographic primitives – as long as the overall amount of time and computational power he needs is bounded (the bound may polynomially depend on security parameters such as key length). More formally, messages are modelled as bit strings and agents and the attacker as probabilistic polynomial time Turing machines. Then, security goals are defined using games played by the attacker who has to be able to distinguish the protocol from an idealised version of it (with a non negligible probability). Finally, proofs are done via reductions (or hops) between successive games starting with the one we are interested in and ending with games expressing computational assumptions on cryptographic primitives. It is generally acknowledged that security proofs in this model offer powerful security guarantees. Indeed, such a model captures logical attacks as well as cryptographic design flaws. The latter rely on one or several flaws in cryptographic primitives. Note that this model does not capture attacks at lower levels such as implementation and hardware attacks. A serious downside of this approach, however, is that even for small protocols, proofs are usually difficult, tedious, and highly error prone. Moreover, due to the high complexity of such a model, automating such proofs is a very intricate problem that is still in its infancy (*e.g.* CryptoVerif [Bla08]). Finally, computer-aided verification allows for only a low level of automation even though a lot of efforts have been put in developing computer-aided cryptographic verifiers (*e.g.* CertiCrypt [BGZB09], EasyCrypt [BGHB11], FCF [BPKA15, PM15], F$^*$ [BFG$^+$14]).

### 1.3.2 Symbolic Approach

By contrast, the symbolic approach, which is the one targeted by this thesis, makes strong assumptions on cryptographic primitives (*i.e.* perfect primitives) but fully models agents' interactions and algebraic properties of these primitives. Compared to the computational approach where cryptographic primitives are modelled via computational assumptions expressing operations the attacker cannot do, the symbolic approach models cryptographic primitives in a term algebra expressing what the attacker can do, that is exploiting expected algebraic properties of primitives. For instance, symmetric encryption and decryption are modelled as function symbols $\mathsf{enc}$ and $\mathsf{dec}$ along with the equation $\mathsf{dec}(\mathsf{enc}(m,k),k) = m$. This means that, without the corresponding key $k$, it is simply impossible to get back the plaintext $m$ from the ciphertext $\mathsf{enc}(m,k)$. As already explained, this does not mean however that protocols relying on these primitives are necessarily secure as logical attacks may remain (*e.g.* see Subsection 1.2.3 and Section 1.1).

Although more abstract, the symbolic approach benefits from automation (through decision or semi-decision procedures) and can thus target more complex protocols than those analysed using the computational approach. It can also be used sooner in the protocol design process as only little effort needs to be put into verification. We target automatic verification for rather complex protocols and thus choose to follow this symbolic approach.

#### Symbolic Models

The first symbolic model has been described by Dolev and Yao [DY83] and several other models have been proposed since then. Verifying security protocols following the symbolic approach mainly involves three-level models: messages, protocols and security goals.

*Messages.* First, one has to model messages and cryptographic primitives. Whereas messages are bit strings in the real-world (and in the computational approach as well), they are modelled using first-order terms within the symbolic model. Atoms can be for instance nonces, keys, or agent identities. Examples of function symbols are concatenation, asymmetric and symmetric encryptions or digital signatures. Next, the algebraic relations of cryptographic primitives are modelled by reduction rules or equational theories on terms. For instance, as said before, one may model the expected relation for the symmetric decryption by the equation $\mathsf{dec}(\mathsf{enc}(m,k),k) = m$. To conduct automatic analysis, existing approaches often consider fixed sets of primitives or user-defined primitives satisfying some properties.

*Security Protocols.* Second, we need to model protocols themselves. The model thus needs to capture the concurrent nature of security protocols. Therefore, process algebras are very natural solutions. Derivatives of $\pi$-calculus have thus been proposed for this purpose: first the spi-calculus [AG97] and then the applied-pi calculus [AF01]. The key difference between the two concerns the way in which cryptographic primitives are handled. The spi-calculus has a fixed set of built-in primitives (namely, symmetric and asymmetric key encryption), while the applied-pi calculus allows a wide variety of primitives to be defined by means of an equational theory. Process algebras are not the only way to model protocols. We may at least mention the multiset rewriting (MSR) model that has been introduced in [CDL+99], and the strand space

model [FJHG99] that comes with an appealing graphical representation, and some dedicated proof techniques.

*Security Goals.* Third, in order to model security goals, one use either reachability properties or equivalence properties. Both kinds of properties capture different security goals. We describe both of them next.

**Reachability Properties.** Until the early 2000s, most works from the symbolic approach were focusing on reachability properties (also called trace properties), that is, statements that something bad never occurs on any execution trace of a protocol. Secrecy and authentication are typical examples of reachability properties.

*(Weak) secrecy* concerns a message used by the protocol. This is typically a nonce or a secret key that should not become public. Even for this quite simple security property, several definitions have been proposed in the literature. When considering the notion of (weak) secrecy, a public message is a message that can be learnt by the attacker.

*Authentication.* Many security protocols aim at authenticating one agent to another: one agent should become sure of the identity of the other. There are also several variants of authentication. A taxonomy of these has been proposed by Lowe in [Low97].

**Equivalence Properties.** However, many other security goals and notably privacy goals (*e.g.* anonymity, unlinkability) are not of this type but are defined relying on a notion of *indistinguishability*. Intuitively, two protocols are indistinguishable if it is not possible for an attacker to decide whether (s)he is interacting with one or the other. This notion of indistinguishability is also used for defining a stronger notion of secrecy, and we may also rely on this notion of indistinguishability to compare a protocol with an idealised version of it. Now, we list some examples of such equivalence-based security goals. We will see later how indistinguishability is formalised in our setting most appropriately via the notion of *trace equivalence*. Below, we simply list some security goals that can be formalised relying on such a notion.

*Strong secrecy* is stronger than (weak) secrecy, and related to the concept of indistinguishability. Intuitively, strong secrecy means that an adversary cannot see any difference when the value of the secret changes [Aba97, Bla04, AG97].

*Anonymity.* Frequently, communication between two agents reveals their identities and presence to third parties. Indeed, *anonymity* is in general not one of the explicit goals of common authentication protocols. However, as already discussed, we may want protocols that achieve this privacy goal. It has been informally defined in the ISO/IEC 15408-2 standard as follows: *"[Anonymity] ensures that users may use a [protocol] without disclosing their identity."* It is usually formally defined (see [ACRR09, SS96, Cho06]) as the fact that an observer cannot distinguish two scenarios where the same protocol is executed by different users.

*Unlinkability.* Protocols that keep the identity of their users secure may still allow an attacker to identify particular sessions as having involved the same agent. Such linkability attacks may, for instance, make it possible for an attacker to trace the movements of someone carrying an RFID tag without him being able to notice anything (as the attacks on the AKA and the BAC protocol

previously mentioned). Intuitively, protocols are said to provide *unlinkability* (or *untraceability*) according to the ISO/IEC 15408-2 standard, if they *"[...] ensure that a user may make multiple uses of [them] without others being able to link these uses together."* Formally, this is often defined as the fact that an attacker should not be able to distinguish a scenario in which the same agent (*i.e.* the user) is involved in many sessions from one that involved different agents in each session. Following this intuition, slightly different definitions have been proposed (see *e.g.* [ACRR09, ACRR10, BCDH10, VDMR08]). A comparison between these definitions may be found in [BCEDH13]. We will also come back to this notion in this thesis.

*Vote Privacy.* In the context of e-voting, privacy means that the vote of a particular voter is not revealed to anyone. This is one of the fundamental security properties that an e-voting system has to satisfy. Vote privacy is typically defined (see *e.g.* [KR05, DKR08]) by the fact that an observer should not observe when two voters swap their votes, *i.e.* distinguish between a situation where Alice votes *yes* and Bob votes *no* and a situation where these two voters have voted the other way around. Some stronger forms of vote-privacy have also been proposed such as *receipt-freeness* and *coercion-resistance* [DKR06, BHM08].

#### Linking Computational and Symbolic Approaches

Note that a line of work known as *computational soundness* aims at spanning the gap between these two approaches by establishing that, in some cases, security guarantees in the symbolic model imply security guarantees in the computational one. This line has been initiated by Abadi and Rogaway [AR00] and has received much attention since then (see [CKW11] for a survey on computational soundness).

## 1.4 State of the Art: Methods and Tools

In this thesis, we are interested in automatic verification in the symbolic model with a strong focus on equivalence-based properties (defined using *trace equivalence*) such as many privacy goals. This section is dedicated to the state-of-the-art of such symbolic verification methods and tools for equivalence.

**How to circumvent undecidability?** Modelling security protocols using the symbolic approach allows one to benefit from machine support through the use of various existing techniques, ranging from model-checking to resolution and rewriting techniques. As already pointed out, aiming at machine support is really relevant since manual proofs are error-prone, tedious and hard to verify. Moreover, new protocols are developed quite frequently and need to be verified quickly. Nevertheless, verifying a security goal in such a setting (and especially those expressed using equivalence properties) is actually undecidable in the general case [Hüt03, CCD15b]. Two main options are possible then: either recover decidability by only considering a bounded number of sessions (and thus a bounded number of agents as well), or only semi-decide the problem for the unbounded case.

| | Apte [Che14] | Akiss [CCK12] | Spec [TD10] |
|---|---|---|---|
| Equivalence | trace equivalence | over and under-approx. of trace equivalence | open bisim. |
| Primitives | standard | convergent with finite variant | pair & sym. encryption |
| Class of protocols | full | linear role with equality tests | linear role with filtering |
| Symbolic Model | applied-pi calculus | | spi-calculus |
| Termination | proved | proved for sub. convergent | proved |
| **Exploration** | **forward** | | |

**Table 1.1** Main features of existing tools (for bounded number of sessions)

### 1.4.1 Decision for a Bounded Number of Sessions

To design decision procedures, a reasonable assumption is to bound the number of protocol sessions to consider (*i.e.* by forbidding replication) thereby limiting the length of execution traces. Under such an assumption, the first decision procedure towards automatic verification of a notion of equivalence between protocols dates back to [Hüt03], where a fragment of the spi-calculus (no replication, no branching on conditional) is considered. Note that, even under this assumption, infinitely many execution traces remain, since each input may be fed infinitely many different messages (chosen by the attacker). This issue has been tackled in various ways using forms of symbolic execution and the development of dedicated procedures. Obtaining a symbolic semantics to avoid potentially infinite branching of execution trees is often a first step towards automation of equivalence. Depending on the expressivity of the calculus and the way its semantics is given, this task can be quite cumbersome (*e.g.* for applied-pi calculus [DKR10], spi-calculus [DSV03, Bor09], psi calculus [BGRV15]) and sometimes only leads to incomplete procedures.

A table summarising the main features of existing tools dedicated to the bounded case is given in Table 1.1.

**Constraint solving approaches.** Some pioneering works paved the way for existing tools but have not been implemented due to their practicality (that was not their focus). For instance, Baudet designed [Bau05] a decision procedure for verifying indistinguishability from a passive attacker (*i.e.* who can only eavesdrop output messages) point of view. The latter boils down to verifying the indistinguishability between two sequences of open messages (*i.e.* messages with some unknown parts constrained with some deducibility and equality constraints). The main novelty of this work was to design a constraint solving procedure that is not only able to solve satisfiability problems (sufficient for reachability properties) but also to establish equivalences

(*i.e.* two systems have the same sets of solutions), which are needed when one wants to verify equivalence-based security goals. This is done for a user-defined equational theory given in the form of a sub-term convergent rewriting system (*i.e.* convergent and such that the right-hand side of each rewriting rule is actually a syntactic sub-term of the left-hand side). As shown in [CCD13a], this yields a procedure deciding trace equivalence of a restricted class of protocols for many primitives. A shorter proof of the result by Baudet is given in [CR12]. It is shown that if two protocols are not equivalent, then there must exist a small witness of non-equivalence, and a decision procedure can be derived by checking every possible small witness. However, as said above, those decision procedures have not been implemented.

A decision procedure for a stronger notion of trace equivalence (namely *open bisimulation*) has been proposed in [TD10] and implemented in the tool Spec[9]. The procedure deals with a fixed set of cryptographic primitives, namely symmetric encryption and pairs, and protocols modelled as linear roles with filtering (*i.e.* each agent has to be modelled by a sequence of output and inputs with possibly expected patterns for the received messages). The procedure is sound and complete w.r.t. open bisimulation (a notion that is strictly stronger than trace equivalence [Tiu07]) and its termination is proved. The attacker's deductive ability is modelled as logical rules in sequent calculus, and procedures deciding message deduction and message indistinguishability are defined as proof-search strategies. Finally, the proposed procedure iteratively builds an open bisimulation from the two initial protocols by symbolically executing them and checking that possible instantiations are coherent on both sides.

For a fixed but richer set of cryptographic primitives (*i.e.* symmetric/asymmetric encryptions, signature, pair, and hash functions), a different procedure, presented in [CCD11] (improved version of [CCD10]), allows to decide equivalence of two *sets* of constraint systems that may also feature disequality tests. Dealing with disequality tests and sets of constraint systems is needed in the presence of protocols that can branch on conditionals (*i.e.* with proper else branches) since different symbolic executions may then be associated to a single symbolic trace. This procedure explores all possible symbolic traces and computes all possible resulting symbolic constraint systems on both sides. This forward symbolic exploration of two protocols is finite since all symbolic traces have a bounded length and the exploration is finitely branching since inputs are abstracted away by variables and constraints. The procedure then checks the symbolic equivalence of all the resulting pairs of sets of constraint systems. Recently (subsequent to the start of this thesis), this procedure has been further extended to deal with some forms of *side-channel* attacks regarding the length of messages [CCP13], and the computation time [CC15]. This procedure and its extensions have been implemented in the tool Apte [Che14].

**Resolution-based approaches.** The procedure described in [CCK12] deals with rich user-defined term algebras provided that they can be defined using a convergent rewriting system enjoying the *finite variant property* [CLD05]. This property basically requires that it is possible to finitely pre-compute possible normal forms of terms with variables. This in particular includes

[9] `http://www.ntu.edu.sg/home/atiu/spec-prover/`

all sub-term convergent equational theories. In the setting of [CCK12], protocols are modelled as linear roles with equality tests (*i.e.* each agent has to be modelled by a sequence of inputs, outputs and equality tests). Further, the authors of [CCK12] use first-order Horn clauses to model all possible instantiations of symbolic traces, and they rely on a saturation procedure to put all clauses into *solved forms*. Finally, this finite description of all possible concrete executions is used to decide equivalence between the two protocols under study. This procedure is actually able to check an over-approximation and an under approximation of trace equivalence, and it has been shown that the former over-approximation actually coincides with trace equivalence for a large class of protocols. This procedure has been implemented in the tool Akiss[10]. After the beginning of this thesis, termination of the procedure has been established for sub-term convergent theories [CCCK16].

### 1.4.2 Semi-Decision for an Unbounded Number of Sessions

The decidability results mentioned so far analyse equivalence for a bounded number of sessions and agents only, that is assuming that protocols are executed a limited number of times. This is of course a strong limitation. Even if no flaw is found when a protocol is executed $n$ times, there is absolutely no guarantee that the protocol remains secure when it is executed $n+1$ times. Therefore, despite the difficulty of the problem in the general case, several solutions have been proposed in order to deal with the unbounded case. To circumvent the undecidability of the problem in that case, many works aim at developing procedures (not necessarily completely automatic) that are sound w.r.t. trace equivalence but not complete. One source of incompleteness is non-termination, another one is the over-approximation of trace equivalence by a stronger form of equivalence. Indeed, in order to make those procedures deal with a form of equivalence, the main idea is to merge the two protocols under study into a so-called *bi-process*, and to consider a strong form of equivalence, namely *diff-equivalence*. This method has first been presented in [BAF05] and implemented in the ProVerif tool. After the start of this thesis, this technique has been integrated into the verification tools Tamarin [BDS15] and Maude−NPA [SEMM14] that have been extended to deal with equivalence properties. A table summarising the main features of existing tools dealing with an unbounded number of sessions is given in Table 1.2.

The method presented in [BAF08] and implemented in the most established tool ProVerif[11] represents bi-processes that are given in input using Horn clauses (performing some well-chosen approximations, and thus losing completeness). Then, a dedicated resolution algorithm tries to derive a logical contradiction from those Horn clauses. Cryptographic primitives (that can be user-defined) are decomposed into: reduction rules and a union of linear equational theories (*i.e.* each equation has the same variables on both sides) and convergent theories (*i.e.* terminating and confluent) that must have the finite variant property. This formalism is flexible enough to model for instance different flavours of encryptions (*e.g.* symmetric, asymmetric, randomised), signature, and blind signature, but excludes exclusive or, and more generally associative and

[10] http://akiss.gforge.inria.fr
[11] http://proverif.inria.fr

| | ProVerif [BAF08] | Maude−NPA [SEMM14] | Tamarin [MSCB13] |
|---|---|---|---|
| **Equivalence** | **diff-equivalence** | | |
| Primitives | linear + convergent with finite variant | convergent modulo AC with finite variant (inc. XOR, A.G.) | convergent with finite variant + DH |
| Class of protocols | full | linear role with filtering | full + state |
| Input syntax | applied-pi calculus | strand spaces | multiset rewriting |
| Termination | may diverge | | |
| Exploration | resolution | backward | |

**Table 1.2** Main features of existing tools (for an unbounded number of sessions)

commutative operators. The resulting tool is very efficient, and terminates on many examples. It has thus established itself as the leading tool.

After the start of this thesis, the approach behind the Tamarin verification tool [MSCB13] has been extended to deal with diff-equivalence. In this approach, protocols are modelled as multiset rewriting (MSR) systems. This allows one to model a rich class of protocols that may branch on conditionals (*i.e.* featuring proper else branches) and allow the storage of some data from one session to another (*i.e.* stateful protocols). The framework supports a rich term algebra including user-defined convergent equational theories that have the finite variant property (thanks to recent efforts [DDKS17]), and Diffie-Hellman exponentiation. The proposed algorithm exploits the finite variant property [CLD05] to get rid of some equations, and it builds on ideas from strands spaces and proof normal forms. It basically performs a backward search from attack states. Tamarin provides two ways of constructing proofs: an efficient, fully automated mode that uses heuristics to guide proof search, and an interactive mode. The interactive mode enables the user to explore the proof states using a graphical interface. The Tamarin tool has been used to analyse different security properties on many protocols. However, regarding equivalence, the tool is less mature and has only been used on a few examples; the main one being a stateful TPM protocol (namely the TPM envelope protocol) on which a strong secrecy property has been established.

The Maude−NPA tool has also been extended [SEMM14] (after the beginning of this thesis) to deal with bi-processes (that they call *synchronous product*) and diff-equivalence. Their semi-decision procedure is able to deal with a very large class of term algebras as long as they are convergent modulo AC (associative and commmutative) and have the *finite variant property* (modulo AC) as defined in [ESM10], such as Abelian groups, exclusive or, and exponentiation. However, it can only be applied to linear role scripts with filtering over inputs (and therefore does not handle protocols with else branches). Regarding equivalence, only a few case studies have been carried out. Moreover, the approach of [SEMM14] suffers from termination problems, especially when considering primitives such as exclusive or.

### 1.4.3 Other Results

Besides those two main approaches to the verification problem, a different recent line of work aims at proving the decidability of trace equivalence verification for the unbounded case but for constrained classes of protocols and cryptographic primitives. For examples, such results were shown (after the beginning of this thesis) for ping-pong protocols [CCD13b, CCD15b] (*i.e.* having at most one variable per protocol rule) that are determinate (*i.e.* agents in parallel are using different channel names), and for type-compliant (generalising the idea of tagged protocols [BP03]) and acyclic (roughly protocols without loops in their well-typed executions) protocols [CCD15a] for a fixed set of primitives (namely pairs, and symmetric encryption only). However, the constraints imposed by the restricted classes of primitives and protocols make those results still too impractical and no usable tool resulted from those decidability results.

## 1.5 Problems

The two approaches, *i.e.* decision for the bounded case or semi-decision for the unbounded case, suffer from two different problems that significantly limit their practical impact: *state space explosion problem* for the former and *lack of precision* when it comes to verifying *privacy goals* for the latter. We describe those two limitations respectively in the two next sections. We aim at addressing those two problems in this thesis.

### 1.5.1 Main Limitation for the Bounded Case: State Space Explosion

As shown in Subsection 1.4.1, all methods and tools (*i.e.* Apte, Spec and Akiss), which decide trace equivalence (or an approximation of it) for a bounded number of sessions, symbolically explore all possible execution traces. However, due to the highly concurrent nature of security protocols, there are too many interleavings of concurrent actions to consider. This well-known problem in concurrency and model-checking is called *state space explosion problem*. This problem seriously limits the practical impact of existing tools in that category.

Indeed, in practice, this translates to really bad scaling. For instance, at the time of the beginning of this thesis, anonymity in the Private Authentication protocol could be established in less than 0.1 second for a simple scenario with only one session of each role whereas it takes respectively an hour and more than 2 days as soon as we want to consider respectively 2 and 3 sessions of each role.

In standard model-checking approaches for concurrent systems, this is handled using *partial order reduction* (POR) techniques [Pel98]. For instance, the order of execution of two independent (parallel) actions is typically irrelevant for checking reachability. The theory of POR is well developed in the context of reactive systems verification (*e.g.* [Pel98, BK08, GvLH+96]). However, as pointed out by E. Clarke *et al.* in [CJM03], POR techniques from traditional model-checking cannot be directly applied in the context of security protocol verification. Indeed, the application to security requires one to keep track of the knowledge of the attacker, and to refer to this knowledge in a meaningful way (in particular to know which messages can be forged at

some point to feed some input). Furthermore, security protocol analysis does not rely on the internal reduction of a protocol, but has to consider arbitrary execution contexts (representing interactions with arbitrary, active attackers). Thus, any input may depend on any output, since the attacker has the liberty of constructing arbitrary messages from past outputs. This results in a dependency relation which is *a priori* very large, rendering traditional POR arguments sub-optimal, and calling for domain-specific techniques. Things become even more complex when working with a symbolic semantics: the states obtained from the interleaving of parallel actions will differ, but the sets of concrete states that they represent will have a significant overlap. Finally, POR techniques are usually devised for reachability properties verification while we are rather interested in trace equivalence verification. Hence, extra precautions have to be taken before discarding a particular interleaving: we have to ensure that this is done in both sides of the equivalence in a similar fashion.

> Hence, there is a need for new POR techniques for trace equivalence that are compatible with the security setting and which can be nicely integrated in existing methods and tools.

### 1.5.2 Main Limitations for the Unbounded Case: Lack of Precision

We now turn to methods & tools that semi-decide trace equivalence (*i.e.* ProVerif, Tamarin and Maude−NPA). As said before in Subsection 1.4.2, existing tools in that category all rely on a stronger notion of equivalence instead (*i.e.* diff-equivalence) that greatly over-approximates trace equivalence. Basically, diff-equivalence requires that the two protocols initially share the same structure and that they can be executed exactly in the same way, notably for internal rules (*e.g.* conditional evaluations), whereas the attacker cannot observe such details. This problem has been partially addressed in [CB13] for ProVerif by pushing away the evaluation of some conditionals into terms. Nevertheless, the problem remains in general (*e.g.* for generic conditionals, indistinguishable actions in parallel[12]).

This gap between trace equivalence and diff-equivalence may not be a problem when verifying some properties such as strong-secrecy. However, it is definitely a problem for the verification of some privacy goals such as *unlinkability* for instance. Thus, considering the well-established formal definition of strong unlinkability of [ACRR10], we end up verifying an equivalence between two protocols that do not share the same structure. Worse, even after merging the two protocols to be proved equivalent using some encodings, the two protocols can mimic each other but cannot do so while keeping exactly the same structure. Therefore, verifying this notion using the unrealistically strong notion of diff-equivalence systematically leads to false attacks. The deep reason is the lack of precision of the over-approximation made by diff-equivalence over trace equivalence. In practice, it means that tools verifying diff-equivalence instead of trace equivalence (*i.e.* ProVerif, Tamarin, Maude−NPA) still cannot be used off-the-shelf to establish

[12]The very recent extension [BS16] neither solves the problem as it only allows to relax those constraints in specific situations.

unlinkability for many security protocols such as[13]: the BAC protocol, PACE protocol that should replace BAC in e-passports, RFID protocols such as Feldhofer, Hash-Lock, LAK. Similar problems arise also in the e-voting context where vote-privacy cannot be established using diff-equivalence for many protocols despite recent efforts trying to address this particular precision issue [BS16].

> Targeting the automation of diff-equivalence is often not enough to obtain verification of privacy. We thus need new methods for verifying privacy in the unbounded case that do not suffer from this lack of precision.

## 1.6 Contributions

We identified in the previous section two important problems, one for each approach of the state-of-the-art, that negatively impact a lot the existing methods and tools. We describe how our contributions respectively address those two problems in subsections 1.6.1 and 1.6.2. Finally, Subsection 1.6.3 focuses on contributions taking the form of software implementations and security protocol modellings and analyses.

### 1.6.1 POR Techniques for the Bounded Case

In this thesis, we develop new POR techniques for trace equivalence checking of security protocols. Besides the already mentioned extra complexity brought by the security setting and trace equivalence, another important challenge is to do it in a way that is compatible with symbolic execution: we should provide a reduction that is effective when messages remain unknown, but leverages information about messages when it is inferred by the constraint solver.

We devise our POR techniques by refining interleaving semantics in two steps, gradually eliminating redundant traces. The first refinement, called *compression*, uses a notion of polarity over processes to impose a simple strategy for the exploration of traces. It does not rely on data analysis at all but only on syntactical information about current available processes. This first technique can thus easily be used as a replacement for the usual semantics in verification algorithms. The second one, called *reduction*, takes data into account and achieves optimality in eliminating redundant traces. In theory, the reduction step can be implemented in an approximated fashion, through an extension of constraint resolution procedures by leveraging information about messages as soon as available. For each refined semantics, we show that it captures essentially the same reachability properties and more importantly that it captures the same trace equivalence notion.

We also formally explain how to integrate such techniques into symbolic semantics with constraint solving. We lift those refined semantics to refined, symbolic semantics and prove that reachability and trace equivalence are preserved. Next, we put those techniques into practice in the tool Apte: from the theoretical aspects of this integration to the implementation in the

[13] We will describe more precisely those protocols and why unlinkability cannot be verified directly using diff-equivalence in Part C.

distributed code of Apte. Hence, we axiomatize the building blocks of the tool Apte in order to prove the correctness of the tool Apte using our refined semantics (instead of the regular, unoptimised semantics). We also present extensive benchmarks of Apte showing that our theoretical results do translate to dramatic speedups.

We claim that our POR techniques – at least compression – and the significant optimisations they allow are generic enough to be applicable to other verification methods as long as they perform forward symbolic executions. In addition to the integration in Apte we will extensively discuss, we also have successfully done so in Spec [Hirb]. Furthermore, parts of POR techniques have been independently integrated and implemented in the distributed version of Akiss[14].

### 1.6.2 Verifying Privacy via Sufficient Conditions for the Unbounded Case

Solutions to the previously mentioned precision issue often consists in improving the tools and the notion of diff-equivalence they verify to get closer to trace equivalence [CB13, BS16]. We believe that targeting trace equivalence verification for the unbounded case is looking at a too general problem. We thus approach the problem differently: by focusing on a class of protocols and some privacy goals we would like to verify on them, we are able to precisely characterise when those goals are met via *sufficient conditions*. While we believe our overall methodology is generic enough to be used in various contexts, in this thesis, we fully develop our methodology for the case of unlinkability and anonymity on 2-party protocols.

We proceed as follows. We carefully analyse reasons for having attacks on unlinkability or anonymity for 2-party protocols. In light of that analysis, we define two conditions preventing two generic kinds of attacks. In a nutshell, our first condition avoids attacks based on control-flow leaks that may be observed through data or nature of actions while our second condition avoids attacks based on data leaks taking the form of relations between outputs. Our main theorem states that these two conditions are actually always sufficient to ensure both unlinkability and anonymity. This is relevant in practice since our two conditions are fundamentally simpler and are in the scope of existing verification tools like ProVerif. They can be precisely verified via diff-equivalence without the previous systematic precision issue. We thus obtain a sound method for automatically checking unlinkability and anonymity. We have implemented a tool called UKano that automatically generates models to verify our two conditions. It suffices to feed this tool with the specification of a security protocol written in a ProVerif model and it will build a model for each condition and call ProVerif on them in order to verify the conditions.

The resulting method and tool is obviously not complete but we show that it is precise enough to conclude on many case studies. We notably establish the first proof of unlinkability for the ABCDH protocol [AH13] and the BAC protocol followed by the Passive Authentication (PA) and Active Authentication (AA) protocols used in e-passports. We also report on an attack that we found on the PACE protocol (that should replace BAC in e-passports), and another one that we found on the LAK protocol whereas it was claimed untraceable. Furthermore, our method also enables us to conduct the first automatic analysis of unlinkability for many security protocols

[14] See `https://github.com/akiss/akiss`.

(*e.g.* PACE, Feldhofer, Hash-Lock, LAK). Note that it happens that our conditions are rather tight: for all our case studies, we provide an attack every time one of them is not satisfied.

Finally, our method is generic enough to verify different flavors of unlinkability and anonymity. We build on previous definitions (that we compare briefly) to propose different variations of unlinkability and anonymity. We show that those variations capture different threat models or practical scenarios and provide examples that distinguish all of them.

Finally, we believe that the underlying methodology is of interest in itself as it may be used to devise new methods for verifying other types of complex privacy goals for other classes of security protocols addressing the precision issue more broadly (*e.g.* as done for the e-voting setting in [CH17]).

### 1.6.3 Developed Software and Models

We now focus on our implementation and modelling efforts.

As already discussed in Subsection 1.6.1, our POR techniques have been implemented in the distributed version of Apte. We stress the fact that Apte is a rather big software of more than 14k OCaml LoC based on a highly complex algorithm: description and proofs of the underlying algorithm are available in a long and technical appendix (more than 100 pages) from [Che12]. The implementation of our POR techniques involved about 3k LoC modifications. We have also carried out extensive benchmarks to test our optimisations on several security protocols. More details about this implementation and the benchmarks can be found at `http://www.lsv.ens-cachan.fr/~hirschi/apte_por`. We have also implemented a preliminary version of our POR techniques in the tool Spec and carried out benchmarks to measure speedups. More details about this implementation are given at `http://www.lsv.ens-cachan.fr/~hirschi/spec`.

We have built the tool UKano briefly presented in Subsection 1.6.2 by modifying the tool ProVerif. We only re-used the lexer, parser and AST of ProVerif and build upon those a generator and translator of ProVerif models. Our tool is able to verify that a given protocol lies in our class of 2-party protocols. It then leverages several heuristics to automatically generate ProVerif models to verify our sufficient conditions. This effort represents more than 2k OCaml LoC. We applied our tool on many real-world case studies. Some of those case studies were never symbolically analysed before. Therefore, we also provide first symbolic models for several security protocols (*e.g.* ABCDH, PACE, Hash-Lock). More details are given on the the official page of UKano at `http://projects.lsv.ens-cachan.fr/ukano/`.

## 1.7 Organisation of the Thesis

The present thesis is divided into three main parts.

**Part A.** The first part is dedicated to the symbolic model and the common security notions that will be used in this thesis. We define a generic symbolic model in Chapter 2 on which all subsequent parts build upon. Then, in Chapter 3, we define trace equivalence before giving a

comparisons with other behavioural equivalences. We also explain how to model some privacy goals using this notion. Finally, we show in Chapter 4 how to derive variations of models (that will be used in subsequent parts) from our generic symbolic model and prove that all those variations capture the same security notions (and threat models) and most notably the same notion of trace equivalence.

**Part B.** Our POR techniques addressing the state space explosion problem are described in the second part. In Chapter 5, we define our POR techniques in the most generic setting. We then show how to lift those techniques to symbolic semantics and prove the correctness of the tool Apte optimised with our POR techniques in Chapter 6. We also show and discuss benchmarks. Finally, we give comprehensive comparisons with related works in Chapter 7.

**Part C.** The final part of this thesis is dedicated to our new method to verify privacy in the unbounded case. In Chapter 8, we define the class of security protocols we will deal with. We also define unlinkability and anonymity for the unbounded case and propose variations of the most-established definitions. We then define in Chapter 9 our two conditions and prove our main theorem stating that they always imply unlinkability and anonymity. Finally, Chapter 10 shows the practical relevance of our method: we explain how to precisely verify the two sufficient conditions and how our tool UKano automates this task. We also discuss an extensive list of case studies.

Finally, we close the thesis with conclusive remarks and avenues for future work in Chapter 12.

### Publications and Other Contributions

Most of the results presented in this thesis have been previously published sometimes in a weaker form and always for specific semantics (thus lacking a uniform symbolic model). Some material from this introduction comes from [DH16]. Part B describes contributions first published at several places [BDH14, BDH15, BDH]. Part C describes enhanced and extended results based on [HBD16] (and a journal version in progress).

However, in the interest of uniformity, several other results have not been included in this thesis. Therefore, for the sake of completeness, we list all the publications of the author on the next page.

# Publications by the Author

David Baelde, Stéphanie Delaune, and Lucca Hirschi. A reduced semantics for deciding trace equivalence. *Logical Methods in Computer Science*, 2017.

Piers O'Hanlon, Ravishankar Borgaonkar, and Lucca Hirschi. Mobile subscriber wifi privacy. In *Proceedings of Mobile Security Technologies (MoST'17), held as part of the IEEE Computer Society Security and Privacy Workshops (SPW'17)*, 2017. To appear.

Stéphanie Delaune and Lucca Hirschi. A survey of symbolic methods for establishing equivalence-based properties in cryptographic protocols. *Journal of Logical and Algebraic Methods in Programming*, 2016.

Amina Doumane, David Baelde, Lucca Hirschi, and Alexis Saurin. Towards completeness via proof search in the linear time $\mu$-calculus: The case of büchi inclusions. In *31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'16)*, pages 377–386. ACM, 2016.

Lucca Hirschi, David Baelde, and Stéphanie Delaune. A method for verifying privacy-type properties: the unbounded case. In *37th Symposium on Security and Privacy (Oakland'16)*, pages 564–581. IEEE, 2016.

David Baelde, Stéphanie Delaune, and Lucca Hirschi. Partial order reduction for security protocols. In *26th International Conference on Concurrency Theory (CONCUR'15)*, page 497, 2015.

David Baelde, Stéphanie Delaune, and Lucca Hirschi. A reduced semantics for deciding trace equivalence using constraint systems. In *3th International Conference on Principles of Security and Trust (POST'14)*, pages 1–21. Springer, 2014.

# Part A

# Model

## Table of Contents of the Part

# Introduction

This part is dedicated to the model we use to represent security protocols and security goals. As already discussed, we target a symbolic model abstracting cryptographic primitives but considering a strong attacker who is controlling all the network: he may eavesdrop messages, inject messages and use cryptographic primitives. As mentioned in the general introduction, several symbolic models have been proposed. The first one has been described by Dolev and Yao [DY83] and several other models in the same flavor have been proposed since then. Unfortunately, there is currently no unified, consensual such model. The reason for having several popular symbolic models probably comes from the fact that they have to achieve two antagonistic goals. On the one hand, models have to be as fine grained and expressive as possible to capture a large range of applications. One the other hand, models have to remain relatively simple in order to allow simple proofs and the design of verification procedures.

In this thesis, we shall work with a variant of the widespread dialect of Blanchet, Abadi & Fournet [BAF08] on which the tool ProVerif is based. This model is inspired from cryptographic calculi and is pretty close to the applied $\pi$-calculus [AF01]. More importantly, the framework we define in this part is generic enough to be instantiated in different ways. We will notably be able to instantiate it in the models on which the tools we eventually leverage in this thesis are based (*i.e.* ProVerif and Apte). In this part, we also define trace equivalence for our symbolic model and explain how some privacy goals can be defined relying on this notion.

We stress the fact that some of our subsequent developments, that will be built upon our uniform and generic symbolic model, have different requirements w.r.t. the way the semantics of security protocols is defined. This is often the case and, as already mentioned, partly explains the diversity of symbolic models that have been proposed in the literature. However, in this thesis, we put effort into defining a generic semantics that can be parametrised in different ways in order to meet different requirements and expose different levels of details. More importantly, we prove that all variations of our symbolic model one can thus obtain capture the same security notions. More precisely, we prove that the induced reachability properties and more importantly trace equivalence coincide for all of them.

**Outline.** First, we formally define the process algebra we shall use to model security protocols in Chapter 2. We notably define a generic semantics describing the processes' behaviours. We also describe different ways to model cryptographic primitives and their algebraic relations. In Chapter 3, we formally define trace equivalence. We also compare this notion with other behavioural equivalences. We conclude this chapter with illustrations of privacy goals modelled using trace equivalence. Finally, in Chapter 4, we show how one can derive variants of the generic semantics defined before, all capturing the same security notions. Those variants have different technical features and we shall use one or the other depending on our needs in subsequent developments.

*Chapter 2*

# Modelling Security Protocols

In this chapter, we define the generic model of security protocols we shall use in this thesis. This model is a variant of the dialect of Blanchet, Abadi & Fournet [BAF08] based on the applied $\pi$-calculus [AF01].

Messages exchanged by agents taking part in the protocol are represented using a term algebra. In such a term algebra, the algebraic properties of the data structures and the cryptographic primitives are modelled through an equational theory and a computation relation (generalizing reduction rules). We define a generic notion of such term algebras and give some examples in Section 2.1.

Agents of a protocol are modelled as processes in a process algebra describing all their possible behaviours. The semantics of that process algebra specifies how processes can interact with each other by sending and receiving messages through an insecure network totally controlled by an attacker. We define the syntax and the semantics of that process algebra in Section 2.2.

Further, we show in Section 2.3 different ways to realise instances of the generic notion of term algebra we defined notably corresponding to the tools we eventually leverage in this thesis: ProVerif and Apte.

## 2.1 Term Algebra

We now present term algebras, which will be used to model messages built and manipulated using various cryptographic primitives and data structures. We consider an infinite set $\mathcal{N}$ of *names* (denoted $k, n$) which are used to represent keys or nonces and two infinite and disjoint sets of *variables* $\mathcal{X}$ and $\mathcal{W}$. Members of $\mathcal{X}$ are denoted $x$, $y$, $z$ and are typically used to refer to unknown parts of messages expected by participants, while variables in $\mathcal{W}$, denoted $w$ and called *handles*, are used to store messages learned by the attacker. We assume a *signature* $\Sigma$, *i.e.* a set of function symbols together with their arity. The elements of $\Sigma$ are split into *constructor* and *destructor* symbols, *i.e.* $\Sigma = \Sigma_c \sqcup \Sigma_d$. On the one hand, *destructors* are meant to model computations that may fail; and such a failure can be observed. Signature and MAC verification are examples of this kind. On the other hand, *constructors* can be used to model total functions

possibly satisfying some algebraic properties. For instance, exclusive or and Diffie-Hellmann exponentiation would be modelled using constructors.

Given a signature $\Sigma$, and a set of atoms $A$, we denote by $\mathcal{T}(\Sigma, A)$ the set of terms built from elements of $A$ by applying function symbols in $\Sigma$. We shall call *terms* the elements of $\mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X})$ and note them $t, s$. Terms in $\mathcal{T}(\Sigma_c, \mathcal{N} \cup \mathcal{X})$ will be called *constructor terms*. We denote $\mathrm{vars}(t)$ the set of variables that occur in a term $t$. A *message* is a constructor term $u$ that is *ground*, *i.e.* such that $\mathrm{vars}(u) = \emptyset$. We let $u, v$ range over messages. Messages are the only terms that can be exchanged through the network while terms shall be *executed* yielding either a message or a failure. Executions of terms are described by the *computation relation* we define in Section 2.1.2.

We denote by $\overline{x}$, $\overline{n}$, $\overline{t}$ a (possibly empty) sequence of variables, names, and terms respectively. The application of a substitution $\sigma$ to a term $u$ is written $u\sigma$, and we denote $\mathrm{dom}(\sigma)$ the *domain* of $\sigma$. The *positions* of a term and the notion of *contexts* (*i.e.* terms with holes) are defined as usual. We write $\mathsf{subTerms}(t)$ to denote the set of syntactic sub-terms of a term $t$.

**Example 1.** *Consider the signature:*

$$\Sigma = \{\mathsf{senc},\ \mathsf{sdec},\ \langle\,\rangle,\ \pi_1,\ \pi_2,\ \oplus,\ 0,\ \mathsf{eq},\ \mathsf{neq},\ \mathsf{ok}\}.$$

*The symbols* $\mathsf{senc}(\bullet, \bullet)$ *and* $\mathsf{sdec}(\bullet, \bullet)$ *of arity 2 represent symmetric encryption and decryption. Pairing is modelled using* $\langle \bullet, \bullet \rangle$ *of arity 2, whereas projection functions are denoted* $\pi_1(\bullet)$ *and* $\pi_2(\bullet)$*, both of arity 1. The function symbol* $\bullet \oplus \bullet$ *of arity 2 and the constant* $0$ *are used to model the exclusive or operator. Finally, we consider the symbol* $\bullet\, \mathsf{eq}\, \bullet$ *of arity 2 to model equality test, as well as the constant symbol* $\mathsf{ok}$*. This signature is split into two parts:* $\Sigma_c = \{\mathsf{senc}, \langle\,\rangle, \oplus, 0, \mathsf{ok}\}$*, and* $\Sigma_d = \{\mathsf{sdec}, \pi_1, \pi_2, \mathsf{eq}, \mathsf{neq}\}$.

### 2.1.1 Semantics of Messages: Equational Theory

Messages are given a semantics through an equational theory. This has proved very useful for modelling algebraic properties of cryptographic primitives (see *e.g.* [CDL06] for a survey). Formally, we consider a relation $\mathsf{E} : \mathcal{T}(\Sigma_c, \mathcal{X}) \times \mathcal{T}(\Sigma_c, \mathcal{X})$ describing a set of equations modelling each an algebraic relation satisfied by constructor symbols. This induces a congruence relation generated from those equations as defined next.

**Definition 1.** *Let* $\mathsf{E} : \mathcal{T}(\Sigma_c, \mathcal{X}) \times \mathcal{T}(\Sigma_c, \mathcal{X})$ *be a relation over constructor terms. The equational theory* $=_{\mathsf{E}}$ *is the smallest equivalence relation over* $\mathcal{T}(\Sigma_c, \mathcal{N} \cup \mathcal{X})$ *containing* $\mathsf{E}$*, stable by substitution from* $\mathcal{X}$ *to* $\mathcal{T}(\Sigma_c, \mathcal{N} \cup \mathcal{X})$ *and stable by application of function symbols in* $\Sigma_c$.

Note that, by definition, $=_{\mathsf{E}}$ is closed under bijective renaming. The equational theory is meant to be transparent from the point of view of participants and the attacker; *i.e.* all representatives of a given equivalent class of $=_{\mathsf{E}}$ are indistinguishable for them since they model the same piece of data. The semantics of protocols itself will be stable by $=_{\mathsf{E}}$.

Note that we assume that $\mathsf{E}$ is chosen such that $=_{\mathsf{E}}$ is not degenerate, *i.e.* there exist two messages $u, v$ such that $u \neq_{\mathsf{E}} v$.

**Example 2.** *To reflect the algebraic properties of the exclusive or operator, we may consider the equational theory generated by the following equations:*

$$\begin{array}{rclcrcl} x \oplus 0 & = & x & & (x \oplus y) \oplus z & = & x \oplus (y \oplus z) \\ x \oplus x & = & 0 & & (x \oplus y) & = & (y \oplus x) \end{array}$$

*In such a case, we have that* $\mathsf{senc}(a \oplus (b \oplus a), k) =_{\mathsf{E}} \mathsf{senc}(b, k)$. *Remind that all representatives of the same* $=_{\mathsf{E}}$*-class essentially models the same piece of data. For instance, nobody (no participant nor the attacker) can distinguish* $k \oplus n$ *from* $n \oplus k$ *where* $k, n \in \mathcal{N}$.

**Example 3.** *When modelling a symmetric encryption scheme whose decryption algorithm is a bijective, total function (*e.g. *block cipher like AES), we shall use constructors for encryption and decryption and model decryption property through equations. For instance, over the signature* $\Sigma_c = \{\mathsf{senc}', \mathsf{sdec}', \oplus\}$, *the equational theory induced by* $\mathsf{sdec}'(\mathsf{senc}'(x, y), y) = x$, $\mathsf{senc}'(\mathsf{sdec}'(x, y), y) = x$ *and equations in Example 2 gives the expected semantics for symmetric decryption. In such a case, we have that* $\mathsf{sdec}'(\mathsf{senc}'(a, k \oplus a \oplus a), k) =_{\mathsf{E}} a$. *Moreover, even though the keys do not match in* $\mathsf{sdec}'(\mathsf{senc}'(a, k), k')$, *this term is considered as a message that might be sent or received. Moreover, the attacker or other participants have no means to know if the keys match*[1].

### 2.1.2 Semantics of Terms: Computation Relation

We now give a semantics to terms through a *computation relation* which describes how terms can be *executed* yielding either a message or a failure.

Contrary to equational theories, computation relations allow to model computations that may fail. For example, when modelling symmetric encryption as in Example 3, decryption never fails even if keys do not match: it always returns a message. With a computation relation, it is possible to express a failure in such a case. For instance, contrary to the Example 3, the destructor symbol $\mathsf{sdec}$ introduced in Example 1 shall be given a semantics corresponding to a partial function that may fail when the given keys do not match: $\mathsf{sdec}(\mathsf{senc}(u, k), k')$ would fail while $\mathsf{sdec}(\mathsf{senc}(\mathsf{ok}, k), k)$ would not fail and would compute the message $\mathsf{ok}$.

We now give a generic definition of computation relations. We prefer axiomatizing computation relations with all requirements they must satisfy rather than giving operational ways to obtain them (we list some of them later on in Section 2.3). There are many requirements for such computation relations mainly because its interactions with the equational theory have to fulfil expected properties. For example, when two messages $u, v$ are equal modulo $=_{\mathsf{E}}$, replacing $u$ by $v$ in a term $t$ should not impact the computation of $t$ since $u$ and $v$ represent the same piece of data. We describe in Section 2.3 several ways to operationally generate such computation relations (*e.g.* from reduction rules or equations) satisfying all requirements.

**Definition 2.** *A* computation relation *is a relation over* $\mathcal{T}(\Sigma, \mathcal{N}) \times \mathcal{T}(\Sigma_c, \mathcal{N})$, *denoted* $\Downarrow$, *such that:*

[1] Except if they already know part of the plaintext. Indeed, in such a case, they can try to decrypt and check whether the result matches the part of the expected plaintext they know.

- $n \Downarrow n$ *for any* $n \in \mathcal{N}$*;*
- $\mathsf{f}(t_1, \ldots, t_k) \Downarrow \mathsf{f}(u_1, \ldots, u_k)$ *for* $\mathsf{f} \in \Sigma_c$ *of arity* $k$ *and* $t_i \Downarrow u_i$ *for all* $1 \leq i \leq k$*,*
- *if* $t \Downarrow u$ *then* $t\rho \Downarrow u\rho$ *for any bijective* $\rho : \mathcal{N} \to \mathcal{N}$*;*
- *for any term* $t$*, messages* $u$ *and* $v$*, and context* $t'[]$ *built from* $\Sigma$ *and* $\mathcal{N}$*, if* $t \Downarrow u$ *and* $t'[u] \Downarrow v$ *then* $t'[t] \Downarrow v$*;*
- *for any term* $t$ *and messages* $u_1, u_2$ *such that* $t \Downarrow u_1$*, it holds that* $t \Downarrow u_2$ *if, and only if,* $u_1 =_{\mathsf{E}} u_2$*;*
- *if* $t[]$ *is a context built from* $\Sigma$ *and* $\mathcal{N}$*,* $u_1$*,* $u_2$ *are messages such that* $u_1 =_{\mathsf{E}} u_2$*, and,* $t[u_1] \Downarrow v_1$ *for some message* $v_1$*, then* $t[u_2] \Downarrow v_2$ *for some message* $v_2$ *such that* $v_1 =_{\mathsf{E}} v_2$*.*

The relation $\Downarrow$ associates, to any ground term $t$, at most one message up to the equational theory $=_{\mathsf{E}}$. When no such message exists, we say that the *computation fails*; this is noted $t \not\Downarrow$. Remark that the fifth requirement implies that $\Downarrow$ associates to each term either a failure or an equivalence class of $=_{\mathsf{E}}$. As a slight abuse of notation, we may sometimes use directly $t \Downarrow$ as a message, when we know that the computation succeeds and the choice of representative is irrelevant.

**Example 4.** *We precisely define later in Section 2.3 a computation relation giving a semantics to symbols in* $\Sigma_d$ *(see Example 1). It notably satisfies the following:* $\mathsf{sdec}(\mathsf{senc}(t, t_k), t'_k) \Downarrow u$ *for some message* $u$ *if, and only if,* $t$ *computes* $u$ *(*i.e. $t \Downarrow u$*) and* $t_k$ *and* $t'_k$ *both compute the same message* $u_k$ *(*i.e. $t_k \Downarrow u_k$ *and* $t'_k \Downarrow u_k$*). We will also have that* $\mathsf{eq}(t_1, t_2) \Downarrow \mathsf{ok}$ *if, and only if,* $t_1 \Downarrow u$ *and* $t_2 \Downarrow u$ *for some* $u$*. Finally,* $\Downarrow$ *will be such that a failure of a computation of a sub-term propagates at top-level (*i.e. $t \not\Downarrow$ *if* $t' \not\Downarrow$ *for* $t' \in \mathsf{subTerms}(t)$*).*

*As a result, we have for instance the following:* $\mathsf{sdec}(\mathsf{senc}(c, a \oplus b), b \oplus a) \Downarrow c$ *when* $a, b$ *and* $c$ *are names, whereas* $\mathsf{sdec}(\mathsf{senc}(c, a \oplus b), b) \not\Downarrow$*, and* $\mathsf{sdec}(a, b) \oplus \mathsf{sdec}(a, b) \not\Downarrow$*.*

### 2.1.3 Attacker's Knowledge: Recipes & Frames

For modelling purposes, we split the signature $\Sigma$ into two parts, namely $\Sigma_{\mathsf{pub}}$ and $\Sigma_{\mathsf{priv}}$. This is orthogonal to the splitting into $\Sigma_c, \Sigma_d$: it allows to consider some function symbols as *private* forbidding the attacker to use them, helping to model threat models more accurately.

The attacker's knowledge, that is the set of messages he learnt by eavesdropping messages outputted by agents of the protocol, is organised into a *frame*. Formally, a *frame* is a substitution from $\mathcal{W}$ to messages and is noted $\Phi, \Psi$. An attacker builds his own messages by applying public function symbols (*i.e.* symbols in $\Sigma_{\mathsf{pub}}$) to terms he already knows and that are available through variables in $\mathcal{W}$. Formally, a computation done by the attacker is a *recipe*, *i.e.* a term in $\mathcal{T}(\Sigma_{\mathsf{pub}}, \mathcal{W})$. Recipes will be denoted by $R$, $M$, $N$. Note that, although we do not give the attacker the ability to generate fresh names to use in recipes, we obtain exactly the same capability by assuming an infinite supply of public constants without equation in $\Sigma_c \cap \Sigma_{\mathsf{pub}}$. Assume a frame $\Phi$ and a recipe $R$ such that $\mathrm{vars}(R) \subseteq \mathrm{dom}(\Phi)$, then $t = R\Phi$ is the term

computed by the attacker using $R$ over $\Phi$. Either this computation fails (*i.e.* $t \not\Downarrow$) or it computes a message (*i.e.* $t \Downarrow u$). This allows the attacker to observe failure of some computations and equalities or inequalities between results of successful computations.

**Example 5.** *Consider a frame* $\Phi = \{w_1 \mapsto k, w_2 \mapsto \mathsf{senc}(k', k), w_3 \mapsto \mathsf{senc}(m, k')\}$ *for some message* $m$ *and names* $k, k' \in \mathcal{N}$*. From this knowledge, the attacker is able to infer* $k'$ *using the recipe* $R = \mathsf{sdec}(w_2, w_1)$ (i.e. $R\Phi \Downarrow k'$) *and* $m$ *using the recipe* $R' = \mathsf{sdec}(w_3, R) = \mathsf{sdec}(w_3, \mathsf{sdec}(w_2, w_1))$ (i.e. $R'\Phi \Downarrow m$)*. The attacker is also able to observe the failure of the computation* $R_f\Phi \not\Downarrow$ *for* $R_f = \mathsf{sdec}(w_3, w_1)$*.*

## 2.2 Process Algebra

We now define the syntax and semantics of the process algebra we use to model security protocols.

### 2.2.1 Syntax

We consider an infinite set $\mathcal{C}$ of *channel names* such that $\mathcal{C} \cap \mathcal{N} = \emptyset$ and an infinite set $\chi$ of *recursive variables* such that $\chi \cap \mathcal{X} = \emptyset$. Agents of protocols are modelled through ground processes using the grammar in Figure 2.1.

| | | | |
|---|---|---|---|
| $P, Q$ | := | $\Pi\,\mathcal{S}$ | parallel composition |
| | \| | $\mathtt{in}(c, x).P$ | input |
| | \| | $\mathtt{out}(c, t).P$ | output |
| | \| | $\nu\overline{n}.P$ | restriction |
| | \| | $!\,P$ | replication |
| | \| | $\mathsf{let}\ \overline{x} = \overline{t}\ \mathtt{in}\ P\ \mathsf{else}\ Q$ | evaluation |
| | \| | $\mathrm{rec}X.P$ | recursive process |
| | \| | $X$ | recursive variable |
| | \| | $\nu\overline{c}\,\mathsf{out}_{\mathrm{ch}}(a, \overline{c}).P$ | creation of new public channels |

where $\mathcal{S}$ is a multiset of processes, $c, a \in \mathcal{C}$, $\overline{x}$ is a sequence of variables, $\overline{t}$ is a sequence of terms, $t$ is a term, $\overline{n}$ is a sequence of names, $\overline{c}$ is a non-empty sequence of channel names in $\mathcal{C}$ that does not contain $a$, and, $X \in \chi$.

**Figure 2.1** Syntax of processes

**Grammar.** For a multiset[2] $\mathcal{S}$ of processes, $\Pi\,\mathcal{S}$ denotes the parallel composition of all processes in $\mathcal{S}$. When $\mathcal{S}$ contains only two processes, we use the notation $P_1 \mid P_2 \stackrel{\text{def}}{=} \Pi\,\{P_1, P_2\}^{\#}$. This is how parallel composition is usually defined. However, we prefer our definition because of the precise analysis of the structure of processes we carry on later in the thesis (in Chapter 5). When $\mathcal{S}$ is empty, the parallel composition describes the null process that does nothing and that we note $0 \stackrel{\text{def}}{=} \Pi\,\emptyset$. Input and output are standard. A process $\mathtt{in}(c, x).P$ waits for an input on channel

[2]When the context is not clear, we use $\{\}^{\#}$ to denote that the underlying object is a multiset and use $\uplus$ to denote union of multisets.

$c$ and then behaves like $P\{x \mapsto u\}$ where $u$ is the inputted message. A process $\mathtt{out}(c,t).P$ can output the message $u$ on channel $c$ provided that $t$ computes $u$ and then behaves like $P$. The process $\nu\overline{n}.P$ can pick up fresh names $\overline{n}$ that $P$ can then use. The replication $!P$ behaves like an infinite parallel composition $P \mid (P \mid (P \mid \ldots))$. The construct $\mathsf{let}\ \overline{x} = \overline{t}\ \mathtt{in}\ P\ \mathsf{else}\ Q$ allows to write computations and conditionals compactly. Such a process tries to evaluate the terms $\overline{t}$ and if no failure happens, *i.e.* when for all $i$, $t_i \Downarrow u_i$ for some message $u_i$ (noted $\overline{t} \Downarrow \overline{u}$), then the process $P\{x_i \mapsto u_i\}_i$ is executed. Otherwise, *i.e.* when there is some $t_i$ such that $t_i \not\Downarrow$ (noted $\overline{t} \not\Downarrow$), the process $Q$ is executed. Note also that the $\mathsf{let}$ instruction together with the $\mathsf{eq}$ theory as defined in Example 10 can encode the usual conditional construction. Indeed, "$\mathsf{let}\ x = \mathsf{eq}(t_1,t_2)\ \mathtt{in}\ P\ \mathsf{else}\ Q$" will execute $P$ only if the computation succeeds on $\mathsf{eq}(t_1,t_2)$, that is only if $t_1 \Downarrow u_1$, $t_2 \Downarrow u_2$, and $u_1 =_{\mathsf{E}} u_2$ for some messages $u_1$ and $u_2$. A process $\mathrm{rec}X.P$ is a recursive process: it intuitively satisfies the equation $X = P$. For instance, the process $\mathrm{rec}X.(\mathtt{out}(c,\mathsf{ok}).X)$ behaves like an infinite sequence of the output $\mathtt{out}(c,\mathsf{ok})$. Finally, the last construct $\nu\overline{c}\,\mathsf{out}_{\mathrm{ch}}(a,\overline{c}).\,P$ allows to create new fresh channel names $\overline{c}$ that the continuation $P$ may use after making them public by outputting them on the channel $a$.

**Binding and conventions.** The construct $\mathrm{rec}X.P$ binds $X$ in $P$, the construct $\mathtt{in}(c,x).P$ and respectively $\mathsf{let}\ \overline{x} = \overline{v}\ \mathtt{in}\ P\ \mathsf{else}\ Q$ bind $x$ and respectively all variables in $\overline{x}$ in $P$ and $\nu n.P$ binds $n$ in $P$. Finally, in the process $\nu\overline{c}\,\mathsf{out}_{\mathrm{ch}}(a,\overline{c}).P$, channel names in $\overline{c}$ are bound in $P$. We use $\alpha$-renaming and capture-avoiding substitution as is standard. We call *ground* a process having no free variable from $\mathcal{X}$ or $\chi$. We denote by $fc(P)$ and $bc(P)$ the set of *free* and *bound channels* of $P$. For brevity, we sometimes omit "$\mathsf{else}\ 0$" and null processes at the end of processes. We often write $P \cup \mathcal{P}$ instead of $\{P\} \cup \mathcal{P}$.

**Example 6.** *We consider the RFID protocol due to Feldhofer* et al. *as described in [FDW04]. This protocol aims to mutually authenticate an RFID tag with a reader device. The protocol is between an initiator $I$ (the reader) and a responder $R$ (the tag) that share a symmetric key $k$. It can be presented using Alice & Bob notation as follows (we note $\{m\}_k$ the symmetric encryption of a message $m$ with a key $k$):*

$$\begin{array}{lll} 1. & I \to R: & n_I \\ 2. & R \to I: & \{n_I, n_R\}_k \\ 3. & I \to R: & \{n_R, n_I\}_k \end{array}$$

*We consider the term algebra introduced in Example 4. The protocol is modelled by the parallel composition of the processes $P_I$ and $P_R$, corresponding respectively to the roles $I$ and $R$.*

$$P_{\mathsf{Fh}} \stackrel{\mathrm{def}}{=} \nu k.\,(\nu n_I.P_I \mid \nu n_R.P_R)$$

*where $P_I$ and $P_R$ are defined as follows:*

$$\begin{array}{rcl} P_I & \stackrel{\mathrm{def}}{=} & \mathtt{out}(c_I, n_I). \\ & & \mathtt{in}(c_I, x_1). \\ & & \mathsf{let}\ x_2, x_3 = \mathsf{eq}(n_I, \pi_1(\mathsf{sdec}(x_1,k))), \pi_2(\mathsf{sdec}(x_1,k)))\ \mathtt{in} \\ & & \mathtt{out}(c_I, \mathsf{senc}(\langle x_3, n_I\rangle, k)) \end{array}$$

$$\begin{array}{rcl} P_R & \stackrel{\text{def}}{=} & \mathtt{in}(c_R, y_1).\\ & & \mathtt{out}(c_R, \mathsf{senc}(\langle y_1, n_R\rangle, k)).\\ & & \mathtt{in}(c_R, y_2).\\ & & \mathsf{let}\ y_3 = \mathsf{eq}(y_2, \mathsf{senc}(\langle n_R, y_1\rangle, k))\ \mathtt{in}\ 0 \end{array}$$

*Note that, the final null process* 0 *in* $P_R$ *may be replaced by the next protocol to be executed after the Feldhofer protocol.*

**Remark 1** (Replication vs. Recursive Processes)**.** *Recursive processes are standard in* $\pi$*-calculus and are equi-expressive to replication in the full* $\pi$*-calculus [SW03]. This is also the case in the full applied* $\pi$*-calculus notably including private channels and internal communication, where a well-known translation can be used to encode recursive processes into non-recursive processes featuring replications and private channels. However, when internal communication is not allowed as it is the case in our calculus, recursivity is actually strictly more expressive than replication. Even though replication could be encoded using recursive processes (*$!P$ *would be encoded as* $\mathrm{rec}X.(P \mid X)$*), we keep the two constructs because we eventually carry out a precise analysis on processes structure that would behave differently on replicated processes than on replicated processes encoded as recursive processes.*

**Configurations.** The *configurations* (denoted by $G, H, K, A, B$) are pairs $(\mathcal{P}; \Phi)$ where: $\mathcal{P}$ is a multiset of ground processes describing a parallel composition of processes ready to be executed and $\Phi$ is a frame denoting the set of messages the attacker currently knows. Intuitively, $\mathcal{P}$ describes the internal state of the protocol while $\Phi$ describes the state of the environment which coincides with the attacker. Given a configuration $K$, $\Phi(K)$ denotes its second component. We may consider processes as configurations. In such cases, the corresponding frame is $\emptyset$.

Note that free names in configurations represent fresh, private, data. Thus, two configurations $G$ and $H$ equal up to a bijection of names essentially represent the same object. Correspondingly, all security notions we define in Chapter 3 are invariant by bijection of names.

**Example 7** (Resuming Example 6)**.** *The configuration* $(\{P_{\mathsf{Fh}}\}; \emptyset)$ *represents a reader and a tag ready to execute one session of the Feldhofer protocol in presence of an attacker with no prior knowledge.*

### 2.2.2 Internal Reduction

Processes are subject to an internal reduction[3] $\rightsquigarrow_{\mathscr{R}}$ parametrized by a basic relation $\mathscr{R}$. Formally, we assume a relation $\mathscr{R}$ over multisets of processes. The choice of $\mathscr{R}$ gives us enough flexibility to consider some actions to be executed greedily without producing any explicit action at the semantical level (to be defined). For instance, $\mathscr{R}$ could break parallel composition: $(\mathcal{P} \uplus \{\Pi\mathcal{S}\})\ \mathscr{R}\ (\mathcal{P} \uplus \mathcal{S})$ yielding an internal reduction breaking all parallel compositions occurring

[3]Our notion of *internal reduction* is close to but different from the one in [AF01]. It notably plays the role of the *structural equivalence* from [AF01] and can be chosen such that it plays parts of the role of the *internal reduction* as defined in [AF01].

at top level. In Chapter 4, we eventually define a generic class of relations $\mathscr{R}$ one may use but, for the moment and for the sake of clarity, one can fix an empty relation $\mathscr{R} = \emptyset$.

Once a relation $\mathscr{R}$ has been chosen (*e.g.* $\mathscr{R} = \emptyset$), the induced internal relation $\rightsquigarrow_{\mathscr{R}}$ applies $\mathscr{R}$ under parallel compositions and reorganises the structure of parallel compositions as formally defined next.

**Definition 3.** *The relation $\rightsquigarrow_{\mathscr{R}}$ is the least relation over multisets of processes containing $\mathscr{R}$ such that:*

- *for any multisets of processes $\mathcal{P}, \mathcal{S}, \mathcal{S}'$, it holds that $\mathcal{P} \uplus \{\Pi(\mathcal{S} \uplus \{\Pi\mathcal{S}'\})\} \rightsquigarrow_{\mathscr{R}} \mathcal{P} \uplus \{\Pi(\mathcal{S} \uplus \mathcal{S}')\}$ (nested parallel compositions collapse);*
- *for any multiset of processes $\mathcal{P}$, any process $P$, it holds that $\mathcal{P} \uplus \{\Pi\{P\}\} \rightsquigarrow_{\mathscr{R}} \mathcal{P} \uplus \{P\}$ (no parallel composition of a single process);*
- *for any multisets of processes $\mathcal{P}, \mathcal{S}, \mathcal{S}'$, if $\mathcal{S} \rightsquigarrow_{\mathscr{R}} \mathcal{S}'$ then $\mathcal{P} \uplus \{\Pi\mathcal{S}\} \rightsquigarrow_{\mathscr{R}} \mathcal{P} \uplus \{\Pi\mathcal{S}'\}$ (congruence w.r.t. parallel composition).*

We lift $\rightsquigarrow_{\mathscr{R}}$ over configurations by applying it on the first component only. We assume that $\mathscr{R}$ is chosen in such a way that the induced relation $\rightsquigarrow_{\mathscr{R}}$ is *terminating* (*i.e.* no infinite chain decreasing w.r.t. $\rightsquigarrow_{\mathscr{R}}$) and *confluent* (*i.e.* if $\mathcal{P} \rightsquigarrow_{\mathscr{R}} \mathcal{Q}_1$ and $\mathcal{P} \rightsquigarrow_{\mathscr{R}} \mathcal{Q}_2$ then there exists $\mathcal{Q}$ such that $\mathcal{Q}_1 \rightsquigarrow_{\mathscr{R}} \mathcal{Q}$, $\mathcal{Q}_2 \rightsquigarrow_{\mathscr{R}} \mathcal{Q}$). In Chapter 4, we define a generic class of relations $\mathscr{R}$ (including the empty relation) and prove that such relations always yield internal reductions that are terminating and confluent. The internal reduction thus yields normal forms: for any multiset of processes $\mathcal{P}$, there exists a unique $\mathcal{Q}$ that we note $\mathrm{NF}_{\mathscr{R}}(\mathcal{Q})$ such that $\mathcal{P} \rightsquigarrow^*_{\mathscr{R}} \mathcal{Q}$ and there is no $\mathcal{Q}'$ such that $\mathcal{Q} \rightsquigarrow_{\mathscr{R}} \mathcal{Q}'$. We now only consider configurations and multisets of processes in normal forms by implicitly reducing multisets of processes.

**Remark 2.** *Note that for any process $P = \Pi\mathcal{S}$ in a configuration $K$ in normal form, no process in $\mathcal{S}$ is a parallel composition (*i.e. *of the form $\Pi\mathcal{S}'$). Remark that as a sub-case of the first item of Definition 3, we also have that $\mathcal{P} \uplus \{\Pi(\mathcal{S} \uplus \{0\})\} \rightsquigarrow_{\mathscr{R}} \mathcal{P} \uplus \{\Pi\mathcal{S}\}$ (0 is the identity element of $\Pi$) independently of $\mathscr{R}$.*

**Example 8.** *For instance, one could reduce $\mathcal{P} = \{\Pi\{\Pi\{Q, 0, 0\}, 0\}\} \rightsquigarrow_{\emptyset} \{\Pi\{\Pi\{Q\}\}\} \rightsquigarrow_{\emptyset} \{Q\}$. Moreover if $Q$ is not a parallel composition, it holds that $\mathrm{NF}_{\emptyset}(\mathcal{P}) = \{Q\}$.*

### 2.2.3 Semantics

The operational semantics of processes is given by a labelled transition system (LTS) over configurations in normal forms (w.r.t. $\rightsquigarrow_{\mathscr{R}}$). The semantics is given in Figure 2.2. This labelled operational semantics allows one to avoid the quantification over all contexts when analysing a protocol in presence of an arbitrary attacker and is therefore more amenable to automation.

The Out rule allows to trigger an output provided that the computation of the outputted term does not fail. Note that the resulting outputted message is added to the current frame modelling the fact that the attacker can eavesdrop all outputted messages. The In rule can be

| | |
|---|---|
| Par | $(\{\Pi\,\mathcal{S}\} \uplus \mathcal{P}; \Phi) \xrightarrow{\tau_{\text{ll}}} (\mathcal{P} \uplus \mathcal{S}; \Phi)$ |
| In | $(\{\mathtt{in}(c,x).P\} \uplus \mathcal{P}; \Phi) \xrightarrow{\mathtt{in}(c,R)} (\{P\{x \mapsto u\}\} \uplus \mathcal{P}; \Phi)$ where $R$ is a recipe such that $R\Phi \Downarrow u$ |
| Out | $(\{\mathtt{out}(c,t).P\} \uplus \mathcal{P}; \Phi) \xrightarrow{\mathtt{out}(c,w)} (\{P\} \uplus \mathcal{P}; \Phi \uplus \{w \mapsto u\})$ where $w$ is a fresh variable and $t \Downarrow u$ |
| New | $(\{\nu\overline{n}.P\} \uplus \mathcal{P}; \Phi) \xrightarrow{\tau_\nu} (\{P\} \uplus \mathcal{P}; \Phi)$ where $\overline{n}$ are fresh names from $\mathcal{N}$ |
| Repl | $(\{!\,P\} \uplus \mathcal{P}; \Phi) \xrightarrow{\tau_!} (\{P, !\,P\} \uplus \mathcal{P}; \Phi)$ |
| Let | $(\{\mathsf{let}\ \overline{x} = \overline{t}\ \mathtt{in}\ P\ \mathsf{else}\ Q\} \uplus \mathcal{P}; \Phi) \xrightarrow{\tau_{\mathsf{then}}} (\{P\{\overline{x} \mapsto \overline{u}\}\} \uplus \mathcal{P}; \Phi)$ when $\overline{t} \Downarrow \overline{u}$ for some $\overline{u}$ |
| Let-Fail | $(\{\mathsf{let}\ \overline{x} = \overline{t}\ \mathtt{in}\ P\ \mathsf{else}\ Q\} \uplus \mathcal{P}; \Phi) \xrightarrow{\tau_{\mathsf{else}}} (\{Q\} \uplus \mathcal{P}; \Phi)$ when $\overline{t} \not\Downarrow$ |
| $\mathsf{Out}_{\mathrm{ch}}$ | $(\{\nu\overline{c}\,\mathsf{out}_{\mathrm{ch}}(a,\overline{c}).\,P\} \uplus \mathcal{P}; \Phi) \xrightarrow{\mathtt{outch}(a,\overline{c})} (\{P\} \uplus \mathcal{P}; \Phi)$ where $\overline{c}$ are fresh |
| Unfold | $(\{\mathrm{rec}X.P\} \uplus \mathcal{P}; \Phi) \xrightarrow{\tau_r} (\{P\{X \mapsto \mathrm{rec}X.P\}\} \uplus \mathcal{P}; \Phi)$ |

**Figure 2.2** Semantics for processes

used to trigger an input for a recipe built on the current frame. The inputted message injected into the process that performed that action is given by the computation of the recipe on the current frame. The latter models the injection capabilities of the attacker. Overall, observe that this semantics models an attacker controlling all the network. Finally, note that the guard condition of the rule New (*i.e.* names $\overline{n}$ must be fresh) is never blocking since it is always possible to $\alpha$-rename $\overline{n}$.

All $\tau$ actions (*i.e.* $\tau_{\text{ll}}, \tau_\nu, \tau_!, \tau_{\mathsf{then}}, \tau_{\mathsf{else}}, \tau_r$) are unobservable actions from the attacker's point of view. They represent progress of the internal control points of the different agents involved in the protocol that have no observable effect. We still label them precisely because of the annotations we eventually equip the semantics with.

As usual, the relation $\xrightarrow{\alpha_1.\ldots.\alpha_n}$ between configurations (where $\alpha_1.\ldots.\alpha_n$ is a trace, *i.e.* a sequence of actions) is defined as the (labelled) reflexive and transitive closure of $\xrightarrow{\alpha}$. We denote by $bc(\mathsf{tr})$ the bound channels of a trace $\mathsf{tr}$, *i.e.* all the channels that occur in second argument of an action $\nu\overline{c}\,\mathsf{out}_{\mathrm{ch}}(a,\overline{c})$ in $\mathsf{tr}$, and we consider traces where channels are bound at most once.

**Example 9.** *Continuing Example 7. We have that:* $(\{P_{\mathsf{Fh}}\}; \emptyset) \xrightarrow{\mathsf{tr}} (\{0,0\}^{\#}; \Phi_0)$ *where* $\mathsf{tr}$ *and* $\Phi_0$ *are as follows, for fresh names* $k', n'_I, n'_R \in \mathcal{N}$*:*

$$\begin{aligned}\mathsf{tr} = \;& \tau_\nu.\tau_{\text{ll}}.\tau_\nu.\tau_\nu.\mathtt{out}(c_I,w_1).\mathtt{in}(c_R,w_1).\mathtt{out}(c_R,w_2).\mathtt{in}(c_I,w_2).\tau_{\mathsf{then}}.\mathtt{out}(c_I,w_3).\mathtt{in}(c_R,w_3).\tau_{\mathsf{then}}\\ \Phi_0 = \;& \{w_1 \mapsto n'_I,\ w_2 \mapsto \mathsf{senc}(\langle n'_I, n'_R\rangle, k'),\ w_3 \mapsto \mathsf{senc}(\langle n'_R, n'_I\rangle, k')\}.\end{aligned}$$

*This execution corresponds to a normal execution of one session of the protocol.*

We prove in Chapter 4 that the choice of $\mathscr{R}$ (from which stems the induced internal reduction $\rightsquigarrow_{\mathscr{R}}$) has no impact on the induced security notions providing that it only reduces processes

as executing $\tau$-actions would do. We will provide a generic class of such choices of $\mathscr{R}$.

## 2.3 Instances of Term Algebras

We show in this section two means to operationally define computation relations corresponding to the ways it is done in respectively ProVerif and Apte. In ProVerif, this is done through rewriting systems over terms while in Apte this is done thanks to equations over terms equipped with a validity predicate.

### 2.3.1 Computation Relation Through Rewriting Systems

A computation relation can be obtained from a *rewriting system* which describes basic rules specifying how destructors affect their arguments.

**Definition 4.** *A* rewriting system *is an ordered set of rewriting rules of the form* $\mathsf{g}(u_1, \ldots, u_n) \to u$ *where* $\mathsf{g} \in \Sigma_d$*, and* $u, u_1, \ldots, u_n$ *are constructors terms (*i.e. *in* $\mathcal{T}(\Sigma_c, \mathcal{X})$*).*

*A ground term* $t$ *can be rewritten into* $t'$ *(noted* $t \rightsquigarrow t'$*) if there is a position* $p$ *in* $t$*, a rewriting rule* $\mathsf{g}(u_1, \ldots, u_n) \to u$ *and a substitution* $\theta$ *from variables to messages such that* $t|_p = \mathsf{g}(v_1, \ldots, v_n)$*,* $v_1 =_{\mathsf{E}} u_1\theta, \ldots, v_n =_{\mathsf{E}} u_n\theta$*, and,* $t' = t[u\theta]_p$ *(*i.e. $t$ *in which the sub-term at position* $p$ *has been replaced by* $u\theta$*). If for some position* $p$*, there is more than one rewriting rule that can be applied, the one occurring first in the ordered set is applied. We note* $\rightsquigarrow^*$ *the reflexive, transitive closure of* $\rightsquigarrow \cup =_{\mathsf{E}}$*.*

**Proposition 1.** *If* $\mathcal{R}$ *is a rewriting system then the relation* $\bullet \Downarrow \bullet$ *induced by* $\mathcal{R}$ *(*i.e. $t \Downarrow u$ *if, and only if,* $t \rightsquigarrow^* u$ *and* $u$ *is a message) is a computation relation.*

*Proof.* We prove all requirements separately:

- Since $\rightsquigarrow^*$ is reflexive, one has $n \Downarrow n$ for any $n \in \mathcal{N}$;
- Consider a $\mathsf{f} \in \Sigma_c$ of arity $k$, some terms $t_i$ and messages $u_i$ such that $t_i \Downarrow u_i$ for all $1 \leq i \leq k$. We thus have $t_i \rightsquigarrow^* u_i$ for each $i$. One then deduces $\mathsf{f}(t_1, \ldots, t_k) \Downarrow \mathsf{f}(u_1, \ldots, u_k)$ since $\mathsf{f}(t_1, \ldots, t_k) \rightsquigarrow^* \mathsf{f}(u_1, t_2, \ldots, t_k) \ldots \rightsquigarrow^* \mathsf{f}(u_1, \ldots, u_k)$. Indeed, $=_{\mathsf{E}}$ is stable by application of constructor symbols, and each application of reduction rules on some $t_i$ can be adapted by transforming the position $p$ into $i \cdot p$.
- We prove that for any term $t$, if $t \Downarrow u$ for some message $u$ and $\rho : \mathcal{N} \to \mathcal{N}$ is a bijective mapping then $t\rho \Downarrow u\rho$. First, we remark that $=_{\mathsf{E}}$ is stable by bijection of names. It then suffices to show that if $t \rightsquigarrow t'$ then $t\rho \rightsquigarrow t'\rho$. By hypothesis, there is a position $p$ and a rewriting rule $\mathsf{g}(u_1, \ldots, u_n) \to u$ such that $t|_p = \mathsf{g}(v_1, \ldots, v_n)$, $v_1 =_{\mathsf{E}} u_1\theta, \ldots, v_n =_{\mathsf{E}} u_n\theta$ for some substitution $\theta$, $t' = t[u\theta]_p$, and, for all $i$, $u_i\theta$ is a message. We let $v'_i = v_i\rho$ and $\theta' = \rho \circ \theta$. It holds that $(t\rho)|_p = \mathsf{g}(v'_1, \ldots, v'_n)$ and $v'_1 =_{\mathsf{E}} u_1\theta', \ldots, v'_n =_{\mathsf{E}} u_n\theta'$, and $t\rho \rightsquigarrow t\rho[u\theta']_p$. Finally, note that $t\rho[u\theta']_p = t'\rho$ since $u\theta' = (u\theta)\rho$. Therefore, one has $t\rho \rightsquigarrow t'\rho$.

- Consider a term $t$ and messages $u_1, u_2$ such that $t \Downarrow u_1$. If $u_1 =_\mathsf{E} u_2$, $t \Downarrow u_2$ follows from the fact that $=_\mathsf{E}$ is included in $\rightsquigarrow^*$. We now assume that $t \Downarrow u_2$ and shall prove that $u_1 =_\mathsf{E} u_2$. By hypothesis, there exist two reductions $t \rightsquigarrow^* u_1$ and $t \rightsquigarrow^* u_2$. In order to prove $u_1 =_\mathsf{E} u_2$, we intuitively show that when $t_1$ and $t_2$ are two terms that are equal modulo $=_\mathsf{E}$ on some sub-terms and $t_1 \rightsquigarrow^* t_1'$, $t_2 \rightsquigarrow^* t_2'$ then the two terms $t_1'$ and $t_2'$ are equal modulo (i) $=_\mathsf{E}$ on some sub-terms and possibly (ii) applications of reduction rules. To that end, we formally define $\equiv_\mathsf{E}$ to be the smallest equivalence relation such that for any term with hole $t[]$ and messages $u_1$ and $u_2$ then $t[u] \equiv_\mathsf{E} t[v]$ when $u =_\mathsf{E} v$. We now prove the following intermediate result:
  *For any two terms $t_1 \equiv_\mathsf{E} t_2$ and two reductions $t_1 \rightsquigarrow t_1'$ and $t_2 \rightsquigarrow t_2'$, either $t_1' \equiv_\mathsf{E} t_2'$ or there exist two terms $t_1'' \equiv_\mathsf{E} t_2''$ and two reductions $t_1' \rightsquigarrow t_1''$ and $t_2' \rightsquigarrow t_2''$.*
  Remark that this intermediate result would conclude the proof. Indeed, applying repeatedly this result on the two finite reductions $t \rightsquigarrow^* u_1$ and $t \rightsquigarrow^* u_2$ entails that $u_1 \equiv_\mathsf{E} u_2$ (since no reduction can be applied on messages) implying $u_1 =_\mathsf{E} u_2$. We now prove the intermediate result. We thus consider two terms $t_1 \equiv_\mathsf{E} t_2$ and two reductions $t_1 \rightsquigarrow t_1'$ and $t_2 \rightsquigarrow t_2'$. There must be two positions $p_1$ and $p_2$ such that for $i \in \{1, 2\}$, there is a reduction rule $\mathsf{g}_i(u_i^1, \ldots, u_i^{n_i}) \to u_i$ in the rewriting system such that $t_{i|_{p_i}} = \mathsf{g}_i(v_i^1, v_i^2, \ldots, v_i^{n_i})$, $v_i^j = u_i^j\theta_i$, $t_i' = t_i[u_i\theta_i]_{p_i}$. We distinguish two cases whether $p_1 = p_2$ or not. If $p_1 = p_2$ then $t_1 \equiv_\mathsf{E} t_2$ implies $\mathsf{g}_1 = \mathsf{g}_2$ and $v_1^j =_\mathsf{E} v_2^j$ for all $j$. Moreover, the same reduction rule is applied on $t_1$ and $t_2$. Indeed, the former implies that the same rules can be applied on $t_{1|_{p_1}}$ and $t_{2|_{p_2}}$ but, by definition of $\rightsquigarrow$, the first one occurring in the ordered set of reduction rules was actually applied to $t_1$ and $t_2$. We thus have $u_1\theta_1 =_\mathsf{E} u_2\theta_2$. Therefore, it holds that $t_1' \equiv_\mathsf{E} t_2'$ concluding the proof. Otherwise, it must be the case that $p_1$ is not a prefix of $p_2$ nor the converse. Indeed, strict-sub terms below the positions where the rules are applied must be messages. Additionally, since $t_i$ has a destructor symbol $\mathsf{g}_i$ at position $p_i$ and $t_1 \equiv_\mathsf{E} t_2$, then, $t_i$ has also a destructor symbol $\mathsf{g}_{2-i}$ at position $p_{2-i}$ and its arguments are equal modulo $=_\mathsf{E}$. Therefore, $t_1 \equiv_\mathsf{E} t_2$ implies $(t_i[u_i\theta_i])_{|_{p_{2-i}}} \equiv_\mathsf{E} \mathsf{g}_i(v_{2-i}^1, v_{2-i}^2, \ldots, v_{2-i}^{n_{2-i}})$. The reduction rule which has been applied on $t_i$ can thus be applied on $t_{2-i}'$ yielding $t_{2-i}' \rightsquigarrow t_{2-i}''$. Moreover, $t_{1|_{p_i}}'' \equiv_\mathsf{E} t_{2|_{p_i}}''$ for all $i \in \{1, 2\}$. Since at other positions (*i.e.* positions that are not a prefix or a suffix of $p_1$ or $p_2$), the same term is present in $t_i''$, $t_i'$ and $t_i$, and $t_1 \equiv_\mathsf{E} t_2$, one can deduce $t_1'' \equiv_\mathsf{E} t_2''$ concluding the proof.

- Let $t[]$ be a context built from $\Sigma$ and $\mathcal{N}$, $u_1$, $u_2$ be messages. We assume $u_1 =_\mathsf{E} u_2$, and, $t[u_1] \Downarrow v_1$ for some message $v_1$. By hypothesis, we thus have that $t[u_1] \rightsquigarrow^* v_1$. Let us prove by induction on the reduction $t[u_1] \rightsquigarrow^* v_1$ that $t[u_2] \rightsquigarrow^* v_1$. Let $p$ be the position of the hole in $t$. We reason by case analysis on the first step of $t[u_1] \rightsquigarrow^* v_1$. If $t[u_1] =_\mathsf{E} v'$ then $v' =_\mathsf{E} v_1$ and $t[u_1]$ and $v'$ are messages and so is $t[u_2]$. In that case, we deduce $t[u_2] \rightsquigarrow^* v_1$ from the stability of $=_\mathsf{E}$ by application of constructor symbols. If $t[u_1] \rightsquigarrow t' \rightsquigarrow^* v_1$ and the reduction rule is applied to a position above $u_1$ (*i.e.* at a position $p'$ that is a prefix of $p$) then the exact reduction rule can be applied to $t[u_2]$ as well, leading to $t'$: $t[u_2] \rightsquigarrow t'$. Indeed, all arguments of the destructor are taken modulo $=_\mathsf{E}$. We conclude by transitivity

of $\rightsquigarrow^*$. Otherwise, $t[u_1] \rightsquigarrow t' \rightsquigarrow^* v_1$ and the reduction rule is applied to a position that is not above nor below $u_1$. Indeed, $u_1$ is a message and has thus no destructor symbol. Hence, no reduction rule can be applied on $u_1$. We thus have some term with hole $t''[]$ such that $t' = t''[u_1]$ and thus $t[u_1] \rightsquigarrow t''[u_1]$. Since the reduction rule is applied on a strict sub-term of $t$, one has $t[u_2] \rightsquigarrow t''[u_2]$. We conclude $t[u_2] \rightsquigarrow^* v_1$ by inductive hypothesis. Hence $t[u_2] \Downarrow v_1$.

- Consider a term $t$, some messages $u$ and $v$, and a context $t'[]$ built from $\Sigma$ and $\mathcal{N}$ such that $t \Downarrow u$ and $t'[u] \Downarrow v$. Let us prove that $t'[t] \rightsquigarrow^* v$ (implying $t'[t] \Downarrow v$). By hypothesis, we have the two following reductions: $t'[u] \rightsquigarrow^* v$ and $t \rightsquigarrow^* u$. First, we show that $t'[t] \rightsquigarrow^* t'[u]$. Indeed, one can modify applications of reduction rules on $t$ to be applied to $t'[t]$ by modifying the position where they are applied. We conclude $t'[t] \rightsquigarrow^* v$ by transitivity of $\rightsquigarrow^*$.

□

**Example 10.** *The properties of symbols in $\Sigma_d$ (see Example 1) are reflected through the following ordered set of rewriting rules:*

$$\mathsf{sdec}(\mathsf{senc}(x,y),y) \to x \qquad \mathsf{eq}(x,x) \to \mathsf{ok}$$
$$\pi_i(\langle x_1, x_2\rangle) \to x_i \quad \textit{for } i \in \{1,2\}.$$
$$\mathsf{neq}(x,x) \to \mathsf{no} \qquad \mathsf{neq}(x,y) \to \mathsf{ok}$$

*Thanks to Proposition 1, we deduce from this rewriting system a computation relation as defined in Definition 2. Note that the resulting computation relations satisfies the relations given in Example 4.*

**Remark 3.** *Thanks to the fact that rules are given priorities, we are able to give a semantics for* neq *such that* $\mathsf{neq}(t_1,t_2) \Downarrow \mathsf{ok}$ *if, and only if, $t_1$ and $t_2$ can be reduced to messages that are not equal modulo $=_{\mathsf{E}}$. In* ProVerif*, the priority over rules is achieved by using the* `otherwise` *key word which specifies a rule that should be used only if previous ones could not. The extension has been introduced with [CB13].*

### 2.3.2 Computation Relation Through an Equational Theory

Another way to obtain computation relations is based on an equational theory over terms extending $=_{\mathsf{E}}$. We consider a set of equations given as a relation over terms and consider the induced extended equational theory as shown below.

**Definition 5.** *Let $\mathsf{E}_d : \mathcal{T}(\Sigma,\mathcal{X}) \times \mathcal{T}(\Sigma,\mathcal{X})$ be a relation over terms. The equational theory $\equiv_{\mathsf{E}_d}$ is the smallest equivalence relation over $\mathcal{T}(\Sigma,\mathcal{N}\cup\mathcal{X})$ containing $\mathsf{E} \uplus \mathsf{E}_d$, stable by substitution from $\mathcal{X}$ to $\mathcal{T}(\Sigma,\mathcal{N}\cup\mathcal{X})$ and stable by application of function symbols in $\Sigma$.*

Note that $=_{\mathsf{E}} \subseteq \equiv_{\mathsf{E}_d}$: if $u =_{\mathsf{E}} v$ for some messages $u, v$ then $u \equiv_{\mathsf{E}_d} v$. We say that a term $t$ computes a message $u$ (*i.e.* $t \Downarrow u$) if $t \equiv_{\mathsf{E}_d} u$ and $t$ is *valid* as defined below.

**Definition 6.** *A term $t$ is said* valid*, denoted* $\mathsf{valid}(t)$*, when for any $t' \in \mathsf{subTerms}(t)$, it holds that there exists a message $v'$ such that $t' \equiv_{\mathsf{E}_d} v'$.*

Intuitively, the validity condition expresses the fact that if a failure occurs during a bottom-up computation of the term then it propagates through the whole term.

Finally, we can prove that the computation relation induced by such equational theories fulfils all requirements of Definition 2 provided that $\equiv_{\mathsf{E}_d}$ is a conservative extension of $=_{\mathsf{E}}$ as defined next.

**Definition 7.** *We say that $\equiv_{\mathsf{E}_d}$ is a conservative extension of $=_{\mathsf{E}}$ when for all messages $u_1, u_2$, if $u_1 \equiv_{\mathsf{E}_d} u_2$ then $u_1 =_{\mathsf{E}} u_2$.*

**Proposition 2.** *Assuming that $\equiv_{\mathsf{E}_d}$ is a conservative extension of $=_{\mathsf{E}}$, the relation $t \Downarrow u$ induced by the validity predicate (*i.e. *$t \Downarrow u$ if $t$ is valid and $u$ a message such that $t \equiv_{\mathsf{E}_d} u$) is a computation relation.*

*Proof.* We prove all requirements separately:

- Consider $n \in \mathcal{N}$. Since $\equiv_{\mathsf{E}_d}$ is reflexive, one has $n \equiv_{\mathsf{E}_d} n$. We conclude from $\mathsf{subTerms}(n) = \{n\}$ and $\mathsf{valid}(n)$.
- $\mathsf{f}(t_1, \ldots, t_k) \Downarrow \mathsf{f}(u_1, \ldots, u_k)$ for $\mathsf{f} \in \Sigma_c$ of arity $k$ and $t_i \Downarrow u_i$ for all $1 \leq i \leq k$ follows from the stability of $\equiv_{\mathsf{E}_d}$ by application of function symbols in $\Sigma_c \subseteq \Sigma$.
- We prove by structural induction on the term $t$ that if $t \Downarrow u$ for some message $u$ and $\rho : \mathcal{N} \to \mathcal{N}$ is a bijective mapping then $t\rho \Downarrow u\rho$. Since $t \Downarrow u$, it holds that $t \equiv_{\mathsf{E}_d} u$ and $\mathsf{valid}(t)$. We trivially have that $\equiv_{\mathsf{E}_d}$ is stable by bijection of name; hence $t\rho \equiv_{\mathsf{E}_d} u\rho$. Since $u$ is a message and $\rho$ is a bijection of names, we have that $u\rho$ is a message. Consider now a strict sub-term $t'$ of $t$. Since $\mathsf{valid}(t)$, there must be a message $u'$ such that $t' \equiv_{\mathsf{E}_d} u'$. Hence $t' \Downarrow u'$. By inductive hypothesis, it holds that $t'\rho \Downarrow u'\rho$ and $u'\rho$ is a message. Since any strict sub-term of $t\rho$ is a strict sub-term of $t$ on which $\rho$ is applied, we have shown that $t\rho$ is valid. Therefore, $t\rho \Downarrow u\rho$.
- Consider a term $t$, some messages $u$ and $v$, and a context $t'[]$ built from $\Sigma$ and $\mathcal{N}$ such that $t \Downarrow u$ and $t'[u] \Downarrow v$ . Let us prove that $t'[t] \Downarrow v$. By hypothesis, we have that $t \equiv_{\mathsf{E}_d} u$ and $t'[u] \equiv_{\mathsf{E}_d} v$. By stability of $\equiv_{\mathsf{E}_d}$ by application of function symbols, one has $t'[t] \equiv_{\mathsf{E}_d} t'[u]$ and thus, by transitivity of $\equiv_{\mathsf{E}_d}$, $t'[t] \equiv_{\mathsf{E}_d} v$. It remains to show $\mathsf{valid}(t'[t])$. We already proved that $t'[t] \equiv_{\mathsf{E}_d} v$ where $v$ is a message. Consider now a strict sub-term $t_s$ of $t'[t]$. If it is a sub-term of $t$ then $\mathsf{valid}(t_s)$ follows from $t \Downarrow u$. If it is a sub-term of $t'$ without hole then $\mathsf{valid}(t_s)$ follows from $t'[u] \Downarrow v$. Otherwise, there exists $t'_s$ a sub-term of $t'$ such that $t_s = t'_s[t]$. Applying the same argument as before, we deduce that $t'_s[t] \equiv_{\mathsf{E}_d} t'_s[u] \equiv_{\mathsf{E}_d} v'$ for some message $v'$.
- Consider a term $t$ and messages $u_1, u_2$ such that $t \Downarrow u_1$. If $u_1 =_{\mathsf{E}} u_2$, $t \Downarrow u_2$ follows from $=_{\mathsf{E}} \subseteq \equiv_{\mathsf{E}_d}$. If $t \Downarrow u_2$, then $u_1 =_{\mathsf{E}} u_2$ is implied by the fact that $\equiv_{\mathsf{E}_d}$ is a conservative extension of $=_{\mathsf{E}}$.

- Let $t[]$ be a context built from $\Sigma$ and $\mathcal{N}$, $u_1$, $u_2$ be messages. We assume $u_1 =_{\mathsf{E}} u_2$, and, $t[u_1] \Downarrow v_1$ for some message $v_1$. Let us prove that $t[u_2] \Downarrow v_2$ for some message $v_2$ such that $v_1 =_{\mathsf{E}} v_2$. By hypothesis, we have that $t[u_1] \equiv_{\mathsf{E}_d} v_1$ and $\mathsf{valid}(t[u_1])$. Since $\equiv_{\mathsf{E}_d}$ is stable by application of function symbols and $u_1 \equiv_{\mathsf{E}_d} u_2$, we also have $t[u_2] \equiv_{\mathsf{E}_d} t[u_1] \equiv_{\mathsf{E}_d} v_1$. Since this reasoning can also be applied for all sub-terms of $t[u_2]$, we deduce $\mathsf{valid}(t[u_2])$.

□

**Example 11.** *The properties of symbols in $\Sigma_d$ (see Example 1) are reflected through the following equations:*

$$\mathsf{sdec}(\mathsf{senc}(x,y),y)\ \mathsf{E}_d\ x \qquad \mathsf{eq}(x,x)\ \mathsf{E}_d\ \mathsf{ok}$$
$$\pi_i(\langle x_1,x_2\rangle)\ \mathsf{E}_d\ x_i \quad \textit{for } i \in \{1,2\}.$$

*It is easy to see that the induced $\equiv_{\mathsf{E}_d}$ is a conservative extension of $=_{\mathsf{E}}$. Indeed, each equation involves only one destructor symbol on one side and a constructor term on the other side and all equations involve pairwise distinct destructor symbols. Therefore, if two messages $u_1, u_2$ are such that $u_1 \equiv_{\mathsf{E}_d} u_2$ involving an equation from $\mathsf{E}_d \backslash \mathsf{E}$ then, since it introduces a destructor symbol that cannot be removed using other equations, the same equation has to applied (but in the other direction). The two applications of this equation must cancel out.*

*The induced computation relation (given by Proposition 2) coincides with the one given in Example 10 except for* neq*. Indeed, equational theories are not flexible enough to give the expected semantics to this destructor. On the contrary, equational theories are better at specifying reductions involving interactions between destructors (*e.g. *for $\mathrm{dest}_1, \mathrm{dest}_2 \in \Sigma_d$, equations $\mathrm{dest}_1(\mathrm{dest}_2(x))\ \mathsf{E}_d\ \mathrm{const}(x)$ cannot directly be specified using reduction rules).*

**Example 12** (Apte's Term Algebra)**.** *We now define the fixed term algebra used in the tool* Apte*. The signature used in* Apte *is $\Sigma = \Sigma_c \cup \Sigma_d$ with:*

$$\begin{aligned}\Sigma_c &= \Sigma_0 \cup \{\mathsf{aenc}, \mathsf{pk}, \mathsf{enc}, \mathsf{hash}, \mathsf{sign}, \mathsf{vk}, \langle\rangle\}\\ \Sigma_d &= \{\mathsf{adec}, \mathsf{dec}, \mathsf{check}, \pi_1, \pi_2\}\end{aligned}$$

*where $\Sigma_0$ may contain some additional user-defined constants (*i.e. *constructor symbols of arity 0) and attacker's nonces. The equational theory is generated from no equation $\mathsf{E} = \emptyset$. We thus have $u =_{\mathsf{E}} v$ if, and only if, $u = v$. To take into account the properties of the destructor symbols, we consider the equational theory $\equiv_{\mathsf{E}_d}$ generated by the following equations over terms:*

$$\begin{aligned}\mathsf{adec}(\mathsf{aenc}(x,\mathsf{pk}(y)),y) &\ \mathsf{E}_d\ x\\ \pi_1(\langle x_1,x_2\rangle) &\ \mathsf{E}_d\ x_1\\ \pi_2(\langle x_1,x_2\rangle) &\ \mathsf{E}_d\ x_2\\ \mathsf{sdec}(\mathsf{senc}(x,y),y) &\ \mathsf{E}_d\ x\\ \mathsf{check}(\mathsf{sign}(x,y),\mathsf{vk}(y)) &\ \mathsf{E}_d\ x\end{aligned}$$

*It is easy to see that the induced $\equiv_{\mathsf{E}_d}$ is a conservative extension of $=_{\mathsf{E}}$ for the same reason given in the previous example (*i.e. *Example 11). By Proposition 2, we thus have that $\mathsf{E}_d$ induces a computation relation. For instance, we have $\pi_2(\mathsf{adec}(\mathsf{aenc}(\langle n, \mathsf{pk}(ska)\rangle, \mathsf{pk}(skb)), skb)) \Downarrow \mathsf{pk}(ska)$. We also have that $\pi_1(\langle \mathsf{ok}, \mathsf{sdec}(\mathsf{senc}(a,k),k')\rangle) \not\Downarrow$ since it is not valid (because $\mathsf{sdec}(\mathsf{senc}(a,k),k') \not\Downarrow$).*

Chapter 3

# Modelling Security Goals

Security goals are expressed as predicates over executions or sets of executions. Some can be expressed as the reachability of some given states; we describe them informally in Section 3.1. However, this thesis focuses on privacy goals that are often modelled based on a more complex notion called behavioural equivalence. There are different variants of behavioural equivalence modelling more or less precisely attacker's capabilities. In Section 3.2, we define the most appropriate one to *model and automatically verify* security goals, which is called *trace equivalence*, and discuss its relations with other behavioural equivalences. Further, we explain how to model some privacy goals using trace equivalence in Section 3.3.

## 3.1 Reachability Properties

*Reachability properties* (also called *trace properties*) are statements that something bad never occurs on any execution trace of a protocol. Such properties are usually defined through predicates over configurations and sometimes over executions. We informally define below some well-known reachability properties. Obviously, there is a flurry of other reachability properties but we do not list them since we rather focus on equivalence-based properties.

**Secrecy.** This property (also called confidentiality) concerns a message used by the protocol and essentially models the fact that this message remains secret (*i.e.* not known by the attacker). For instance, considering a configuration $K$ and a message $u$, we say that $K$ *keeps* $u$ *secret* if for any execution of $K$, the attacker is not able to deduce $u$: *i.e.* for any $K \xrightarrow{\mathsf{tr}} K'$, there is no recipe $R$ over $\Phi(K')$ such that $R\Phi(K') \Downarrow u$.

**Example 13.** *For example, the configuration* $(\{\mathtt{out}(c, \mathsf{senc}(s,k))\}; \{w_0 \mapsto k\})$ *for some* $k, s \in \mathcal{N}$ *does not keep* $s$ *secrect.*

**Authentication.** Many security protocols aim at authenticating one agent to another: one agent should become sure of the identity of the other. As already said in the general introduction, there are several variants of authentication and a taxonomy of these has been proposed by Lowe

in [Low97]. Let us informally describe such a property. For a configuration $K$ made of two agents modelled by processes $P$ and $Q$ communicating respectively on channel $c_P$ and $c_Q$, we say that $K$ guarantees to $P$ a *weak agreement* with $Q$ if for any execution of $K$, if $P$ has finished (all actions on channel $c_p$ have been triggered) then $Q$ has finished (all actions on channel $c_q$ have been triggered). Sometimes, processes and the semantics are equipped with additional annotations in order to define other aspects of authentication (*e.g.* $P$ and $Q$ really exchanged messages, $P$ and $Q$ agreed on some value).

**Example 14.** *For example, the configuration* $(\{P_{\mathsf{Fh}}\};\emptyset)$ *from Example 7 guarantees to* $P_I$ *a weak agreement with* $P_R$*. Establishing this property is not easy but can be automated using* e.g. ProVerif*.*

## 3.2 Behavioural Equivalences

Reachability properties are useful to model many security goals. However, privacy goals such as anonymity or unlinkability cannot be defined (or cannot be naturally defined) as reachability predicates. They are rather defined relying on a notion of *behavioural equivalence* relating configurations that are *indistinguishable*.

**Discussion.** Intuitively, two configurations are indistinguishable if an attacker has no way to tell them apart even when he is actively trying to do so. A natural starting point is to say that configurations $A$ and $B$ are indistinguishable if they can output on the same channels, no matter the context in which they are placed. Such a notion (called *may-testing equivalence* [CCD13a]) thus represents the attacker as a context and would represent behaviours of the protocol in presence of this attacker through a reduction semantics (that would not produce labels). Yet, the quantification over contexts makes this definition hard to use in practice. Therefore, indistinguishability notions based on a LTS (as the one presented in Chapter 2) have been proposed. For such LTS's, executions model all behaviours of the protocol that may interact with an environment that represents an arbitrary attacker. The attacker's behaviour is described by the trace associated to the execution. Intuitively, the indistinguishability notion can now be expressed as the fact that two configurations produce the same set of traces and frames (we define below *trace equivalence* that formalises this notion). Such a definition is more suitable for both manual and automatic reasoning since it avoids the quantification over contexts required when using a reduction semantics.

Actually linking these two semantics (LTS vs. reduction semantics) and their associated notions of equivalence (trace equivalence vs. may-testing) is not an easy task. Starting with the pioneering work of Milner and Sangiorgi [MS92], this problem has been addressed for different calculi and different notions of equivalences in several works (*e.g.* $\pi$-calculus, spi-calculus [AG98, MNP02], applied-pi calculus [AF01], and psi-calculus [BJPV11]). Usually, the two notions of equivalence coincide but that is not the case for the applied-pi calculus. Indeed, it has been proved [CCD13a] that trace equivalence always implies may-testing equivalence and that may-testing equivalence implies trace equivalence only for image-finite configurations; *i.e.* for any

trace, there are only finitely many resulting frames modulo an indistinguishability notion over frames (*i.e.* static equivalence defined below). Note that, all configurations according to our definitions are necessarily image-finite since our model does not feature internal communication (*i.e.* communication that does not produce any visible action for the attacker). In our setting, *trace equivalence* is thus the most appropriate notion.

**Static Equivalence.** Before formally defining *trace equivalence*, we first introduce a notion of equivalence between frames, called *static equivalence*.

**Definition 8** (Static equivalence)**.** *A frame* $\Phi$ *is* statically included *in* $\Psi$ *when* $\mathrm{dom}(\Phi) = \mathrm{dom}(\Psi)$*, and*

- *for any recipe* $R$ *such that* $R\Phi \Downarrow u$ *for some message* $u$*, there exists some message* $u'$ *such that* $R\Psi \Downarrow u'$*;*
- *for any recipes* $R_1, R_2$ *such that* $R_1\Phi \Downarrow u_1$ *for some message* $u_1$ *and* $R_2\Phi \Downarrow u_2$ *for some message* $u_2 =_{\mathsf{E}} u_1$*, there exist* $v_1, v_2$ *such that* $R_1\Psi \Downarrow v_1$*,* $R_2\Psi \Downarrow v_2$*, and* $v_1 =_{\mathsf{E}} v_2$*.*

*Two frames* $\Phi$ *and* $\Psi$ *are in* static equivalence*, written* $\Phi \sim \Psi$*, if the two static inclusions hold.*

Intuitively, an attacker can distinguish two frames if he is able to perform some computation or a test of equality that succeeds in $\Phi$ and fails in $\Psi$ (or the converse).

**Example 15.** *Consider the following frame (given in Example 9):*

$$\phi_0 = \{w_1 \mapsto n'_I,\ w_2 \mapsto \mathsf{senc}(\langle n'_I, n'_R\rangle, k'),\ w_3 \mapsto \mathsf{senc}(\langle n'_R, n'_I\rangle, k')\}.$$

*We have that* $\phi_0 \sqcup \{w_4 \mapsto k'\} \not\sim \phi_0 \sqcup \{w_4 \mapsto k''\}$*. Indeed, the attacker may observe that the computation* $R = \mathsf{sdec}(w_2, w_4)$ *succeeds in* $\phi \sqcup \{w_4 \mapsto k'\}$ *but fails in* $\phi \sqcup \{w_4 \mapsto k''\}$*.*

### 3.2.1 Trace Equivalence

Then, *trace equivalence* is the active counterpart of static equivalence taking into account the fact that the attacker may interfere during the execution in order to distinguish between the two configurations. However, we require from the attacker to behave similarly for the two configurations. In other words, the attacker can choose an experiment represented by a trace of observable actions and then, he has to distinguish the frames resulting from the two configurations for this trace. For this, we define for a trace $\mathsf{tr}$ the trace $\mathsf{obs}(\mathsf{tr})$ to be the subsequence of $\mathsf{tr}$ obtained by erasing all unobservable actions; *i.e.* the $\tau$ actions: $\tau_{\text{ш}}, \tau_\nu, \tau_!, \tau_{\mathsf{then}}, \tau_{\mathsf{else}}, \tau_r$.

**Definition 9** (Trace equivalence)**.** *Let* $K_1$ *and* $K_2$ *be two configurations. We say that* $K_1 \sqsubseteq K_2$ *when, for any* $K_1 \xrightarrow{\mathsf{tr}_1} K'_1$ *such that* $bc(\mathsf{tr}_1) \cap fc(K_2) = \emptyset$*, there exists* $K_2 \xrightarrow{\mathsf{tr}_2} K'_2$ *such that* $\mathsf{obs}(\mathsf{tr}_1) = \mathsf{obs}(\mathsf{tr}_2)$ *and* $\Phi(K'_1) \sim \Phi(K'_2)$*. They are* trace equivalent*, written* $K_1 \approx K_2$*, when* $K_1 \sqsubseteq K_2$ *and* $K_2 \sqsubseteq K_1$*.*

**Example 16** (Continuing Example 7)**.** *Consider the process* $Q = \nu n_I.P_I \mid \nu n_R.P_R$ *modelling two parties of identity* $k$ *ready to execute one session of the Feldhofer protocol. We may be interested in checking whether* $K = (\{\Pi\{\nu k.Q, \nu k.Q\}^{\#}\}; \emptyset)$ *and* $K' = (\{\nu k.\Pi\{Q, Q\}^{\#}\}; \emptyset)$ *are in trace equivalence. Intuitively, this equivalence models the fact that two sessions of* $P_{\mathsf{Fh}}$ *are unlinkable:* i.e. *two sessions of the protocol performed by the same tag and reader (*i.e. *having the same symmetric key* $k$*) modelled by* $K'$ *appear to an attacker as if they have been initiated by two different tags (*i.e. *having two different fresh keys) modelled by* $K$*. This equivalence actually holds. It is non-trivial to establish though. Moreover, we will discuss in Section 3.3 a stronger notion of unlinkability taking into account an unbounded number of sessions of the protocol.*

### 3.2.2 Other Behavioural Equivalences

Showing trace equivalence properties is a very difficult task notably because of its forall-exists structure: for any execution of one configuration, one has to find an indistinguishable execution of the other configuration. For this reason, other notions of equivalence are sometimes considered in order to under-approximate trace equivalence: they consider strictly less pairs of configurations as equivalent.

**Labelled bisimilarity.** In the security setting, the notion of labelled bisimilarity has first been introduced to approximate trace equivalence [AG97] for the spi-calculus. We only informally define this notion here since we do not use it in later developments. The labelled bisimilarity is defined as the largest symmetric relation on configurations that: (i) relate only statically equivalent configurations, and, (ii) that relate configurations which can execute the same observable actions and such that their continuations remain related. The fact that labelled bisimilarity is based on a notion of step-by-step simulation between configurations makes this notion sometimes easier to establish directly.

It is well-known that labelled bisimilarity implies trace equivalence whereas the converse is false in general. However, it has been proved in [CCD13a] that these two notions coincide for a large class of configurations that includes in particular the class of *simple processes* that essentially contains linear processes (only input, conditional and output) in parallel playing each on pairwise distinct channels (we formally define this class later on in Chapter 6).

**Diff-equivalence.** Another notion of equivalence that has been extensively used in verification tools is the notion of *diff-equivalence*. Such a notion is defined on bi-configurations that are pairs of configurations that have the same structure and differ only in the choice of terms they use. The syntax is similar to the one introduced in Chapter 2 but each term $u$ has to be replaced by a bi-term written $\mathsf{choice}[u_1, u_2]$ (using ProVerif syntax). Given a bi-process $P$, the process $\mathrm{fst}(P)$ is obtained by replacing all occurrences of $\mathsf{choice}[u_1, u_2]$ with $u_1$. Similarly, $\mathrm{snd}(P)$ is obtained by replacing $\mathsf{choice}[u_1, u_2]$ with $u_2$. These notations are also used for bi-configurations and bi-frames.

$$\begin{array}{lll}
\text{Then} & (\{\mathsf{let}\ \mathsf{choice}[\overline{x_1},\overline{x_2}] = \mathsf{choice}[\overline{t_1},\overline{t_2}]\ \mathtt{in}\ P\ \mathsf{else}\ Q\} \uplus \mathcal{P};\Phi) & \\
& \xrightarrow{\tau_{\mathsf{then}}}_{\mathsf{bi}} (\{P\{\mathsf{choice}[\overline{x_1},\overline{x_2}] \mapsto \mathsf{choice}[\overline{u_1},\overline{u_2}]\} \uplus \mathcal{P};\Phi) & \text{when } \overline{t_1} \Downarrow \overline{u_1} \text{ and } \overline{t_2} \Downarrow \overline{u_2} \\
\text{Else} & (\{\mathsf{let}\ \mathsf{choice}[\overline{x_1},\overline{x_2}] = \mathsf{choice}[\overline{t_1},\overline{t_2}]\ \mathtt{in}\ P\ \mathsf{else}\ Q\} \uplus \mathcal{P};\Phi) & \\
& \xrightarrow{\tau_{\mathsf{else}}}_{\mathsf{bi}} (\{Q \uplus \mathcal{P};\Phi) & \text{when } \overline{t_1} \not\Downarrow \text{ and } \overline{t_2} \not\Downarrow
\end{array}$$

**Figure 3.1** Semantical rules of biprocesses for conditional

The semantics of bi-configurations is defined as expected via a relation that expresses when and how a bi-configuration may evolve. A bi-process reduces if, and only if, both sides of the bi-process reduce in the same way; *i.e.* exactly the same semantical rule (from Figure 2.2) should be applied on both sides. For instance a conditional has to be evaluated in the same way on both sides as shown by the rules in Figure 3.1. When the two sides of the bi-process reduce in different ways, the bi-process blocks. This leads us to the following notion of diff-equivalence.

**Definition 10.** *An initial bi-configuration $K_0$ satisfies* diff-equivalence *if for every bi-configuration $K = (\mathcal{P};\phi)$ such that $K_0 \xrightarrow{\mathsf{tr}}_{\mathsf{bi}} K$ for some trace $\mathsf{tr}$, we have that:*

- $\mathrm{fst}(\phi) \sim \mathrm{snd}(\phi)$*;*
- *if* $\mathrm{fst}(K) \xrightarrow{\alpha} K'_L$ *then there exists a bi-configuration $K'$ such that $K \xrightarrow{\alpha}_{\mathsf{bi}} K'$ and* $\mathrm{fst}(K') = K'_L$ *(and similarly for* snd*).*

As expected, this notion of diff-equivalence is actually stronger than the usual notion of labelled bisimilarity, and thus trace equivalence. Indeed, it may be the case that the two sides of the bi-process reduce in different ways (e.g. taking two different branches in a conditional) but still produce the same observable actions. This strong notion of diff-equivalence happens to be sufficient to establish some interesting equivalence-based properties such as strong secrecy. It can be automatically verified using the tools ProVerif, Tamarin and Maude−NPA. However, as already discussed in the general introduction, this notion is actually too strong to establish for example vote privacy for many interesting e-voting protocols [DKR08], or unlinkability as defined in [ACRR10]. We come back to this problem in Subsection 3.3.1 and address it for a rich class of protocols in Part C.

## 3.3 Examples of Privacy Goals Modelling

We now give two examples of privacy goals defined using trace equivalence: unlinkability property on the Feldhofer protocol in Subsection 3.3.1 and anonymity of the initiator role of the Private Authentication protocol in Subsection 3.3.2. Only illustrative examples are given in this section, showing typical uses of trace equivalence. Formal definitions of different variants of unlinkability and anonymity for unbounded number of sessions and users will be given in Part C (Chapter 8).

### 3.3.1 Unlinkability of Feldhofer

We informally explain how to model unlinkability of the Feldhofer protocol for an unbounded number of sessions and users. Note that there are other definitions of unlinkability corresponding to different modellings, scenarios or threat models. We will come back to those definitions in Chapter 8.

We iterate from Example 16 where a definition of unlinkability for two sessions was given and now seek for a definition of unlinkability considering an unbounded number of users and sessions. In Example 16, unlinkability was defined by comparing a "real scenario" where two sessions were performed by the same tag and reader with an "ideal scenario" where two sessions are performed by two different pairs of tags and readers. For the unbounded case, the "real scenario" models now an unbounded number of pairs of tags and readers playing each an unbounded number of sessions while the "ideal scenario" models an unbounded number of pairs of tags and readers playing each at most one session. The former and the latter are respectively modelled by $K_r = (\{!\nu k!Q\}; \emptyset)$ and $K_i = (\{!\nu k.Q\}; \emptyset)$. The trace equivalence between $K_r$ and $K_i$ models the fact that $P_{\mathsf{Fh}}$ is unlinkable for an unbounded number of users and sessions: each session of the protocol appears to an attacker as if it has been initiated by a different tag, since a given tag can perform at most one session in the idealised scenario $K_i$. This equivalence actually holds.

The trace equivalence between $K_r$ and $K_i$ is non-trivial to establish. Actually, no tool is able to verify it directly. Indeed, the only existing tools that can verify a notion of behavioural equivalence for unbounded sessions (*i.e.* ProVerif, Tamarin or Maude−NPA) all verify diff-equivalence rather than trace equivalence. But the above equivalence cannot be established via diff-equivalence because the underlying bi-process (*i.e.* $(\{!\nu k_l!\nu k_r.\mathsf{let}\ k = \mathsf{choice}[k_l, k_r]\ \mathtt{in}\ Q\}; \emptyset)$) is not diff-equivalent while $K_i$ and $K_r$ are trace equivalent. Unfortunately, this is not an isolated problem: unlinkability as defined above is often incompatible with diff-equivalence because the two processes to be verified do not share the same structure. To put in other words: diff-equivalence considers an over-approximation of the Dolev-Yao attacker by giving him the internal structure of the two processes to be verified, the attacker is thus able to observe from which replication a given thread originates often breaking unlinkability by construction. We detail at greater length the reasons explaining the above in the introduction of Part C.

In order to address this problem, we devise sufficient conditions that imply unlinkability and anonymity and that can be verified using such existing tools without the aforementioned precision issue. We describe those conditions and the overall methodology in Part C.

### 3.3.2 Anonymity of the Private Authentication Protocol

We consider the Private Authentication protocol given in [AF04] designed for authenticating an agent with another one without revealing their identities to other participants.

**Private Authentication Protocol.** In this protocol, $A$ is willing to engage in communication with $B$ and wants to be sure that she is indeed talking to $B$ and not to an attacker who is trying to impersonate $B$. However, $A$ wants to protect her privacy and expects that her identity is not

disclosed except to $B$. The protocol specifies the following interactions between participants $A$ and $B$ (in Alice & Bob notations):

$$\begin{array}{lcl} A \to B & : & \{N_a, \mathsf{pub}_A\}_{\mathsf{pub}_B} \\ B \to A & : & \{N_a, N_b, \mathsf{pub}_B\}_{\mathsf{pub}_A} \end{array}$$

First $A$ sends to $B$ a nonce $N_a$ and her public key (asymmetrically) encrypted with the public key of $B$. If the plaintext is of the expected form (*i.e.* the right-part of the pair contains the public key $\mathsf{pub}_A$) then $B$ sends to $A$ the nonce $N_a$, a freshly generated nonce $N_b$ and his public key, all of this being encrypted with the public key of $A$. Moreover, if the message received by $B$ is not of the expected form then $B$ sends out a "decoy" message: $\{N_b\}_{\mathsf{pub}_B}$. This message should basically be indistinguishable from $B$'s other message from the point of view of an outsider.

Relying on the signature and equational theory introduced in Example 12 (in Section 2.1), a session of role $A$ played by agent $a$ (with private key $ska$) with $b$ (with public key $pkb$) can be modelled as follows:

$$\begin{array}{lcl} P(ska, pkb) & \stackrel{\text{def}}{=} & \mathtt{out}(c_A, \mathsf{aenc}(\langle n_a, \mathsf{pk}(ska)\rangle, pkb)). \\ & & \mathtt{in}(c_A, x). \\ & & \mathsf{let}\ z = \mathsf{neq}(\langle \pi_1(\mathsf{adec}(x, ska)), \pi_2(\pi_2(\mathsf{adec}(x, ska)))\rangle, \langle n_a, pkb\rangle)\ \mathtt{in}\ 0\ \mathsf{else}\ 0 \end{array}$$

Here, we are only considering the authentication protocol. A more comprehensive model should include the access to an application in case of a success (*i.e.* replacing the null process in the then branch by a process modelling the continuation in case of success). Similarly, a session of role $B$ played by agent $b$ with $a$ can be modelled by the following process, where $u = \mathsf{adec}(y, skb)$.

$$\begin{array}{lcl} Q(skb, pka) & \stackrel{\text{def}}{=} & \mathtt{in}(c_B, y). \\ & & \mathtt{let}\ z = \mathsf{neq}(\pi_2(u), pka)\ \mathtt{in}\ \mathtt{out}(c_B, \mathsf{aenc}(\langle \pi_1(u), \langle n_b, \mathsf{pk}(skb)\rangle\rangle, pka)) \\ & & \qquad\qquad\qquad\qquad \mathtt{else}\ \mathtt{out}(c_B, \mathsf{aenc}(n_b, \mathsf{pk}(skb))) \end{array}$$

To model a scenario with one session of each role (played by the agents $a$ and $b$), we may consider the configuration $(\mathcal{P}; \Phi_0)$ where:

- $\mathcal{P} = \{P(ska, \mathsf{pk}(skb)), Q(skb, \mathsf{pk}(ska))\}$, and
- $\Phi_0 = \{w_0 \mapsto \mathsf{pk}(ska'),\ w_1 \mapsto \mathsf{pk}(ska),\ w_2 \mapsto \mathsf{pk}(skb)\}$.

The purpose of $\mathsf{pk}(ska')$ will be clear later on. It allows us to consider the existence of another agent $a'$ whose public key $\mathsf{pk}(ska')$ is known by the attacker.

**Anonymity.** Intuitively, the private authentication protocol preserves anonymity of the role $A$ for one session if an attacker cannot distinguish between the two following scenarios (provided $a$, $a'$ and $b$ are honest participants):

- $a$ and $b$ who is willing to talk to $a$ play a session of the protocol (represented by the processes $\mathcal{P}^a = \mathcal{P} = \{P(ska, \mathsf{pk}(skb)), Q(skb, \mathsf{pk}(ska))\}$) or

- $a'$ and $b$ who is willing to talk to $a'$ play a session of the protocol (represented by the processes $\mathcal{P}^{a'} = \{P(ska', \mathsf{pk}(skb)), Q(skb, \mathsf{pk}(ska'))\}$).

This can be expressed relying on the following equivalence:

$$(\mathcal{P}^a; \Phi_0) \stackrel{?}{\approx} (\mathcal{P}^{a'}; \Phi_0).$$

Note that we define anonymity more formally and for an unbounded number of sessions and users in Chapter 8.

For illustration purposes, we also consider a variant of the process $Q$, denoted $Q_0$, where its `else` branch has been replaced by 0 (*i.e.* the null process). We note $\mathcal{P}_0^a = \{P(ska, \mathsf{pk}(skb)), Q_0(skb, \mathsf{pk}(ska))\}$ and correspondingly for $\mathcal{P}_0^{a'}$. We will see that the "decoy" message plays a crucial role to ensure privacy. We have that:

$$(\mathcal{P}_0^a; \Phi_0) \xrightarrow{\mathtt{in}(c_B, \mathsf{aenc}(\langle w_1, w_1\rangle, w_2)).\tau_{\mathsf{then}}.\mathtt{out}(c_B, w_3)} (\{P(ska, \mathsf{pk}(skb)), 0\}; \Phi)$$

where $\Phi = \Phi_0 \uplus \{w_3 \mapsto \mathsf{aenc}(\langle \mathsf{pk}(ska), \langle n_b, \mathsf{pk}(skb)\rangle\rangle, \mathsf{pk}(ska))\}$. This trace has no counterpart involving the same observable actions in $(\mathcal{P}_0^{a'}; \Phi_0)$. Indeed, we have that:

$$(\mathcal{P}_0^{a'}; \Phi_0) \xrightarrow{\mathtt{in}(c_B, \mathsf{aenc}(\langle w_1, w_1\rangle, w_2)).\tau_{\mathsf{else}}} (\{P(ska', \mathsf{pk}(skb)), 0\}; \Phi_0).$$

Hence, it holds that $(\mathcal{P}_0^a; \Phi_0) \not\approx (\mathcal{P}_0^{a'}; \Phi_0)$.

However, it is the case that $(\mathcal{P}^a; \Phi_0) \approx (\mathcal{P}^{a'}; \Phi_0)$ meaning that the Private Authentication protocol with the decoy message ensures the anonymity of the role $A$ for one session. This equivalence can be checked using the tool Apte [APTa] in less than 0.1 second for a simple scenario as the one considered here, and that takes few minutes/days as soon as we want to consider 2/3 sessions of each role. As already mentioned in the introduction this is not an isolated problem. The fundamental reason behind this is the *state space explosion problem* stemming from the concurrent nature of the problem leading to too many interleavings to explore.

In Part B, we show how to address this problem through dedicated Partial Order Reduction Techniques.

*Chapter 4*

# Variations of the Semantics

In the different chapters of this thesis, we need different levels of precision on the processes' structure and on the unobservable actions (*e.g.* $\tau$-actions). For instance, in Chapter 5, our analysis heavily relies on the internal structure of processes, notably on the structure of parallel compositions. Moreover, evaluations of conditionals do not take part to our analysis while replication should be executed lazily (*i.e.* only if the unfolded process is then immediately executed). By contrast, in Part C, evaluations of conditionals have to produce (unobservable) actions and replications are not constrained. Furthermore, parallel compositions are unimportant as our analysis focuses on the underlying multiset of (non-parallel) processes.

In this chapter, we show that, thanks to the flexibility of our framework, one can refine the semantics by forcing a subset of non observable actions to be executed as soon as possible (*i.e.* in a greedy way) and others as late as possible (*i.e.* in a lazy way) without impacting the different notions of security properties. The two types of refinements are presented respectively in Section 4.1 and Section 4.2. The semantics one can obtain in such a way do not capture different threat models or yield different security notions but only correspond to different definition flavors of the same notion that are captured by the same unified, generic framework we defined in previous chapters. We formally prove the latter in Section 4.3.

## 4.1 Executing Unobservable Actions Greedily

There are two ways to define semantics executing unobservable actions greedily. First, one can achieve this goal by choosing appropriate relations $\mathscr{R}$ on which the internal reduction $\rightsquigarrow_{\mathscr{R}}$ is built. However, this cannot tackle the creation of names since, on the one hand, the New rule non-deterministically chooses a fresh name, while, on the other hand, the relation $\mathscr{R}$ must yield a *confluent* internal reduction. Hence the need for a dedicated notion covering the $\nu$ case.

### 4.1.1 Internal Reduction: Conditional, Parallel Composition and Blocked Output

We first define a generic relation $\mathscr{R}$ containing all reductions we might use in this thesis. Variations of the semantics can then be obtained using a sub-relation of this generic relation and the

resulting class of relations one may use will be called *conform*.

One of the reductions we need is the evaluation of conditionals. For instance if for some term $t$ there exists $u$ such that $t \Downarrow u$ then we would like to reduce $P_t = \mathsf{let}\ x = t\ \mathtt{in}\ P\ \mathsf{else}\ Q$ to $P\{x \mapsto u\}$. However, remind that the internal reduction must be confluent and that there may be more than one message $v$ satisfying $t \Downarrow v$. Fortunately, we know by Definition 2 that all such messages are in the same equivalence class of $=_{\mathsf{E}}$. In order to achieve convergence, we thus introduce an arbitrary fixed selection function $\mathsf{selec} : \mathcal{T}(\Sigma_c, \mathcal{N}) \mapsto \mathcal{T}(\Sigma_c, \mathcal{N})$ such that: (i) for any messages $u =_{\mathsf{E}} v$, it holds that $\mathsf{selec}(u) = \mathsf{selec}(v)$, and, (ii) for any message $u$, it holds that $\mathsf{selec}(u) =_{\mathsf{E}} u$.

**Definition 11.** *We first define 3 types of reductions $\mathscr{R}_{\text{test}}, \mathscr{R}_{\text{par}}, \mathscr{R}_{\text{out}}$ as the least relations satisfying the following:*

- *(evaluation of conditionals) for any process of the form $P_t = \mathsf{let}\ t = v\ \mathtt{in}\ P\ \mathsf{else}\ Q$,*
  - *if $t \Downarrow u$ for some $u$ then $\{P_t\} \uplus \mathcal{P}\ \mathscr{R}_{\text{test}}\ \{P\{x \mapsto \mathsf{selec}(u)\}\} \uplus \mathcal{P}$*
  - *if $t \not\Downarrow$ then $\{P_t\} \uplus \mathcal{P}\ \mathscr{R}_{\text{test}}\ \{Q\} \uplus \mathcal{P}$*
- *(breaking parallel composition and removal of $0$) for any multiset of process $\mathcal{S}$, then $\{\Pi\mathcal{S}\} \uplus \mathcal{P}\ \mathscr{R}_{\text{par}}\ \mathcal{S} \uplus \mathcal{P}$.*
- *(removal of blocked outputs) for any term $t$ and process $P = \mathtt{out}(c, t).P'$ such that $t \not\Downarrow$, then $\{P\} \uplus \mathcal{P}\ \mathscr{R}_{\text{out}}\ \mathcal{P}$.*

*Further, $\mathscr{R}_{\text{r}}$ is the reunion of relations defined above;* i.e. *$\mathscr{R}_{\text{r}} = \mathscr{R}_{\text{test}} \cup \mathscr{R}_{\text{par}} \cup \mathscr{R}_{\text{out}}$. A relation $\mathscr{R}$ is said to be* conform *when $\mathscr{R} \subseteq \mathscr{R}_{\text{r}}$ and $\mathscr{R}_{\text{par}} \cap \mathscr{R} \neq \emptyset$ implies $\mathscr{R}_{\text{par}} \subseteq \mathscr{R}$.*

The second condition on the class of conform relations is needed to achieve confluence of the induced $\rightsquigarrow_{\mathscr{R}}$. For instance, a relation $\mathscr{R}$ that would collapse parallel compositions of size two only would not be confluent (and thus not convergent).

**Proposition 3.** *For any conform relation $\mathscr{R}$, the internal reduction $\rightsquigarrow_{\mathscr{R}}$ is terminating and confluent.*

*Proof.* (Termination) For $\mathscr{R}_{\text{r}}$, the sum of the structural sizes of processes (*i.e.* number of nodes of the productions of the grammar defining processes) in the multiset strictly decreases. This is thus also the case for $\mathscr{R}$. Further, this also applies to $\rightsquigarrow_{\mathscr{R}}$. Note that in the case of the collapse of parallel compositions, the modified process has one less parallel composition $\Pi$ so its size decreases by 1.

(Confluence) Since $\rightsquigarrow_{\mathscr{R}}$ has been proved terminating, it suffices to prove its local confluence to obtain its confluence (by Newman's lemma). It is easy to see that $\mathscr{R}_{\text{test}}$ and $\mathscr{R}_{\text{out}}$ cannot break the local confluence since those reductions (i) modify processes that cannot be modified by other rules and (ii) modify only one process independently from the rest of the processes in the multiset and thus do not induce any critical pair. Moreover, critical pairs involving collapse of parallel compositions or removal of parallel compositions of single processes can easily be

shown convergent. Therefore, one can deduce the local confluence of $\rightsquigarrow_{\mathscr{R}}$ when $\mathscr{R}_{\text{par}} \cap \mathscr{R} = \emptyset$. Otherwise, it holds that $\mathscr{R}_{\text{par}}$ is entirely included in $\mathscr{R}$. In that case, it suffices to remark that $\rightsquigarrow_{\mathscr{R}}$ always removes top-level parallel compositions by merging underneath processes into the multiset of processes. □

**Example 17.** *Consider $\mathscr{R}_{\text{test}}$ (that is conform). The semantics induced by $\rightsquigarrow_{\mathscr{R}_{\text{test}}}$ evaluates conditionals as soon as possible, even under parallel composition, without producing any action. For instance, we would have* $\{\Pi\{\mathsf{out}(d,\mathsf{ok}), \mathsf{let}\ \mathsf{eq}(\mathsf{ok},\mathsf{no})\ \mathtt{in}\ \mathsf{out}(c,\mathsf{ok})\ \mathsf{else}\ 0\}\} \rightsquigarrow_{\mathscr{R}_{\text{test}}} \{\mathsf{out}(d,\mathsf{ok})\}$.

### 4.1.2 $\nu$-greedy Executions: Creation of Names

As argued in the beginning of the chapter, a dedicated notion is needed to execute name creations greedily. The following definition essentially states that as soon as a process in the current configuration is ready to create a name, it is actually executed and before all other types of actions.

**Definition 12.** *We say that an execution $G \xrightarrow{\mathsf{tr}} H = (\mathcal{Q};\Psi)$ is* $\nu$-greedy *if (i) $H$ does not contain any $\nu$-process (*i.e. *there is no process of the form $\nu\overline{n}.Q$ in $\mathcal{Q}$) and (ii) for any prefix of the former execution of the form $G \xrightarrow{\mathsf{tr}_0} (\{\nu\overline{n}.P\} \uplus \mathcal{P};\Phi) \xrightarrow{\alpha} K$, the action $\alpha$ is $\tau_\nu$.*

Remark that we could have defined similar notions for conditionals and parallel compositions but we prefer to use internal reduction to deal with those cases since the latter allows to avoid producing spurious unobservable actions that will not get in the way of our analysis. Moreover, using internal reductions allows to simplify reasoning on configurations since, then, normal forms do not feature some constructs.

**Example 18.** *Consider the configuration $K = (\Pi\{P,P\}^{\#};\emptyset)$ where $P = \nu n.\mathsf{out}(c,n)$. The execution $K \xrightarrow{\tau_\Pi} (\{P,P\}^{\#};\emptyset) = K_1$ is not $\nu$-greedy and nor is $K \xrightarrow{\tau_\Pi.\tau_\nu.\mathsf{out}(c,w)} (\{P,0\}^{\#};\{w \mapsto n\}) = K_2$. Indeed, the resulting configurations $K_1$ and $K_2$ contain some $\nu$-processes. However, the following execution is $\nu$-greedy:*

$$K \xrightarrow{\tau_\Pi.\tau_\nu.\tau_\nu.\mathsf{out}(c,w).\mathsf{out}(c,w')} (\{0,0\}^{\#};\{w \mapsto n, w' \mapsto n'\}).$$

## 4.2 Executing Unobservable Actions Lazily

In this thesis, we are only interested in replication to be lazily executed and thus limit our definitions to this case but note that we could have dealt similarly with recursion, conditionals, creation of names and parallel compositions.

**Definition 13.** *We say that an execution $K \xrightarrow{\mathsf{tr}} K'$ is* !-lazy *if for any of its prefix of the form $K \xrightarrow{\mathsf{tr}_0} (\{!P\} \uplus \mathcal{P};\Phi) \xrightarrow{\tau_!} (\mathcal{S} \uplus \{!P\} \uplus \mathcal{P};\Phi)$ where $\mathcal{S}$ is the normal form (w.r.t. the internal reduction) of $P$, one of the processes in $\mathcal{S}$ is executed (producing at least one action) in the execution under consideration immediately after this prefix.*

**Example 19.** *Consider the configuration* $K = (\{P\};\emptyset)$ *where* $P = !\mathtt{out}(c, ok)$. *The execution* $K \xrightarrow{\nu_1} (\{\mathtt{out}(c, ok), P\};\emptyset)$ *is not* !*-lazy and nor is*

$$K \xrightarrow{\nu_1.\nu_1.\mathtt{out}(c,w).\mathtt{out}(c,w')} (\{0, 0, P\}^{\#};\{w \mapsto \mathsf{ok}, w' \mapsto \mathsf{ok}\}).$$

*Indeed, the resulting replicated processes are not immediately executed after their replication. However,* $K \xrightarrow{\nu_1.\mathtt{out}(c,w)} (\{0, P\}^{\#};\{w \mapsto \mathsf{ok}\})$ *or* $K \xrightarrow{\nu_1.\mathtt{out}(c,w).\nu_1.\mathtt{out}(c,w')} (\{0, 0, P\}^{\#};\{w \mapsto \mathsf{ok}, w' \mapsto \mathsf{ok}\})$ *are both* !*-lazy.*

## 4.3 Stability of the Security Notions

We now turn to the proofs that all possible variations one can derive using previous definitions capture essentially the same security notions. To put in other words, those choices actually do not impact security notions. To that end, we need to introduce some notations to distinguish the different notions of semantics and induced trace equivalence.

For a relation $\mathscr{R} \subseteq \mathscr{R}_{\mathrm{r}}$ over multisets of processes, we note $\xrightarrow{\bullet}_{\mathscr{R}}$ the semantics equipped with $\rightsquigarrow_{\mathscr{R}}$. Further, we note $\xrightarrow{\bullet}^{\nu}_{\mathscr{R}}$ the semantics exploring only $\nu$-greedy executions. Similarly, we consider $\xrightarrow{\bullet}^{!}_{\mathscr{R}}$ (!-lazy) and $\xrightarrow{\bullet}^{\nu,!}_{\mathscr{R}}$ (!-lazy and $\nu$-greedy). Finally, $\approx^{\alpha}_{\mathscr{R}}$ refers to the trace equivalence notion induced by the semantics $\xrightarrow{\bullet}^{\alpha}_{\mathscr{R}}$ (*i.e.* replace $\rightarrow$ by $\rightarrow^{\alpha}_{\mathscr{R}}$ in Definition 9).

**Reachability.** We first show that the choice of a conform internal reduction and the choice of executing $\nu$ greedily or/and ! lazily does not impact reachable states and observable actions necessary to reach them. This is formalised by the two following easy properties.

**Proposition 4.** *For any conform relation* $\mathscr{P}, \mathscr{R}$, *and for any execution* $A \xrightarrow{\mathsf{tr}}_{\mathscr{P}} B$ *there exists a trace* $\mathsf{tr}'$ *and a configuration* $B'$ *such that* $A \xrightarrow{\mathsf{tr}'}_{\mathscr{R}} B'$, $\Phi(B) = \Phi(B')$ *and* $\mathsf{obs}(\mathsf{tr}) = \mathsf{obs}(\mathsf{tr}')$.

*Proof.* It suffices to show the result for (i) $\mathscr{P} = \emptyset$ (which is conform) and an arbitrary conform $\mathscr{R}$ and (ii) for an arbitrary conform relation $\mathscr{P}$ and $\mathscr{R} = \emptyset$.

Both results can be established easily by remarking that all reductions of $\rightsquigarrow_{\mathscr{R}}$ (resp. $\rightsquigarrow_{\mathscr{P}}$) made available by a conform non-empty $\mathscr{R}$ (resp. $\mathscr{P}$) can be exactly mimicked by applying semantical rules producing unobservable actions. The only exception being the blocked outputs (*i.e.* processes of the form $\mathtt{out}(c, t).P$ such that $t \not\downarrow$) that may be removed by reduction rules when present in configurations. Fortunately, those outputs cannot be triggered (since $t \not\downarrow$) so removing them as soon as they appear in a given execution does not alter this execution. □

**Proposition 5.** *For any conform relation* $\mathscr{R}$, $\alpha \subseteq \{\nu, !\}$ *and execution* $A \xrightarrow{\mathsf{tr}}_{\mathscr{R}} G$ *there exists a trace* $\mathsf{tr}'$ *and a configuration* $K$ *such that* $A \xrightarrow{\mathsf{tr}'}{}^{\alpha}_{\mathscr{R}} K$, $\Phi(G) = \Phi(K)$ *and* $\mathsf{obs}(\mathsf{tr}) = \mathsf{obs}(\mathsf{tr}')$.

*Proof.* Firstly, in case $\nu \in \alpha$ and $G$ contains $\nu$-processes, it is easy to complete the former execution with all available $\tau_\nu$ actions until the resulting process does not contain any $\nu$-process any more. We can thus assume the former w.l.o.g.. Now, we prove the result by induction on the number of prefixes of $A \xrightarrow{\mathsf{tr}} G$ that violate one of the $\alpha$ condition(s). When there is no such prefix, the resulting execution can be played by $\xrightarrow{\bullet}^{\alpha}_{\mathscr{R}}$. Otherwise, consider a prefix $A \xrightarrow{\mathsf{tr}_0} B \xrightarrow{\beta} C$

of $A \xrightarrow{\mathsf{tr}} G$ of minimal length that violates at least one $\alpha$ condition(s). If $\beta$ breaks a $\nu$-greedy condition only then $\mathsf{tr}_0$ produces at least a process starting with a creation of names. In that case, the prefix is of the following form: $A \xrightarrow{\mathsf{tr}_0} (\{\nu\overline{n}.P\} \uplus \mathcal{P}; \Phi) \xrightarrow{\beta} C$ and $\beta$ is not $\tau_\nu$. If $\nu\overline{n}.P$ is never executed in the rest of the execution then it suffices to execute it (and all its next creation of names if any) by completing the execution by adding unobservable actions $\tau_\nu$. Otherwise, it suffices to move existing $\tau_\nu$ actions towards the end of this prefix. The resulting prefix no longer violates the condition and the extra actions we added do not introduce extra prefixes that violate $\alpha$ condition(s).

If $\beta$ breaks a !-lazy condition only then $\beta$ must follow a $\tau_!$ action and $\beta$ is not performed by the replicated processes. The prefix is thus the form: $A \xrightarrow{\mathsf{tr}'_0} (\{!P\} \uplus \mathcal{P}; \Phi) \xrightarrow{\tau_!} (\mathcal{S} \uplus \{!P\} \uplus \mathcal{P}; \Phi) \xrightarrow{\beta} C$ such that $\beta$ does not come from the execution of a process in $\mathcal{S}$. If process(es) in $\mathcal{S}$ are never executed, then one can remove the above $\tau_!$ action from the execution without impacting the executability, the observable part of the trace nor the resulting frame. By doing so, we remove one prefix violating a condition in $\alpha$. Otherwise, it suffices to move $\tau_!$ just before the first action performed by a process in $\mathcal{S}$. The new prefix it thus creates (replacing the old one) now satisfies the conditions in $\alpha$. Moreover, this does not create new prefixes that could violate one of the $\alpha$ condition(s).

If $\beta$ breaks both $\nu$-greedy and !-lazy condition then the prefix is of the form $A \xrightarrow{\mathsf{tr}'_0} (\{!P\} \uplus \mathcal{P}; \Phi) \xrightarrow{\tau_!} ((\mathcal{S} \uplus \{\nu\overline{n}.Q\}) \uplus \{!P\} \uplus \mathcal{P}; \Phi) \xrightarrow{\beta} C$ such that $\beta$ does not come from the execution of a process in $(\mathcal{S} \uplus \{\nu\overline{n}.Q\})$. Dealing with this case as we did for the !-lazy condition removes two prefixes violating $\alpha$ conditions and creates at most one prefix violating $\alpha$ condition (when the first following action on $(\mathcal{S} \uplus \{\nu\overline{n}.Q\})$ is not a $\tau_\nu$); hence strictly decreasing the number of prefixes violating $\alpha$ conditions. □

Therefore, reachability properties expressed as predicates over reachable configurations (and observable actions necessary to reach them) are not impacted by those choices. The two previous propositions are also key ingredients for the trace equivalence case we tackle next.

**Equivalence.** Now, we shall prove that the notion of trace equivalence neither depends on the $\nu$-greedy and !-lazy status of the semantics nor on $\mathscr{R}$ as long as it is conform.

**Proposition 6.** *For any conform $\mathscr{R}$, the trace equivalences induced by $\mathscr{R}$ with any combination of $\nu$-greedy and !-lazy constraints coincides with $\approx$ (*i.e. *for any $\alpha \subseteq \{\nu, !\}$, it holds that $\approx^{\alpha}_{\mathscr{R}} = \approx_{\emptyset}$)*

*Proof.* ($\approx^{\alpha}_{\mathscr{R}} \subseteq \approx_{\emptyset}$) Consider an execution $A \xrightarrow{\mathsf{tr}_A} A'$. Applying Proposition 4 and then Proposition 5, we obtain an execution $A \xrightarrow{\mathsf{tr}'_A}{}^{\alpha}_{\mathscr{R}} A''$ such that $\Phi(A'') = \Phi(A')$ and $\mathsf{obs}(\mathsf{tr}_A) = \mathsf{obs}(\mathsf{tr}'_A)$. By hypothesis, there must be some $B \xrightarrow{\mathsf{tr}'_B}{}^{\alpha}_{\mathscr{R}} B''$ such that $\Phi(A'') \sim \Phi(B'')$ and $\mathsf{obs}(\mathsf{tr}'_A) = \mathsf{obs}(\mathsf{tr}'_B)$. Trivially, this execution is also valid without any greedy or lazy constraints: it holds that $B \xrightarrow{\mathsf{tr}'_B}{}_{\mathscr{R}} B''$. Applying Proposition 4, we obtain an execution $B \xrightarrow{\mathsf{tr}_B} B'$ such that $\Phi(B'') = \Phi(B')$ and $\mathsf{obs}(\mathsf{tr}_B) = \mathsf{obs}(\mathsf{tr}'_B)$ concluding the proof (by symmetry of $\approx_{\emptyset}$).

($\approx^{\alpha}_{\mathscr{R}} \supseteq \approx_{\emptyset}$) Consider an execution $A \xrightarrow{\mathsf{tr}_A}{}^{\alpha}_{\mathscr{R}} A'$. This execution is obviously valid as well without any greedy or lazy constraint: it holds that $A \xrightarrow{\mathsf{tr}_A}{}_{\mathscr{R}} A'$. We now apply Proposition 4

and obtain an execution $A \xrightarrow{\mathsf{tr}'_A} A''$ such that $\Phi(A'') = \Phi(A')$ and $\mathsf{obs}(\mathsf{tr}_A) = \mathsf{obs}(\mathsf{tr}'_A)$. By hypothesis, there must be some $B \xrightarrow{\mathsf{tr}'_B} B''$ such that $\Phi(A'') \sim \Phi(B'')$ and $\mathsf{obs}(\mathsf{tr}'_A) = \mathsf{obs}(\mathsf{tr}'_B)$. Applying Proposition 4 and then Proposition 5, we obtain an execution $B \xrightarrow{\mathsf{tr}_B}{}^{\alpha}_{\mathscr{R}} B'$ such that $\Phi(B') = \Phi(B'')$ and $\mathsf{obs}(\mathsf{tr}_B) = \mathsf{obs}(\mathsf{tr}'_B)$ concluding the proof (by symmetry of $\approx^{\alpha}_{\mathscr{R}}$). □

In the different chapters of this thesis, depending on our needs, we may use one or the other such variant of the semantics. The results we eventually establish there thus concern the same semantics and the same model as shown by propositions 4 to 6.

# Part B

# Partial Order Reduction Techniques

**Significantly Improving Efficiency for the Bounded Case**

## Table of Contents of the Part

# Introduction

**Problem.** As already discussed in the general introduction, one approach to the automatic decision of trace equivalence for bounded number of sessions consists in using symbolic execution and dedicated constraint solving procedures. Unfortunately, the state-of-the-art procedures and tools for deciding equivalence have a limited practical impact because they scale very badly. Take for example the Private Authentication protocol: verifying anonymity for that protocol as explained in Subsection 3.3.2 (in Part A) can be carried out using the tool Apte [APTa] within few seconds for a simple scenario with only one session of each role whereas it takes few minutes/days as soon as we want to consider 2/3 sessions of each role. We later conduct comprehensive benchmarks (Chapter 6) showing the same limitations. This is thus not an isolated problem and greatly impacts negatively the usefulness of those methods and tools.

This is not surprising since those tools treat concurrency in a very naive way, exploring all possible (symbolic) interleavings of concurrent actions. As a consequence, those tools suffer from the so-called *state explosion problem* caused by the too many interleavings to consider.

**Our goal.** In standard model-checking approaches for concurrent systems, the interleaving problem is handled using partial order reduction (POR) techniques [Pel98]. In a nutshell, these techniques aim to effectively exploit the fact that the order of execution of two independent (parallel) actions is irrelevant when checking reachability.

In the general introduction (Subsection 1.5.1), we already have briefly explained why the extensive state-of-the-art of POR in model-checking cannot be efficiently leveraged for our setting. Note that we describe this state-of-the-art into much more details and argue much further in a dedicated related-work chapter (Chapter 7). We thus seek for new POR techniques that are dedicated to our demanding security framework. Additionally to the previously discussed acknowledged difficulties of combining POR with the security framework, we shall address two major challenges.

First, we shall devise POR techniques that can be used to verify not only reachability properties but also trace equivalence. Contrary to what happens for reachability-based properties, trace equivalence cannot be decided relying only on the reachable states. The sequence of actions that leads to this state plays a role. Hence, if a specific interleaving is identified as redundant because

it allows to reach states that are already reachable (from other interleavings) then it cannot be simply discarded (*i.e.* not explored at all). Indeed, this interleaving might be the only witness of inequivalence so discarding it might make us lose completeness w.r.t. trace equivalence.

Second, in order to improve existing verification tools for security protocols, one has to design POR techniques that integrate nicely with symbolic execution. In particular, we shall seek for reductions that are effective when messages remain unknown, but leverage information about messages when it is inferred by the constraint solver. This is necessary to precisely deal with infinite, structured data. In this task, we get some inspiration from Mödersheim *et* al. [MVB10], who design a partial order reduction technique that blends well with symbolic execution in the context of security protocols verification. However, we shall see that their key insight is not fully exploited, and yields only a quite limited partial order reduction. Moreover, they only consider reachability properties (like all previous work on POR for security protocol verification) while we seek an approach that is adequate for model-checking equivalence properties.

**Contributions.** We develop POR techniques for trace equivalence checking for a rich class of processes. We achieve this by refining interleaving semantics in two steps leveraging two techniques, eliminating more and more redundant interleavings and traces.

The first refinement, called *compression*, aims at discarding redundant interleavings that can be detected only by looking at the nature of actions. For instance, one may remark that when an output and an input are both available, performing the output first and then the input is sufficient for being able to explore all reachable states. Intuitively, we do not lose states by performing the output in priority since it boils down to give outputted messages to the attacker as soon as they are available. This idea (quite trivial for the reachability case) only relies on the nature of available actions in order to constrain the way the semantics is explored. In other words, it imposes a syntactically-based strategy of exploration. Our compression technique is based on several other ideas of this kind (*e.g.* inputs must be executed in a row, focus on replicated processes) yielding an highly constrained strategy of exploration. It does not rely on data analysis at all and can easily be used as a replacement for the usual semantics in verification algorithms.

The second refinement, called *reduction*, builds upon compression by taking data into account and achieves optimality in eliminating redundant traces. Intuitively, this technique discards traces when the underlying choices of recipes allow one to reorganise the trace into another trace leading to the same state and following an interleaving that has been already explored. We define a predicate that operationally detects when a trace can be reorganised as above and shall be discarded for this reason.

For each of our optimised semantics, we prove that the induced, optimised trace equivalence coincides with the regular trace equivalence for a rich class of processes called *action-deterministic*. Essentially, this class enforces that two processes in parallel must always use distinct channels. This common assumption in POR techniques [BK08] is also reasonable in the context of security protocols, where the attacker knows with whom he is communicating. We also show that reachability properties are preserved by our POR techniques without assumption on processes.

On the practical side, we show how to integrate our reductions into a symbolic framework consisting of a symbolic semantics and a constraint solving procedure. We also explain how we integrated our partial order reduction into the state-of-the art tool Apte [CCD11], prove the correctness of this integration, and provide experimental results showing that our theoretical results do translate to dramatic improvements.

We stress that our presentation is generic enough to be easily adapted for other tools provided that they are based on a forward symbolic exploration of traces combined with a constraint solving procedure. This is supported by our preliminary implementation of our techniques in Spec and the independent implementation of our compression technique in Akiss (see Section 6.5).

**Organisation of this part.** In Chapter 5, we focus on the theoretical aspects of our POR techniques. We define the *compressed* and the *reduced* semantics, which refine the regular semantics. We do so for a rich class of processes notably featuring replication, parallel composition and conditionals with else branches. For all of our optimised semantics, we first show that they preserve reachable states and then prove that, for *action-deterministic processes*, they induce the same trace equivalence.

In Chapter 6, we explain how to put those optimised semantics into practice by adapting them for the symbolic framework. We define symbolic variants of our compressed and reduced semantics for the class of *simple processes*, which is roughly a sub-class of action-deterministic

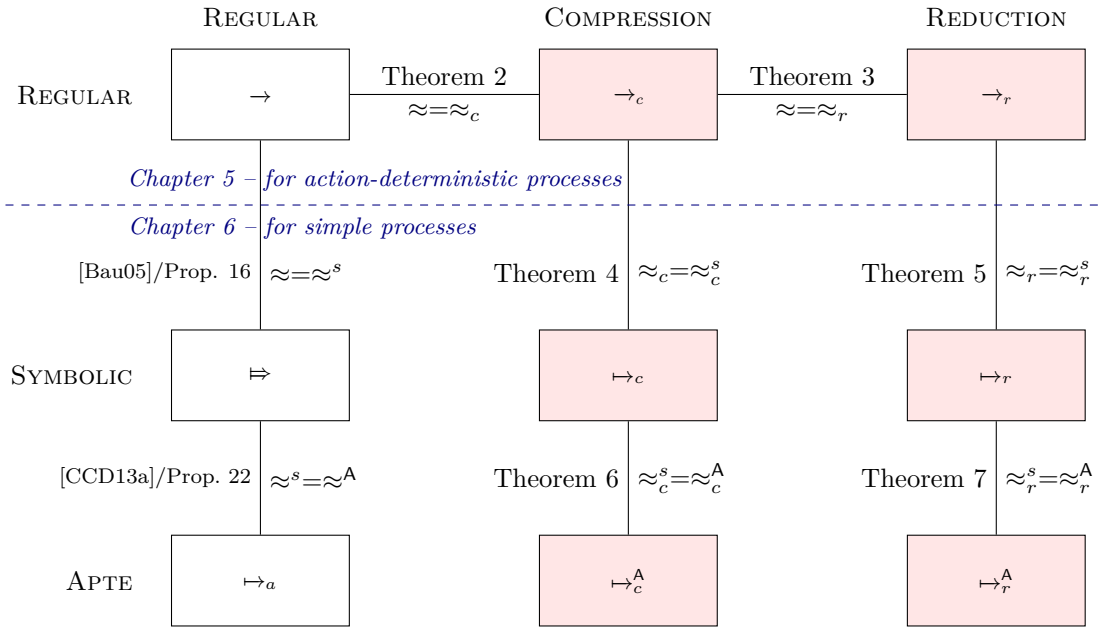


**Figure 4.1** Overview of chapters 5 and 6. Optimised semantics part of our contributions are colored in red. Vertically, it goes from the regular semantics (as defined in Chapter 2), to symbolic semantics and Apte's semantics. Those semantics have variants when our optimisations are applied or not: regular (*i.e.* no optimisation), compression or reduction (that includes compression). The dashed line separates contents of Chapter 5 from Chapter 6.

processes based on a simple syntactical criterion. We also define the semantics that the tool Apte is exploring and show how to adapt this tool to implement our reductions. Finally, we prove the soundness of this implementation and discuss conclusive benchmarks.

An overview of the different semantics we define in chapters 5 and 6 and the results relating them is depicted in Figure 4.1.

Finally, we conduct a comprehensive analysis of related works in Chapter 7.

*Chapter 5*

# A Reduced Semantics: Theory

In this chapter, we define our POR techniques by iteratively refining the regular semantics. In Section 5.1, we first define the instantiation of the model we shall work with and the class of processes we will deal with in this chapter. We notably define the *action-deterministic* class. We give in Section 5.2 an *annotated* semantics that will facilitate the next technical developments. We then define our *compressed* semantics in Section 5.3 and our *reduced* semantics in Section 5.4. In both sections, we first restrict the transition system, then show that the restriction is adequate for checking trace equivalence under the action-determinism condition. Our main result is given in Section 5.5 with some concluding remarks.

## 5.1 Instantiation of the Model and Class of Processes

The POR techniques we develop in this chapter are defined in a rather generic setting. We present in this section the instantiation of the semantics we shall work with and the class of configurations we deal with.

**Term Algebra.** We consider an arbitrary term algebra made of an arbitrary signature $\Sigma$, equational theory $=_{\mathsf{E}}$ and computation relation $\Downarrow$ as defined in Chapter 2, Section 2.1.

**Semantics.** We shall consider the internal reduction $\rightsquigarrow_{\mathscr{R}}$ induced by the relation $\mathscr{R} = \mathscr{R}_{\text{test}} \cup \mathscr{R}_{\text{out}}$ (see Definition 11 in Chapter 4). Remind that this internal reduction evaluates conditionals ($R_{\text{test}}$) and removes blocked outputs ($R_{\text{out}}$) greedily. We consider a !-lazy and $\nu$-greedy semantics based on this internal reduction (see definitions 12 and 13 in Chapter 4). The semantics thus executes creation of names actions as soon as possible and unfolds replications only if the unfolded process is then immediately executed. As already argued (see Proposition 6 in Chapter 4), this is not a restriction on the semantics or the threat model but a technical tool that facilitates the development.

| | | |
|---|---|---|
| Par | $(\{\Pi\,\mathcal{S}\} \uplus \mathcal{P}; \Phi) \xrightarrow{\tau_n} (\mathcal{P} \uplus \mathcal{S}; \Phi)$ | |
| In | $(\{\mathtt{in}(c,x).P\} \uplus \mathcal{P}; \Phi) \xrightarrow{\mathtt{in}(c,R)} (\{P\{x \mapsto u\}\} \uplus \mathcal{P}; \Phi)$ where $R$ is a recipe such that $R\Phi \Downarrow u$ | |
| Out | $(\{\mathtt{out}(c,t).P\} \uplus \mathcal{P}; \Phi) \xrightarrow{\mathtt{out}(c,w)} (\{P\} \uplus \mathcal{P}; \Phi \uplus \{w \mapsto u\})$ where $w$ is a fresh variable and $t \Downarrow u$ | |
| Repl | $(\{!^a_{\overline{c},\overline{n}} P\} \uplus \mathcal{P}; \Phi) \xrightarrow{\mathtt{sess}(a,\overline{c})} (\{P; !^a_{\overline{c},\overline{n}} P\} \uplus \mathcal{P}; \Phi)$ | $\overline{c}, \overline{n}$ fresh |

**Figure 5.1** Semantics for this chapter

**Class of Processes.** In this chapter, for parts of our results, we eventually impose an action-determinism condition which essentially requires that processes we are dealing with have always at most one way to execute a given observable action. This is *a priori* incompatible with replication (*i.e.* construct !) since it introduces an unbounded number of equal processes in parallel ready to execute the same action (on the same channel). However, action-determinism would not forbid a constrained form of replication where different sessions coming from the same replication use different public channels. For this reason, we introduce a new construct *as syntactic sugar* combining replication with channel and name restriction.

**Notation 1.** *For a process $P$, a non-empty sequence of channels $\overline{c}$ and sequence of names $\overline{n}$, we note $!^a_{\overline{c},\overline{n}} P$ the process $!\nu\overline{c}\,\mathtt{out}_{\mathrm{ch}}(a,\overline{c}).\nu\overline{n}.P$. We also note $\mathtt{sess}(a,\overline{c})$ the sequence of actions $\tau_!.\mathtt{outch}(a,\overline{c}).\tau_\nu$ that such a process may produce.*

Intuitively, such a process always advertises on the public channel $a$ any new copy of the replicated process. At the same time, it makes public the new channels $\overline{c}$ on which the copy may operate — but not the new names $\overline{n}$.

Moreover, we do not consider recursive processes in this chapter. There are fundamental reasons for that which will be made explicit in Section 5.5. Since our class forbids recursion and only considers the above constrained form of replication which bundles the generation of new (private) names as part of the construct, we can, without loss of expressiveness, forbid creation of names in other places. Indeed, we can always anticipate the generation of these names and pull them until reaching a replication or the top of the process (in that case, those names can be transformed into private constants).

**Hypothesis 1** (Class of Processes)**.** *In this chapter, we only consider the* class of processes *without recursion, creation of names, creation of channels and replication except ones in constructs $!^a_{\overline{c},\overline{n}} P$.*

Remind that we consider in this chapter the variation of the semantics executing replication lazily and creation of names greedily. We thus only consider executions where processes of the form $!^a_{\overline{c},\overline{n}} P \stackrel{\text{def}}{=} !\nu\overline{c}\,\mathtt{out}_{\mathrm{ch}}(a,\overline{c}).\nu\overline{n}.P$ are fully executed up to $P$, always producing at once a sequence of actions of the form $\tau_!.\mathtt{outch}(a,\overline{c}).\tau_\nu$ (noted $\mathtt{sess}(a,\overline{c})$). Hence, we are allowed to consider the new rule Repl defined in Figure 5.1, which can be seen as the concatenation of the original rules Repl,$\mathrm{Out}_{\mathrm{ch}}$,New, as a replacement of the original rule for replication. Finally, the only remaining

needed rules are Par, In and Out since configurations in normal form from the considered class do not feature conditional, creation of channels, recursion or creation of names. In the end, for the class of processes under consideration, the variation of the semantics we are dealing with in this chapter is depicted in Figure 5.1. We also remark that processes in the above class remain in the class throughout executions.

In order to lift our optimised semantics to trace equivalence, we will require configurations to be *action-deterministic*.

**Definition 14.** *A configuration $K$ is* action-deterministic *if whenever $K \xrightarrow{\mathsf{tr}} (\mathcal{P}; \Phi)$, and $P, Q$ are two processes in $\mathcal{P}$, we have that $P$ and $Q$ cannot perform an observable action of the same nature (*`in`*,* `out`*, or* `sess`*) on the same channel (*i.e. *if both actions are of same nature, their first argument has to differ).*

Note that action-deterministic processes remain action-deterministic throughout executions. The problem of deciding whether a configuration is action-deterministic or not is undecidable (it suffices to reduce from one of the numerous existing undecidability results *e.g.* [MSDL99] for secrecy). However, it is easy to under-approximate this class with syntactical criteria easily checkable. A simple such criterion is given in Section 6.1 (*i.e.* class of *simple processes*).

## 5.2 Annotated Semantics

Our goal in this section is to define an intermediate semantics that uses extra annotations on produced actions. We prove that those annotations can be used to transform executions by permuting some actions and are useful to ensure deep symmetries on executions when proving trace equivalence: we will show a strong symmetry lemma stating that the same swaps can be done for trace equivalent action-deterministic processes. Later on, we build our optimised semantics on this intermediate one and use its properties as stepping stones.

### 5.2.1 Annotations and Semantics

The annotated actions notably feature *labels* indicating from which concurrent processes and unfoldings of replication they originate.

**Definition 15** (Labels)**.** *A* label *(ranges over $\ell, \ell'$) is an element of* $(\mathbb{N} \cup \mathcal{C} \cup \{\boxdot\})^*$.

We shall label actions and processes. Intuitively, a label on a process represents the history of that process and reflects how it has been obtained from an initial process through the execution. We will see that $\boxdot$ is added to the current label of a process when it performs an input or an output. An integer is used to keep track of the position the current process has among all sub-processes when breaking a parallel composition. Finally, channels are used to refer to a specific unfolding when performing a replication.

A *labelled action* is written $[\alpha]^\ell$ where $\alpha$ is an action and $\ell$ is a label. Similarly, a *labelled process* is written $[P]^\ell$. One of the roles of the annotated semantics we define next is to add those labels to produced actions and processes throughout executions.

$$\text{In} \quad (\{[\mathtt{in}(c,x).Q]^{\ell}\} \uplus \mathcal{P}; \Phi) \xrightarrow{[\mathtt{in}(c,R)]^{\ell}}_{a} (\{[Q\{x \mapsto u\}]^{\ell \cdot \boxdot}\} \uplus \mathcal{P}; \Phi) \quad \text{where } R \text{ is a recipe such that } R\Phi \Downarrow u$$

$$\text{Out} \quad (\{[\mathtt{out}(c,t).Q]^{\ell}\} \uplus \mathcal{P}; \Phi) \xrightarrow{[\mathtt{out}(c,w)]^{\ell}}_{a} (\{[Q]^{\ell \cdot \boxdot}\} \uplus \mathcal{P}; \Phi \cup \{w \mapsto u\}) \quad \text{where } w \text{ is a fresh variable and } t \Downarrow u$$

$$\text{Repl} \quad (\{[!^{a}_{\overline{c},\overline{n}} P]^{\ell}\} \uplus \mathcal{P}; \Phi) \xrightarrow{[\mathtt{sess}(a,\overline{c})]^{\ell \cdot \overline{c}}}_{a} (\{[P]^{\ell \cdot \overline{c}}, [!^{a}_{\overline{c},\overline{n}} P_0]^{\ell}\} \uplus \mathcal{P}; \Phi) \qquad \overline{c}, \overline{n} \text{ fresh}$$

$$\text{Par} \quad (\{[\Pi\mathcal{S}]^{\ell}\} \uplus \mathcal{P}; \Phi) \xrightarrow{[\mathtt{par}(\sigma_1;\ldots;\sigma_n)]^{\ell}}_{a} (\{[P_1]^{\ell \cdot 1}, \ldots, [P_n]^{\ell \cdot n}\} \uplus \mathcal{P}; \Phi) \quad n > 1,\ \mathcal{S} = \{P_1, \ldots, P_n\}^{\#},\ \sigma_i = \mathsf{sk}(P_i) \text{ and } \sigma_1 \leq \ldots \leq \sigma_n$$

$$\text{Zero} \quad (\{[0]^{\ell}\} \uplus \mathcal{P}; \Phi) \xrightarrow{[\mathtt{zero}]^{\ell}}_{a} (\mathcal{P}; \Phi)$$

**Figure 5.2** Annotated semantics

**Example 20.** *We will see that the process $P$ resulting from the execution*

$$([\mathtt{in}(a,x).!^{a}_{(c_1,c_2),\emptyset}\, \Pi\{\mathtt{out}(c_1,\mathsf{ok}), \mathtt{out}(c_2,\mathsf{ok}).P\}]^{\ell_0}; \emptyset) \xrightarrow{\mathsf{tr}} (\{[P]^{\ell}, [!^{a}_{(c_1,c_2),\emptyset}\, \Pi\{\mathtt{out}(c_1,\mathsf{ok}), \mathtt{out}(c_2,\mathsf{ok}).P\}]^{\ell_0 \cdot \boxdot}\}; \Phi)$$

*where* $\mathsf{tr} = \mathtt{in}(a,\mathsf{ok}).\mathtt{sess}(a,(c_1,c_2)).\tau_{\text{n}}.\mathtt{out}(c_1,w_1).\mathtt{out}(c_2,w_2)$ *may be given a label* $\ell = \ell_0 \cdot \boxdot \cdot c_1 \cdot c_2 \cdot 2 \cdot \boxdot$.

When reasoning about trace equivalence between two configurations, it will be crucial to maintain a consistent labelling between configurations along the execution. In order to do so, we define *skeletons of observable actions*.

**Definition 16** (Skeletons)**.** Skeletons of observable actions *are of the form* $\mathtt{in}_c$, $\mathtt{out}_c$ *or* $!^{a}$ *where* $a, c \in \mathcal{C}$. *Any process in a configuration in normal form that is neither* 0 *nor a parallel composition induces a skeleton corresponding to its toplevel connective, and we denote it by* $\mathsf{sk}(P)$. *We consider an (arbitrary) total ordering over those skeletons, denoted* $<$ *with* $\leq$ *being its reflexive closure.*

**Example 21.** *For instance, one has* $\mathsf{sk}(\mathtt{in}(c,x).0) = \mathtt{in}_c$.

Next, we define in Figure 5.2 the *annotated semantics* $\rightarrow_a$ over configurations whose processes are labelled. In rule Par, note that $\mathsf{sk}(P_i)$ is well defined as $P_i$ cannot be 0 nor a parallel composition since it would not be a normal form for the internal reduction $\rightsquigarrow_{\mathscr{R}}$ otherwise (remind that the internal reduction collapses nested parallel compositions as specified in Section 2.2, Definition 3). Note that the annotated transition system does not restrict the executions of a process but simply annotates them with labels, and replaces $\tau$ actions by more descriptive actions (*e.g.* $\mathtt{par}(\bullet), \mathtt{zero}$). Further, remark that Zero is a special case of the parallel composition rule in the regular semantics for null processes (remind that $0 = \Pi\emptyset$) producing a distinguished action $\mathtt{zero}$. We distinguish this special case because the exploration strategies which we eventually define behave differently when executing a null process or a "true" parallel composition. Finally, note that the label produced by the Repl rule is different from the label of the process that is

executed. We will see that this is necessary to eventually be able to swap two unfoldings of the same replicated process.

Skeletons are notably used to annotate actions produced when disassembling a parallel composition or unfolding a replication making more explicit, through the produced action, what processes are thus made available. Essentially, the action produced when disassembling $\Pi\{P_i\}_{1\leq i\leq n}$ is $\mathtt{par}(\mathtt{sk}(P_1)\dots\mathtt{sk}(P_n))$ making explicit that we break a parallel composition and make all skeletons of $P_i$'s available. Besides, processes $P_i$ are ordered in increasing order (according to skeletons and $<$) in order to make the labelling consistent[1] when applied to two different (but equivalent) configurations.

We extend $\mathtt{obs}(\bullet)$ to annotated actions as follows: $\mathtt{obs}([\alpha]^\ell) = \alpha$. We thus make observable the actions $\mathtt{zero}$ and $\mathtt{par}(\bullet)$ that replaced the unobservable $\tau$-action $\tau_{\text{ш}}$. We will show that for action-deterministic configurations, those extra information given to the attacker does not modify the resulting notion of trace equivalence.

### 5.2.2 Action Dependencies

We now define how to extract dependencies from annotated traces, which will allow us to analyse concurrency in an execution without referring to configurations. We obtain sequential dependencies from labels, in a way that is similar, *e.g.* to the use of *causal relations* in CCS [DDNM90]. We also define *recipe dependencies* which are a sort of data dependencies reflecting our specific setting, where we consider an arbitrary attacker who may interact with the process, relying on (maybe several) previously outputted messages to derive input messages.

**Definition 17** (Sequential Dependency)**.** *Two labels $\ell, \ell'$ are* independent *if $\ell$ is not a prefix of $\ell'$ and vice versa. They are* dependent *otherwise.*

**Example 22.** *For instance, $\ell \cdot \boxdot \cdot c_1 \cdot c_2 \cdot 2 \cdot \boxdot$ is dependent with $\ell$ but is independent with $\ell \cdot \boxdot \cdot c_1 \cdot c_2 \cdot 1$ (were in parallel), $\ell \cdot \boxdot \cdot c_1' \cdot c_2' \cdot 2 \cdot \boxdot$ (come from different unfoldings).*

**Definition 18** (Dependency)**.** *We say that the labelled actions $\alpha$ and $\beta$ are* sequentially dependent *when their labels are dependent, and* recipe dependent *when $\{\alpha, \beta\} = \{[\mathtt{in}(c, M)]^\ell, [\mathtt{out}(c', w)]^{\ell'}\}$ with $w$ occurring in $M$. They are* dependent *when they are sequentially or recipe dependent. Otherwise, they are* independent*.*

Our goal with action dependency is to characterise actions that can be swapped; *i.e.* (i) actions that do not use data of each other and (ii) actions that are performed by processes in parallel or resulting from two unfoldings of the same replicated process. However, (ii) can only be achieved when the initial configuration is well-labelled as defined next (*e.g.* $(\{[P]^\ell, [P]^\ell\}; \Phi)$ would trivially contradicts (ii)).

**Definition 19.** *A configuration $(\mathcal{P}; \Phi)$ is* well labelled *if $\mathcal{P}$ is a multiset of labelled processes such that:*

[1] Note that we will see that action-determinism guarantees that such skeletons are pariwise distinct.

1. *for any two elements* $[P]^{\ell}, [Q]^{\ell'}$ *of* $\mathcal{P}$ *such that* $P$ *and* $Q$ *are not replicated processes (*i.e. *not of the form* $!^{a}_{\tilde{c},\tilde{n}}P'$*) then* $\ell$ *and* $\ell'$ *are independent and;*
2. *for any* $[!^{a}_{\tilde{c},\tilde{n}}P]^{\ell} \in \mathcal{P}$ *and process* $[Q]^{\ell'} \in \mathcal{P}$ *that is not a replicated process then* $\ell'$ *is not a prefix of* $\ell$*.*

Remark that any two processes in a well-labelled configuration always produce sequentially independent actions. Obviously, any unlabelled configuration may be well labelled. Further, it is easy to see that well labelling is preserved by $\rightarrow_a$ as shown next.

**Proposition 7.** *Well labelling is preserved by* $\rightarrow_a$*;* i.e. *if* $K$ *is well labelled and* $K \xrightarrow{\mathsf{tr}}_a K'$ *for some configuration* $K'$ *then* $K'$ *is well labelled.*

*Proof.* By induction on $\mathsf{tr}$, it suffices to prove that well labelling is preserved by the execution of a single action. We reason by case analysis on this action.

Consider a Par transition, represented below (where $\mathcal{S} = \{P_i\}^{\#}_{1\leq i\leq n}$ such that $\sigma_i = \sigma(P_i)$ are well ordered):

$$(\{[\{\Pi\mathcal{S}\}]^{\ell}\} \uplus \mathcal{Q}; \Phi) \xrightarrow{[\mathtt{par}(\sigma_1,\ldots,\sigma_n)]^{\ell}}_a (\{[P_1]^{\ell\cdot 1}, \ldots, [P_n]^{\ell\cdot n}\} \uplus \mathcal{Q}; \Phi)$$

We first check that labels of all pairs of non-replicated processes are independent. This is obviously the case between new labels $\ell \cdot i$. Let us now consider a label $\ell'$ from a non-replicated process in $\mathcal{Q}$ and show that it is independent from any $\ell \cdot i$. For the sake of the contradiction, if $\ell \cdot i$ is a prefix of $\ell'$ then $\ell$ is also a prefix of $\ell'$ which contradicts the well labelling of the initial configuration. Similarly, if $\ell'$ is a prefix of $\ell \cdot i$ then either $\ell' = \ell \cdot i$ or $\ell'$ is a prefix of $\ell$. In both cases, we infer a contradiction. We now verify the second requirement of well labelling. If a process $[P_i]^{\ell\cdot i}$ is a replicated process, we already have proven above that $\ell \cdot i$ is independent with the label of any non-replicated process in $\mathcal{Q}$ which thus cannot be a prefix of the former. Next, consider a replicated process $[P]^{\ell'}$ in $\mathcal{Q}$. We have that no label $\ell \cdot i$ is a prefix of $\ell'$. Indeed, since the initial configuration was well labelled and $\Pi\mathcal{S}$ is a non-replicated process, one has that $\ell$ is not a prefix of $\ell'$.

Next, consider a Repl transition, represented below:

$$(\{[!^{a}_{\tilde{c},\tilde{n}}P]^{\ell}\} \uplus \mathcal{Q}; \Phi) \xrightarrow{[\mathtt{sess}(a,\tilde{c})]^{\ell\cdot\tilde{c}}}_a (\{[P]^{\ell\cdot\tilde{c}}, [!^{a}_{\tilde{c},\tilde{n}}P]^{\ell}\} \uplus \mathcal{Q}; \Phi)$$

We first check that labels of all pairs of non-replicated processes are independent. Let us consider a label $\ell'$ from a non-replicated process in $\mathcal{Q}$ and show that it is independent from $\ell \cdot \tilde{c}$. For the sake of the contradiction, assume that $\ell'$ is a prefix of $\ell \cdot \tilde{c}$. Remark that since $\tilde{c}$ are fresh names and $\ell'$ was present in the initial configuration, it holds that $\ell' \neq \ell \cdot \tilde{c}$ implying that $\ell'$ is also a prefix of $\ell$. This contradicts the well labelling of the initial configuration since a replicated process labelled $\ell$ was present. Conversely, if $\ell \cdot \tilde{c}$ is a prefix of $\ell'$ then $\ell'$ contains the names $\tilde{c}$ which contradicts the freshness condition of $\tilde{c}$ since $\ell'$ was initially present. We now verify the second requirement of well labelling. If the $[P]^{\ell\cdot\tilde{c}}$ is itself a replicated process, we already have proven above that $\ell \cdot \tilde{c}$ is independent with the label of any non-replicated process in $\mathcal{Q}$ which

thus cannot be a prefix of the former. Next, consider a replicated process $[P]^{\ell'}$ in $\mathcal{Q}$. We have that the label $\ell \cdot \overline{c}$ cannot be a prefix of $\ell'$ by the freshness condition for $\overline{c}$ since $\ell'$ was present in the initial configuration.

The cases of all other transitions are similar to the case of a Par transition. □

Thus, we shall implicitly assume to be working with well labelled configurations in the rest of the chapter. Under this assumption, we obtain the following fundamental lemma.

**Lemma 1** (Permutation Lemma)**.** *Let $A$ be a (well labelled) configuration, $\alpha$ and $\beta$ be two independent labelled actions. We have $A \xrightarrow{\alpha.\beta}_a A'$ if, and only if, $A \xrightarrow{\beta.\alpha}_a A'$.*

*Proof.* By symmetry it is sufficient to show one implication. Assuming $\ell_1$ and $\ell_2$ to be independent, we consider a transition labelled $\alpha = [\alpha']^{\ell_1}$ followed by one labelled $\beta = [\beta']^{\ell_2}$. We distinguish two cases whether $\alpha$ and $\beta$ are two unfoldings of the same replicated process or not.

**(If $\alpha' = \mathtt{sess}(a, \overline{c_1})$ and $\beta' = \mathtt{sess}(a, \overline{c_2})$.)** The considered execution is thus of the form (for some annotated processes $P_\alpha, P_\beta$):

$$A = (\mathcal{Q} \uplus \{[!^a_{\overline{c},\overline{n}} P]^\ell\}; \Phi) \xrightarrow{[\alpha']^{\ell \cdot \overline{c_1}}}_a (\mathcal{Q} \uplus \{[!^a_{\overline{c},\overline{n}} P]^\ell, P_\alpha\}; \Phi) \xrightarrow{[\beta']^{\ell \cdot \overline{c_2}}}_a (\mathcal{Q} \uplus \{[!^a_{\overline{c},\overline{n}} P]^\ell, P_\alpha, P_\beta\}; \Phi)$$

The freshness condition implies that elements of $\overline{c_1}$ (resp. $\overline{n_1}$) do not occur in $A$ and elements of $\overline{c_2}$ (resp. $\overline{n_2}$) do not occur in the configuration in the middle. We thus deduce that $\overline{c_2}$ (resp. $\overline{n_2}$) do not occur in $A$ and executing $\beta$ from $A$ would not introduce channels from $\overline{c_1}$ or names from $\overline{n_1}$. Therefore, one can easily swap the two actions $\alpha$ and $\beta$ in the execution under scrutiny.

**(Otherwise.)** We now deal with other cases; *i.e.* $\alpha'$ and $\beta'$ are not two unfoldings of the same replicated process. Let us prove that $\ell_2$ was already present in $A$ by showing that the process resulting from the action $\alpha$ cannot produce the action $[\beta']^{\ell_2}$. For the sake of the contradiction, assume the contrary. If $\alpha'$ is an input or an output action then we necessarily have that $\ell_2 = \ell_1 \cdot \boxdot$ (the resulting process is non-replicated) or $\ell_2 = \ell_1 \cdot \boxdot \cdot \overline{c}$ (otherwise). Both cases contradict the independence of $\ell_1$ and $\ell_2$. If $\alpha'$ is a $\mathtt{par}(\bullet)$ action then the same contradiction can be deduced. If $\alpha' = \mathtt{zero}$ then no process results from $\alpha$. Finally, consider the case $\alpha' = \mathtt{sess}(a, \overline{c})$ implying $\ell_1 = \ell_0 \cdot \overline{c}$. If $\beta$ is executed by the unfolded process then $\ell_2 = \ell_1$ (when the unfolded process is a non-replicated) or $\ell_2 = \ell_1 \cdot \overline{d}$ (otherwise). Both cases contradict the independence of $\ell_1$ and $\ell_2$. Finally, if $\beta'$ is executed by the replicated process that performed the action $\alpha$ then $\beta' = \mathtt{sess}(a, \overline{d})$ which contradicts our hypothesis.

Therefore, $\ell_2$ must be present in the original configuration and our execution is of the following form, where we write $P_\alpha$ (resp. $P_\beta$) instead of $[P_\alpha]^{\ell_1}$ (resp. $[P_\beta]^{\ell_2}$):

$$A = (\mathcal{Q} \uplus \{P_\alpha, P_\beta\}; \Phi) \xrightarrow{[\alpha']^{\ell_1}}_a (\mathcal{Q} \uplus \mathcal{Q}_\alpha \uplus \{P_\beta\}; \Phi_\alpha) \xrightarrow{[\beta']^{\ell_2}}_a (\mathcal{Q} \uplus \mathcal{Q}_\alpha \uplus \mathcal{Q}_\beta; \Phi_\beta)$$

It remains to check that $\beta$ can be performed by $P_\beta$ in the original configuration, and that doing so would not prevent the $\alpha$ transition to happen next. The only thing that could prevent $\beta$ from being performed is that the frames $\Phi$ and $\Phi_\alpha$ may be different, in the case where $\alpha$ is an input. In that case, the recipe independence hypothesis guarantees that $\beta$ does not rely on the new handle introduced by $\alpha$ and can thus be played with only $\Phi$. Finally, performing $\alpha$ after $\beta$

is easy. We also perform corresponding internal reductions that were applied after $\beta$ and $\alpha$ on the corresponding resulting processes. We only detail the case where $\beta' = \mathtt{out}(c, w)$ and $\alpha'$ is an input of recipe $M$. In that case we have $\Phi_\alpha = \Phi$, $\Phi_\beta = \Phi_\alpha \uplus \{w \mapsto m\}$, and $M \in \mathcal{T}(\Sigma_{\mathsf{pub}}, \mathrm{dom}(\Phi))$. We observe that $M \in \mathcal{T}(\Sigma_{\mathsf{pub}}, \mathrm{dom}(\Phi_\beta))$ and we construct the execution:

$$A = (\mathcal{Q} \uplus \{[P_\alpha]^{\ell_1}, [P_\beta]^{\ell_2}\}; \Phi) \xrightarrow{[\beta']^{\ell_2}}_a (\mathcal{Q} \uplus \mathcal{Q}_\beta \uplus \{[P_\alpha]^{\ell_1}\}; \Phi_\beta) \xrightarrow{[\alpha']^{\ell_1}}_a (\mathcal{Q} \uplus \mathcal{Q}_\beta \uplus \mathcal{Q}_\alpha; \Phi_\beta).$$

□

### 5.2.3 Symmetries of Trace Equivalence

We will see that, when checking $A \approx B$ for action-deterministic configurations, it is sound to require that $B$ can perform all traces of $A$ in the annotated semantics (and the converse). In other words, labels and detailed unobservable actions $\mathtt{zero}$ and $\mathtt{par}(\sigma_1 \dots \sigma_n)$ can be made observable by the attacker without impacting trace equivalence. Obviously, this can only hold if $A$ and $B$ are (initially) labelled consistently. In order to express this, we extend $\mathsf{sk}(P)$ to parallel and zero processes: we let their skeletons be the associated action in the annotated semantics. Next, we define the labelled skeletons by $\mathsf{skl}([P]^\ell) = [\mathsf{sk}(P)]^\ell$. When checking for equivalence of $A$ and $B$, we shall assume that $\mathsf{skl}(A) = \mathsf{skl}(B)$, *i.e.* the configurations have the same multiset of labelled skeletons. This technical condition is not restrictive in practice: as a consequence of action-determinacy, we show below (see Corollary 1) that, after pre-executing non-observable $\tau_{\text{II}}$ actions of $A$ and $B$, $\mathsf{sk}(A) = \mathsf{sk}(B)$ is a necessary condition for $A \approx B$. Hence, once $\mathsf{sk}(A) = \mathsf{sk}(B)$ is known, one can label $A$ and $B$ arbitrarily but consistently such that $\mathsf{skl}(A) = \mathsf{skl}(B)$.

**Example 23.** *Let* $A = (\{[\mathtt{in}(a, x).(\mathtt{out}(b, m).P_1 \mid P_2)]^0\}; \Phi)$ *with* $P_1 = \mathtt{in}(c, y).0$ *and* $P_2 = \mathtt{in}(d, z).0$*, and let* $B$ *be the configuration obtained from* $A$ *by swapping* $P_1$ *and* $P_2$*. We have* $\mathsf{skl}(A) = \mathsf{skl}(B) = \{[\mathtt{in}_a]^0\}^\#$*. Consider the following trace:*

$$\mathsf{tr} = [\mathtt{in}(a, \mathtt{ok})]^0.[\mathtt{par}(\{\mathtt{out}_b; \mathtt{in}_d\})]^{0 \cdot \boxdot}.[\mathtt{out}(b, w)]^{0 \cdot \boxdot \cdot 1}.[\mathtt{in}(c, w)]^{0 \cdot \boxdot \cdot 1 \cdot \boxdot}.[\mathtt{in}(d, w)]^{0 \cdot \boxdot \cdot 2}$$

*Assuming* $\mathtt{out}_b < \mathtt{in}_c < \mathtt{in}_d$ *and* $\mathsf{ok} \in \Sigma$*, we have* $A \xrightarrow{\mathsf{tr}}_a A'$ *for some* $A'$*. However, there is no* $B'$ *such that* $B \xrightarrow{\mathsf{tr}}_a B'$*, for two reasons. First,* $B$ *cannot perform the second action since skeletons of sub-processes of its parallel composition are* $\{\mathtt{out}_b; \mathtt{in}_c\}$*. Second, even if we ignored that mismatch on an unobservable action,* $B$ *would not be able to perform the action* $\mathtt{in}(c, w)$ *with the right label. Such mismatches can actually be systematically used to show* $A \not\approx B$*, as shown next.*

**Lemma 2** (Strong Symmetry)**.** *Let* $A$ *and* $B$ *be two action-deterministic configurations such that* $A \approx B$ *and* $\mathsf{skl}(A) = \mathsf{skl}(B)$*. For any execution*

$$A \xrightarrow{[\alpha_1]^{\ell_1}}_a A_1 \xrightarrow{[\alpha_2]^{\ell_2}}_a \dots \xrightarrow{[\alpha_n]^{\ell_n}}_a A_n$$

*with* $bc(\alpha_1 \dots \alpha_n) \cap fc(B) = \emptyset$*, there exists an execution*

$$B \xrightarrow{[\alpha_1]^{\ell_1}}_a B_1 \xrightarrow{[\alpha_2]^{\ell_2}}_a \dots \xrightarrow{[\alpha_n]^{\ell_n}}_a B_n$$

*such that* $\Phi(A_i) \sim \Phi(B_i)$ *and* $\mathsf{skl}(A_i) = \mathsf{skl}(B_i)$ *for any* $1 \le i \le n$*.*

Before proving the previous fundamental lemma (see Subsection 5.2.4), we show that the induced trace equivalence coincides with the regular trace equivalence as stated below.

**Definition 20.** *Let $A$ and $B$ be two configurations. We have that $A \sqsubseteq_a B$ if, for every $A'$ such that $A \xrightarrow{\mathsf{tr}}_a A'$ with $bc(\mathsf{tr}) \cap fc(B) = \emptyset$, then there exists $B'$ such that $B \xrightarrow{\mathsf{tr}}_a B'$, and $\Phi(A') \sim \Phi(B')$. They are in* annotated trace equivalence*, denoted $A \approx_a B$, if $A \sqsubseteq_a B$ and $B \sqsubseteq_a A$.*

In contrast to the regular semantics, the annotated semantics requires that both $A$ and $B$ are able to execute the same traces including unobservable actions and annotations.

**Theorem 1.** *Let $A$ and $B$ be two action-deterministic configurations such that $\mathsf{skl}(A) = \mathsf{skl}(B)$, it holds that:*

$$A \approx B \text{ if, and only if, } A \approx_a B.$$

*Proof.* The direction $\Leftarrow$ is trivial since any annotated execution is also a regular execution. The direction $\Rightarrow$ is a direct corollary of Lemma 2. □

### 5.2.4 Proof of the Strong Symmetry Lemma

This section is dedicated to the proof of Lemma 2.

First, we define a notion describing the set of all possible types of actions (*i.e.* skeletons) a process may exhibit maybe after some unobservable actions.

**Definition 21.** *Given a process $P$, we define the set of its* enabled skeletons *as*

$$\mathsf{enable}(P) = \begin{cases} \{\mathsf{sk}(P)\} & \text{if } P \text{ starts with an observable action} \\ \cup_{P \in \mathcal{S}} \{\mathsf{sk}(P)\} & \text{otherwise and } P \text{ is thus of the form } P = \Pi\mathcal{S} \end{cases}$$

In particular, note that one has $\mathsf{enable}(0) = \mathsf{enable}(\Pi\emptyset) = \emptyset$. We may consider skeletons, labelled skeletons and enabled skeletons of a configuration by taking the multiset of the corresponding objects of all its processes.

Next, we make two trivial remarks formalised below.

**Proposition 8.** *For any configurations $A$, $A'$ and unobservable action $\alpha$, if $A \xrightarrow{\alpha}_a A'$ or $A \xrightarrow{\tau_\Pi} A'$ then $\mathsf{enable}(A) = \mathsf{enable}(A')$.*

**Proposition 9.** *Let $A$ be an action-deterministic configuration and $P$, $Q$ two of its processes. We have that $\mathsf{enable}(P) \cap \mathsf{enable}(Q) = \emptyset$.*

Further, we prove a key proposition essentially showing that an action-deterministic configuration may have different executions yielding the same sequence of observable actions but they all intuitively represent the same execution if one forgets about the precise structure of parallel compositions (captured by $\mathsf{enable}(\cdot)$).

**Proposition 10.** *Let $A$ be an action-deterministic configuration. If $A \xrightarrow{\mathsf{tr}_1} A_1$ and $A \xrightarrow{\mathsf{tr}_2} A_2$ for some traces $\mathsf{tr}_1, \mathsf{tr}_2$ such that $\mathsf{obs}(\mathsf{tr}_1) = \mathsf{obs}(\mathsf{tr}_2)$ then $\mathsf{enable}(A_1) = \mathsf{enable}(A_2)$ and $\Phi(A_1) = \Phi(A_2)$.*

*Proof.* We first prove a stronger result when the configurations $A_1$ and $A_2$ are *canonical*, *i.e.* only contain processes that are neither 0 nor a parallel composition. Actually, in such a case, we prove that $A_1 = A_2$.

To prove this intermediate result, we proceed by induction on $\mathsf{obs}(\mathsf{tr}_1)$. The base case is trivial. Let us show the inductive case. We assume that $\mathsf{tr}_1 = \mathsf{tr}_1^0.\alpha.\mathsf{tr}_1^-$ with $\alpha$ an observable action and $\mathsf{tr}_1^-$ containing only unobservable actions. Since $\mathsf{obs}(\mathsf{tr}_1) = \mathsf{obs}(\mathsf{tr}_2)$, we have that $\mathsf{tr}_2 = \mathsf{tr}_2^0.\alpha.\mathsf{tr}_2^-$ with $\mathsf{tr}_2^-$ containing only unobservable actions and $\mathsf{obs}(\mathsf{tr}_1^0) = \mathsf{obs}(\mathsf{tr}_2^0)$. Our given executions are thus of the form:

$$A \xrightarrow{\mathsf{tr}_1^0} A_1^0 \xrightarrow{\alpha.\mathsf{tr}_1^-} A_1 \text{ and } A \xrightarrow{\mathsf{tr}_2^0} A_2^0 \xrightarrow{\alpha.\mathsf{tr}_2^-} A_2$$

It may be the case that $A_1^0$ or $A_2^0$ are not canonical. The idea is to reorder some unobservable actions. More precisely, we perform all available unobservable actions of $A_1^0$ and $A_2^0$ before performing $\alpha$. By doing this, we do not change the observable actions of the different sub-traces and obtain

$$A \xrightarrow{\mathsf{tr}_1^0.\mathsf{tr}_3^-} A_1'^0 \xrightarrow{\alpha.\mathsf{tr}_1'^-} A_1 \text{ and } A \xrightarrow{\mathsf{tr}_2^0.\mathsf{tr}_4^-} A_2'^0 \xrightarrow{\alpha.\mathsf{tr}_2'^-} A_2$$

with $A_1'^0$ and $A_2'^0$ canonical. By inductive hypothesis, we have that $A_1'^0 = A_2'^0$. We now must show $A_1 = A_2$. By action-determinism of $A$, there is only one process $P$ that can perform $\alpha$ in $A_1'^0 (= A_2'^0)$. The resulting process $P'$ after performing $\alpha$ is thus the same in the two executions. Since $A_1$ and $A_2$ are canonical and $\mathsf{tr}_1'^-$ and $\mathsf{tr}_2'^-$ contain only unobservable actions, $A_1 = A_2$.

In order to be able to apply our previous result, we complete the executions with all available unobservable actions:

$$A \xrightarrow{\mathsf{tr}_1} A_1 \xrightarrow{\mathsf{tr}_1^-} A_1' \text{ and } A \xrightarrow{\mathsf{tr}_2} A_2 \xrightarrow{\mathsf{tr}_1^-} A_2'$$

such that $A_1$ and $A_2$ are canonical and $\mathsf{tr}_1^-$ and $\mathsf{tr}_2^-$ contain only unobservable actions. We also have that:

- $\Phi(A_1) = \Phi(A_1')$ and $\mathsf{enable}(A_1) = \mathsf{enable}(A_1')$; and
- $\Phi(A_2) = \Phi(A_2')$ and $\mathsf{enable}(A_2) = \mathsf{enable}(A_2')$.

We now conclude thanks to our previous result, and obtain $A_1' = A_2'$ implying the desired equalities. □

The following proposition essentially lifts the previous one to the equivalence case.

**Proposition 11.** *Let $A$ and $B$ be two action-deterministic configurations such that $A \approx B$. If $A \xrightarrow{\mathsf{tr}_A} A'$ and $B \xrightarrow{\mathsf{tr}_B} B'$ with $\mathsf{obs}(\mathsf{tr}_A) = \mathsf{obs}(\mathsf{tr}_B)$ then $\Phi(A') \sim \Phi(B')$ and $\mathsf{enable}(A') = \mathsf{enable}(B')$.*

*Proof.* By hypothesis, we know that $A \approx B$, and also that $A \xrightarrow{\mathsf{tr}_A} A'$. Moreover, the freshness conditions on channels (*i.e.* $bc(\mathsf{tr}_A) \cap fc(B) = \emptyset$) holds as $B$ is able to perform $\mathsf{tr}_B$, and $\mathsf{tr}_A$ and $\mathsf{tr}_B$ share the same bound channels. Hence, we know that there exist $\mathsf{tr}'_B$ and $B''$ such that

$$B \xrightarrow{\mathsf{tr}'_B} B'', \mathsf{obs}(\mathsf{tr}_A) = \mathsf{obs}(\mathsf{tr}'_B), \text{ and } \Phi(A') \sim \Phi(B'').$$

Now, since $B$ is an action-deterministic configuration, applying Proposition 10 on $\mathsf{tr}_B$ and $\mathsf{tr}'_B$, we obtain that $\mathsf{enable}(B') = \mathsf{enable}(B'')$ and $\Phi(B') = \Phi(B'')$. This allows us to conclude that $\Phi(A') \sim \Phi(B')$.

It remains to show that $\mathsf{enable}(A') = \mathsf{enable}(B')$. By symmetry, we only show one inclusion. Let $\alpha_s \in \mathsf{enable}(A')$, we shall show that $\alpha_s \in \mathsf{enable}(B')$. We deduce from $\alpha_s \in \mathsf{enable}(A')$ that there is a trace $\mathsf{tr}'$ that is either $\alpha$ or $\tau.\alpha$ (where $\alpha$ is an observable action whose the skeleton is $\alpha_s$) such that

$$A \xrightarrow{\mathsf{tr}_A} A' \xrightarrow{\mathsf{tr}'} A_0$$

for some $A_0$. Since $A \approx B$, we know that there exist $\mathsf{tr}'_0$, $\mathsf{tr}'$, $B'_0$, and $B'$ such that

$$B \xrightarrow{\mathsf{tr}'_0} B'_0 \xrightarrow{\mathsf{tr}'} B_0$$

with $\Phi(A_0) \sim \Phi(B_0)$ and $\mathsf{obs}(\mathsf{tr}_A) = \mathsf{obs}(\mathsf{tr}'_0)$ and $\mathsf{obs}(\mathsf{tr}') = \alpha$. We have that the skeleton of $\alpha$ (that is $\alpha_s$) is in $\mathsf{enable}(B''_0)$ where $B''_0$ is the configuration producing $\alpha$ in the latter execution. Since $B''_0$ can be reached from $B'_0$ by executing $\tau$ actions, Proposition 8 implies $\alpha_s \in \mathsf{enable}(B'_0)$.

Now, since $B$ is an action-deterministic configuration, applying Proposition 10 on $\mathsf{tr}_B$ and $\mathsf{tr}'_0$ we obtain $\mathsf{enable}(B') = \mathsf{enable}(B'_0)$, and thus $\alpha_s \in \mathsf{enable}(B')$. □

The last ingredient which is essential to the Strong Symmetry Lemma (*i.e.* Lemma 2) allows to lift information about $\mathsf{enable}(\cdot)$ to information about $\mathsf{sk}(\cdot)$.

**Proposition 12.** *Let $A, B$ be two action-deterministic configurations and $P$ (resp. $Q$) a process of $A$ (resp. $B$). If $\mathsf{enable}(P) = \mathsf{enable}(Q)$ then $\mathsf{sk}(P) = \mathsf{sk}(Q)$.*

*Proof.* Let us show that $\mathsf{sk}(P) = \mathsf{sk}(Q)$. If $\mathsf{enable}(P)$ is an empty set then $P = 0$ and thus from $\mathsf{enable}(Q) = \emptyset$ we deduce that $Q = 0$ as well implying the required equality on skeletons. If $\mathsf{enable}(P)$ is a singleton then it must be $\{\mathsf{sk}(P)\}$ — we cannot be in the case where $P$ is a parallel composition, since, in that case there would be at least two skeletons in $\mathsf{enable}(P)$ by action-determinism of $A$ (and the fact a process $\Pi\{P_0\}^{\#}$ is reduced to $P_0$). The same goes with $Q$ thus we have $\{\mathsf{sk}(P)\} = \{\mathsf{sk}(Q)\}$. Finally, if $\mathsf{enable}(P)$ contains at least two skeletons then it must be the case that $P$ is a parallel composition of the form $\Pi\mathcal{S}$, $\mathcal{S} \neq \emptyset$ and $\mathsf{enable}(P) = \cup_{P' \in \mathcal{S}}\{\mathsf{sk}(P')\}$. Similarly, $Q$ must be of the form $\Pi\mathcal{S}_Q$ and $\mathsf{enable}(Q) = \cup_{Q' \in \mathcal{S}_Q}\{\mathsf{sk}(Q')\}$. Moroever, $|\mathsf{enable}(Q)| = |\mathcal{S}_Q|$ and $|\mathsf{enable}(P)| = |\mathcal{S}|$. Here, we make use of action-determinism to obtain that the number of subprocesses in parallel is the same as the cardinality of the sets of skeletons, and thus the same for $P$ and $Q$: indeed, no two parallel subprocesses can have the same skeleton. Hence $\uplus_{Q' \in \mathcal{S}_Q}\{\mathsf{sk}(Q')\} = \uplus_{P' \in \mathcal{S}}\{\mathsf{sk}(P')\}$ (as multisets). We conclude that $\mathsf{sk}(P) = \mathtt{par}(S)$ where $S$ is the ordered sequence of skeletons from $\uplus_{P' \in \mathcal{S}}\{\mathsf{sk}(P')\}$ (which are totally ordered by

action-determinacy), and $\mathsf{sk}(Q) = \mathtt{par}(S)$ where $S$ is the ordered sequence of skeletons from $\uplus_{Q' \in \mathcal{S}_Q}\{\mathsf{sk}(Q')\} = \uplus_{P' \in \mathcal{S}}\{\mathsf{sk}(P')\}$. □

Before proving Lemma 2, we state in the following corollary that requiring that initial configurations have same sets of labelled skeletons is not restrictive. Indeed, it suffices to pre-execute non-observable actions $\tau_{\shuffle}$ of the initial configurations. If the resulting configurations do not have the same set of skeletons then the following corollary allows to conclude that the configurations are not trace equivalent. Otherwise, it suffices to label them arbitrarily but consistently yielding configurations satisfying all necessary conditions for our developments (notably $\mathsf{skl}(A) = \mathsf{skl}(B)$).

**Corollary 1.** *Let $A, B$ be two action-deterministic configurations such that $A \approx B$ and any process in $A$ or $B$ is not a parallel composition. It holds that $\mathsf{sk}(A) = \mathsf{sk}(B)$.*

*Proof.* Applying Proposition 10 for $\mathsf{tr}_1 = \mathsf{tr}_2 = \epsilon$, we obtain $\mathsf{enable}(A) = \mathsf{enable}(B)$. Since no process in $A$ and $B$ is a parallel composition, one has that $\mathsf{enable}(A) = \uplus_{P \in A}\mathsf{enable}(P)$ and each $\mathsf{enable}(P)$ is a singleton (similarly for $B$). We thus deduce from Proposition 9 a bijection $m$ between processes in $A$ and $B$ such that for any $P$ in $A$, one has $\mathsf{enable}(P) = \mathsf{enable}(m(P))$. We thus conclude by applying Proposition 12: $\mathsf{sk}(A) = \uplus_{P \in A}\mathsf{sk}(P) = \uplus_{P \in A}\mathsf{sk}(m(P)) = \uplus_{Q \in B}\mathsf{sk}(Q) = \mathsf{sk}(B)$. □

We now conclude with the proof of Lemma 2.

*Proof of Lemma 2.* We show this result by induction on the length of the derivation $A \xrightarrow{\mathsf{tr}}_a A_n$. The case where $\mathsf{tr}$ is empty (*i.e.* no action even a unobservable one) is obvious. Assume that we have proved such a result for all the executions of length $n$, and we want to establish the result for an execution of length $n+1$.

Consider an execution of $[\alpha_1]^{\ell_1} \dots [\alpha_n]^{\ell_n}$ from $A$ to $A_n$, followed by $[\alpha_{n+1}]^{\ell_{n+1}}$ towards $A_{n+1}$. By induction hypothesis, we know that there exists an execution

$$B \xrightarrow{[\alpha_1]^{\ell_1}}_a B_1 \xrightarrow{[\alpha_2]^{\ell_2}}_a \dots \xrightarrow{[\alpha_n]^{\ell_n}}_a B_n$$

such that $\Phi(A_n) \sim \Phi(B_n)$ and $\mathsf{skl}(A_i) = \mathsf{skl}(B_i)$ for any $1 \leq i \leq n$. It remains to establish that there exists $B_{n+1}$ such that $B_n$ can perform $[\alpha_{n+1}]^{\ell_{n+1}}$ towards $B_{n+1}$, $\Phi(A_{n+1}) \sim \Phi(B_{n+1})$ and $\mathsf{skl}(A_{n+1}) = \mathsf{skl}(B_{n+1})$. We distinguish several cases depending on the action $\alpha_{n+1}$.

*Case $\alpha_{n+1} = \mathtt{zero}$.* We have that $[\mathtt{zero}]^{\ell_{n+1}} \in \mathsf{skl}(A_n)$, and thus, since $\mathsf{skl}(A_n) = \mathsf{skl}(B_n)$, we have also that $[\mathtt{zero}]^{\ell_{n+1}} \in \mathsf{sk}(B_n)$. We deduce that $A_n = (\{[0]^{\ell_{n+1}}\} \uplus \mathcal{P}_0; \Phi_0)$, and $B_n = (\{[0]^{\ell_{n+1}}\} \uplus \mathcal{Q}_0; \Psi_0)$ for some $\mathcal{P}_0$, $\mathcal{Q}_0$, $\Phi_0$, and $\Psi_0$. Moreover, since $\mathsf{skl}(A_n) = \mathsf{skl}(B_n)$, we deduce that $\mathsf{skl}(\mathcal{P}_0) = \mathsf{skl}(\mathcal{Q}_0)$. Let $B_{n+1} = (\mathcal{Q}_0; \Psi_0)$. We have that:

- $B_n = (\{[0]^{\ell_{n+1}}\} \uplus \mathcal{Q}_0; \Psi_0) \xrightarrow{[\mathtt{zero}]^{\ell_{n+1}}}_a (\mathcal{Q}_0; \Psi_0) = B_{n+1}$,
- $\Phi(A_{n+1}) = \Phi(A_n) \sim \Phi(B_n) = \Phi(B_{n+1})$, and
- $\mathsf{skl}(A_{n+1}) = \mathsf{skl}(\mathcal{P}_0) = \mathsf{skl}(\mathcal{Q}_0) = \mathsf{skl}(B_{n+1})$.

*Case* $\alpha_{n+1} = \mathtt{par}(S)$ *for some sequence* $S = (\beta_1, \ldots, \beta_k)$. Note that this sequence is ordered according to our order $<$ over skeletons (*i.e.* $\beta_1 < \ldots < \beta_k$) and $\beta_i$'s are pairwise distinct by action-determinism of $A$. Since $A_n$ is able to produce the action $\mathtt{par}(S)$, it must be of the form $A_n = (\{[\Pi \uplus_{i=1}^{k} \{P_i\}]^{\ell_{n+1}}\} \uplus \mathcal{P}_0; \Phi_0)$ for some $P_i$, $\mathcal{P}_0$ and $\Phi_0$ such that $\forall i, \mathsf{sk}(P_i) = \beta_i$. It follows that $[\mathtt{par}(S)]^{\ell_{n+1}} \in \mathsf{skl}(A_n)$, and thus, since $\mathsf{skl}(A_n) = \mathsf{skl}(B_n)$, we have also that $[\mathtt{par}(S)]^{\ell_{n+1}} \in \mathsf{skl}(B_n)$. The latter implies $B_n = (\{[\Pi \mathcal{S}_Q]^{\ell_{n+1}}\} \uplus \mathcal{Q}_0; \Psi_0)$ for some $\mathcal{S}_Q$, $\mathcal{Q}_0$ and $\Psi_0$ such that $\mathsf{sk}(\mathcal{S}_Q) = \{\beta_i\}_i$. Further, we have

$$A_{n+1} = (\uplus_{i=1}^{k} \{[P_i]^{\ell_{n+1} \cdot i}\} \uplus \mathcal{P}_0; \Phi_0).$$

Moreover, since $\mathsf{skl}(A_n) = \mathsf{skl}(B_n)$, we deduce that $\mathsf{skl}(\mathcal{P}_0) = \mathsf{skl}(\mathcal{Q}_0)$, and

$$\{\mathsf{sk}(Q_j) \mid Q_j \in \mathcal{S}_Q\}^{\#} = \{\mathsf{sk}(P_i) \mid 1 \leq i \leq k\}^{\#} = \{\beta_1, \ldots, \beta_k\}$$

Remark that, since $P_i$ (resp. $Q_j$) cannot be a zero or a parallel we have that $\mathsf{enable}(P_i) = \{\mathsf{sk}(P_i)\}$ (resp. $\mathsf{enable}(Q_i) = \{\mathsf{sk}(Q_i)\}$) and those sets are singletons and $|\mathcal{S}_Q| = k$. Moreover, by action-determinism of $A$ and $B$ we know that all those singletons are pairwise disjoint. From this, we conclude that there exists a permutation $\pi$ over $[1; k]$ such that

$$\forall i,\ \mathsf{sk}(Q_{\pi(i)}) = \beta_i = \mathsf{sk}(P_i)$$

and thus

$$\forall i,\ \mathsf{skl}([Q_{\pi(i)}]^{\ell_{n+1} \cdot i}) = \mathsf{skl}([P_i]^{\ell_{n+1} \cdot i})$$

We can finally let $B_{n+1}$ be $(\uplus_{i=1}^{k} \{[Q_{\pi(i)}]^{\ell_{n+1} \cdot i}\} \uplus \mathcal{Q}_0; \Psi_0)$ and we have:

- $B_n = (\{[\Pi \uplus_{i=1}^{k} Q_{\pi(i)}]^{\ell_{n+1}}\} \uplus \mathcal{Q}_0; \Psi_0) \xrightarrow{[\mathtt{par}(S)]^{\ell_{n+1}}}_a (\uplus_{i=1}^{k} \{[Q_{\pi(i)}]^{\ell_{n+1} \cdot i}\} \uplus \mathcal{Q}_0; \Psi_0) = B_{n+1}$,
- $\Phi(A_{n+1}) = \Phi(A_n) \sim \Phi(B_n) = \Phi(B_{n+1})$, and
- $\begin{aligned} \mathsf{skl}(A_{n+1}) &= \mathsf{skl}(\mathcal{P}_0) \ \uplus \ \biguplus_{i=1}^{k} \mathsf{skl}([P_i]^{\ell_{n+1} \cdot i}) \\ &= \mathsf{skl}(\mathcal{Q}_0) \ \uplus \ \biguplus_{i=1}^{k} \mathsf{skl}([Q_{\pi(i)}]^{\ell_{n+1} \cdot i}) = \mathsf{skl}(B_{n+1}). \end{aligned}$

*Case* $\alpha_{n+1} = \mathtt{in}(c, M)$ *for some* $c$, *and* $M$ *with* $M \in \mathcal{T}(\Sigma_{\mathsf{pub}}, \mathrm{dom}(\Phi(A_n)))$. We have that $[\mathtt{in}_c]^{\ell_{n+1}} \in \mathsf{skl}(A_n)$, and thus, since $\mathsf{skl}(A_n) = \mathsf{skl}(B_n)$, we have $[\mathtt{in}_c]^{\ell_{n+1}} \in \mathsf{skl}(B_n)$. We deduce that $A_n = (\{[\mathtt{in}(c, x_A).P]^{\ell_{n+1}}\} \uplus \mathcal{P}_0; \Phi_0)$, and $B_n = (\{[\mathtt{in}(c, x_B).Q]^{\ell_{n+1}}\} \uplus \mathcal{Q}_0; \Psi_0)$ for some $x_A$, $x_B$, $P$, $Q$, $\mathcal{P}_0$, $\mathcal{Q}_0$, $\Phi_0$, and $\Psi_0$. We also deduce that there exists a message $u_A$ such that $M\Phi_0 \Downarrow u_A$. Moreover, since $\mathsf{skl}(A_n) = \mathsf{skl}(B_n)$, we deduce that $\mathsf{skl}(\mathcal{P}_0) = \mathsf{skl}(\mathcal{Q}_0)$. By inductive hypothesis, we also have that $\Phi_0 \sim \Psi_0$; hence a message $u_B$ such that $M\Psi_0 \Downarrow u_B$. Let $B_{n+1} = ([Q\{u_B/x_B\}]^{\ell_{n+1} \cdot \boxdot} \uplus \mathcal{Q}_0; \Psi_0)$. We have that:

- $B_n \xrightarrow{[\mathtt{in}(c, M)]^{\ell_{n+1}}}_a B_{n+1}$, and
- $\Phi(A_{n+1}) = \Phi(A_n) \sim \Phi(B_n) = \Phi(B_{n+1})$.

It remains to show that $\mathsf{skl}(A_{n+1}) = \mathsf{skl}(B_{n+1})$. Since we have that $\mathsf{skl}(\mathcal{P}_0) = \mathsf{skl}(\mathcal{Q}_0)$, and the label of the new subprocess is the same (namely, $\ell_{n+1} \cdot \boxdot$) on both sides, we only need to show that:

$$\mathsf{sk}(P\{u_A/x_A\}) = \mathsf{sk}(Q\{u_B/x_B\})$$

In order to improve the readability, we will note $P' = P\{u_A/x_A\}$ and $Q' = Q\{u_B/x_B\}$. We have that $A$ and $B$ are two action-deterministic configurations such that $A \approx B$. Moreover, they perform the same trace, respectively towards $A_{n+1}$ and $B_{n+1}$. Thus, thanks to Proposition 11, we deduce that $\mathsf{enable}(A_{n+1}) = \mathsf{enable}(B_{n+1})$. Moreover, our hypothesis $\mathsf{skl}(\mathcal{P}_0) = \mathsf{skl}(\mathcal{Q}_0)$ implies that $\mathsf{enable}(\mathcal{P}_0) = \mathsf{enable}(\mathcal{Q}_0)$, and thus we deduce that $\mathsf{enable}(P') = \mathsf{enable}(Q')$ (recall that by action-determinism, unions of the form $\mathsf{enable}(A_{n+1}) = \mathsf{enable}(\mathcal{P}_0) \cup \mathsf{enable}(P')$ are actually disjoint unions). We conclude using Proposition 12.

*Case* $\alpha_{n+1} = \mathtt{out}(c, w)$ *for some* $c$ *and some* $w$ *with* $w \notin \mathrm{dom}(\Phi(A_n))$. This case is similar to the previous one. However, during such a step, the frame of each configuration is enriched, and thus the fact that $\Phi(A_{n+1}) \sim \Phi(B_{n+1})$ is now a consequence of Proposition 11.

*Case* $\alpha_{n+1} = \mathtt{sess}(a, \overline{c})$ *for some* $a$*, and some* $\overline{c}$. Firstly, we show for later that $\overline{c}$ are fresh in $B_n$. Indeed, we deduce from $bc(\alpha_1 \dots \alpha_{n+1}) \cap fc(B) = \emptyset$ that $\overline{c}$ are fresh in $B$ and we know that free channels of $B_n$ are included in $fc(B) \cup bc(\alpha_1 \dots \alpha_n)$. Thereby, if there was a channel $c_i \in \overline{c} \cap fc(B_n)$ it would be in $bc(\alpha_1 \dots \alpha_n)$ but this is forbidden because of the freshness condition (in the current trace) over channels, *i.e.* new channels cannot be introduced twice (once in $\alpha_1 \dots \alpha_n$ and once in $\alpha_{n+1}$).

As before, we obtain $\ell_{n+1} = \ell \cdot \overline{c}$ and

$$A_n = (\{[!^a_{\overline{c},\overline{n_P}} P]^\ell\} \uplus \mathcal{P}_0; \Phi_0) \quad \text{and} \quad B_n = (\{[!^a_{\overline{c},\overline{n_Q}} Q]^\ell\} \uplus \mathcal{Q}_0; \Psi_0)$$

for some $P$, $Q$, $\mathcal{P}_0$, $\mathcal{Q}_0$, $\Phi_0$, and $\Psi_0$. Moreover, since $\mathsf{skl}(A_n) = \mathsf{skl}(B_n)$, we deduce that $\mathsf{skl}(\mathcal{P}_0) = \mathsf{skl}(\mathcal{Q}_0)$.

We have:

$$A_{n+1} = (\{[P]^{\ell_{n+1}}, [!^a_{\overline{c},\overline{n_P}} P]^\ell\} \uplus \mathcal{P}_0; \Phi_0)$$

Accordingly, let us pose

$$B_{n+1} = (\{[Q]^{\ell_{n+1}}, [!^a_{\overline{c},\overline{n_Q}} Q]^\ell\} \uplus \mathcal{Q}_0; \Psi_0).$$

We have that:

- $B_n \xrightarrow{[\mathsf{sess}(a,\overline{c})]^{\ell_{n+1}}}_a B_{n+1}$; and
- $\Phi(A_{n+1}) = \Phi(A_n) \sim \Phi(B_n) = \Phi(B_{n+1})$.

It remains to show that $\mathsf{skl}(A_{n+1}) = \mathsf{skl}(B_{n+1})$. Since we have that $\mathsf{skl}(\mathcal{P}_0) = \mathsf{skl}(\mathcal{Q}_0)$, and since the labels of corresponding subprocesses are the same on both sides, we only need to show that:

- $\mathsf{sk}(P) = \mathsf{sk}(Q)$ and
- $\mathsf{sk}(!^a_{\overline{c},\overline{n_P}} P) = \mathsf{sk}(!^a_{\overline{c},\overline{n_Q}} Q)$

As in the previous case, thanks to Proposition 11, we know that $\mathsf{enable}(A_{n+1}) = \mathsf{enable}(B_{n+1})$, and we deduce that $\mathsf{enable}(P) = \mathsf{enable}(Q)$ and $\mathsf{enable}(!^a_{c,\overline{n_P}}P) = \mathsf{enable}(!^a_{c,\overline{n_Q}}Q)$, which allows us to conclude using Proposition 12. □

## 5.3 Compression

Our first refinement of the semantics, which we call compression, is closely related to focusing from proof theory [And92]: we will assign a polarity to processes and constrain the shape of executed traces based on those polarities. This will provide a first significant reduction of the number of traces to consider when checking reachability-based properties such as secrecy, and more importantly, equivalence-based properties in the action-deterministic case. We illustrate below one of the aspect of the compressed semantics.

**Example 24.** *Consider the process* $(\mathcal{P};\Phi)$ *with* $\mathcal{P} = \{\mathtt{in}(c_1,x).P_1, \mathtt{out}(c_2,b).P_2\}$. *In order to reach* $(\{P_1\{x \mapsto u\}, P_2\}; \Phi \cup \{w \mapsto b\})$, *we have to execute the action* $\mathtt{in}(c_1,x)$ *(using a recipe* $M$ *that allows one to deduce* $u$*) and the action* $\mathtt{out}(c_2,b)$ *(giving us a label of the form* $\mathtt{out}(c_2,w)$*). In case of reachability properties, the execution order of these actions only matters if* $M$ *uses* $w$*. Thus we can safely perform the outputs in priority. It turns out that, when one has such choices, executing outputs in priority is always a* complete *strategy;* i.e. *all reachable states are actually explored.*

*The situation is more complex when considering trace equivalence. In that case, we are concerned not only with reachable states, but also with* how *those states are reached. Quite simply, traces matter. Thus, if we want to discard the trace* $\mathtt{in}(c_1,M).\mathtt{out}(c_2,w)$ *when studying processes* $\mathcal{P}$ *and consider only its permutation* $\mathtt{out}(c_2,w).\mathtt{in}(c_1,M)$*, we have to make sure that the same permutation is available on the other process. The key to ensure that identical permutations will be available on both sides of the equivalence is the action-deterministic assumption and the Strong Symmetry Lemma (*i.e. *Lemma 2).*

### 5.3.1 Compressed Strategy

The semantics we shall define is based on syntactical criteria over processes and configurations as formalised next.

**Definition 22** (Polarity assignment)**.** *A process* $P$ *is* positive *if it is of the form* $\mathtt{in}(c,x).Q$*, and is* negative *otherwise. A multiset of processes* $\mathcal{P}$ *is* initial *if it contains only positive or* replicated *processes (*i.e. *of the form* $!^a_{c,\overline{n}}Q$*).*

The compressed semantics (see Figure 5.3) is built upon the annotated semantics. It constrains the traces to follow a particular strategy, alternating between *negative* and *positive* phases. It uses enriched configurations of the form $(\mathcal{P};F;\Phi)$ where $(\mathcal{P};\Phi)$ is a labelled configuration and $F$ is either a process (signalling which process is *under focus* in the positive phase) or $\varnothing$ (in the negative phase). The negative phase lasts until the configuration is initial (*i.e.* unfocused with an initial underlying multiset of processes) and in that phase we perform actions that decompose

$$\text{Start/In}\quad \frac{\mathcal{P} \text{ is initial} \quad (P;\Phi) \xrightarrow{\mathtt{in}(c,M)}_a (P';\Phi)}{(\mathcal{P} \uplus \{P\};\varnothing;\Phi) \xrightarrow{\mathtt{foc}(\mathtt{in}(c,M))}_c (\mathcal{P};P';\Phi)}$$

$$\text{Start/!}\quad \frac{\mathcal{P} \text{ is initial} \quad (!^a_{\overline{c},\overline{n}}P;\Phi) \xrightarrow{\mathtt{sess}(a,\overline{c})}_a (\{!^a_{\overline{c},\overline{n}}P;Q\};\Phi)}{(\mathcal{P} \uplus \{!^a_{\overline{c},\overline{n}}P\};\varnothing;\Phi) \xrightarrow{\mathtt{foc}(\mathtt{sess}(a,\overline{c}))}_c (\mathcal{P} \uplus \{!^a_{\overline{c},\overline{n}}P\};Q;\Phi)}$$

$$\text{Pos/In}\quad \frac{(P;\Phi) \xrightarrow{\mathtt{in}(c,M)}_a (P';\Phi)}{(\mathcal{P};P;\Phi) \xrightarrow{\mathtt{in}(c,M)}_c (\mathcal{P};P';\Phi)}$$

$$\text{Neg}\quad \frac{(P;\Phi) \xrightarrow{\alpha}_a (\mathcal{P}';\Phi')}{(\mathcal{P} \uplus \{P\};\varnothing;\Phi) \xrightarrow{\alpha}_c (\mathcal{P} \uplus \mathcal{P}';\varnothing;\Phi')} \ \alpha \in \{\mathtt{par}(\bullet), \mathtt{zero}, \mathtt{out}(\bullet,\bullet)\}$$

$$\text{Release}\quad (\mathcal{P};[P]^\ell;\Phi) \xrightarrow{[\mathtt{rel}]^\ell}_c (\mathcal{P} \uplus \{[P]^\ell\};\varnothing;\Phi) \quad \text{when } P \neq 0 \text{ is negative}$$

$$\text{Release}_\perp\quad (\mathcal{P};[0]^\ell;\Phi) \xrightarrow{[\mathtt{rel}]^\ell.[\mathtt{zero}]^\ell}_c (\emptyset;\varnothing;\Phi)$$

Labels are implicitly set in the same way as in the annotated semantics. Neg is made non-branching by imposing an arbitrary order on labelled skeletons of available actions.

**Figure 5.3** Compressed semantics

negative non-replicated processes. This is done using the Neg rule, in a completely deterministic way. When the configuration becomes initial, a positive phase can be initiated: we choose one process and start executing the actions of that process (only inputs, possibly preceded by a new session) without the ability to switch to another process of the multiset, until a negative subprocess different from 0 is released (we explain what happens otherwise in Subsection 5.3.2) and we go back to the negative phase. The active process in the positive phase is said to be *under focus*. Between any two initial configurations, the compressed semantics executes a sequence of actions, called *blocks*, of the form $\mathtt{foc}(\alpha).\mathsf{tr}^+.\mathtt{rel}.\mathsf{tr}^-$ where $\mathsf{tr}^+$ is a (possibly empty) sequence of input actions, whereas $\mathsf{tr}^-$ is a non empty sequence of `out`, `par`, and `zero` actions. Note that, except for choosing recipes, the compressed semantics is completely non-branching when executing a block. It may branch only when choosing which block is performed.

**Example 25.** *Consider the process* $P = !^a_{c,k}\mathtt{in}(c,x).\mathtt{out}(c,\mathtt{senc}(x,k)).0$. *We have that (omitting labels and assuming the existence of a message* $u_i$ *such that* $M_i\Phi \Downarrow u_i$*):*

$$\begin{aligned}(\{P\};\varnothing;\Phi) &\xrightarrow{\mathtt{foc}(\mathtt{sess}(a,c_i))}_c (\{P\};\{\mathtt{in}(c_i,x).\mathtt{out}(c_i,\mathtt{senc}(x,k_i)).0\};\Phi)\\ &\xrightarrow{\mathtt{in}(c_i,M_i).\mathtt{rel}}_c (\{P,\mathtt{out}(c_i,\mathtt{senc}(u_i,k_i)).0\};\varnothing;\Phi)\\ &\xrightarrow{\mathtt{out}(c_i,w_i).\mathtt{zero}}_c (\{P\};\varnothing;\Phi').\end{aligned}$$

*Once a replication is performed, the resulting process is under focus and must be executed in priority until the end. Note that, after executing the input, the resulting process is negative and, thus, still has priority. Thus, on this example, the compressed semantics only explores executions that are made of blocks of the form* $\mathtt{sess}(a,c_i).\mathtt{in}(c_i,M_i).\mathtt{out}(c_i,w_i)$ *and discards all other interleavings (*e.g. *the one that unfolds two sessions, executes the two available inputs and then executes the two available outputs).*

**Remark 4.** *It is clear why disassembling parallel compositions and erasing zeros take part to the semantics instead of the internal reduction. Indeed, if one had chosen to add $\mathscr{R}_{\text{par}}$ to the relation on which the internal reduction is based then the induced internal reduction $\rightsquigarrow_{\mathscr{R}}$ would always break parallel compositions at top level. Consequently, the semantics would never have access to the parallel composition structure because it only handles configurations in normal form. However, it is now clear that the compressed semantics needs to have access to the configuration's structure details. For example, this is needed to decide to stop the current focus because new processes in parallel have just been made available.*

### 5.3.2 Improper Blocks and Release$_\perp$ Rule

Note that blocks of the form $\mathtt{foc}(\alpha).\mathsf{tr}^+.\mathtt{rel}.\mathtt{zero}$ that we call *improper blocks* do not bring any new information to the attacker. One could argue that it is then useless to explore such blocks. However, when checking trace equivalence, how to make sure then that both configurations can exhibit exactly the same kinds of improper blocks? Moreover, how can one predict, when choosing a focus, that the initiated block will be improper[2] in order to not explore it ?

While it would be incorrect to fully ignore such *improper* blocks, it is in fact sufficient to only consider them at the end of traces. This is implemented by the rule Release$_\perp$: if the current block is released improperly (*i.e.* the produced block is improper) then the semantics stops exploring (*i.e.* the multiset of processes is set to $\emptyset$). We give intuitions explaining why this does not break completeness (w.r.t. trace equivalence) in the next example. We call *proper* the blocks that are not improper and *proper trace* a trace made of proper blocks.

**Example 26.** *Consider $\mathcal{P} = \{\mathtt{in}(c,x).\mathtt{in}(c,y),\ \mathtt{in}(c',x')\}$. Its compressed traces are of the form $\mathtt{foc}(\mathtt{in}(c,M)).\mathtt{in}(c,N).\mathtt{rel}.\mathtt{zero}$ and $\mathtt{foc}(\mathtt{in}(c',M')).\mathtt{rel}.\mathtt{zero}$. The concatenation of those two improper traces cannot be executed in the compressed semantics. Intuitively, we do not lose anything for trace equivalence, because if a process can exhibit those two improper blocks they must be in parallel (by action-determinacy) and hence considering their combination is redundant.*

Finally, in order to ease the presentation and the structure of future proofs, we introduce a variant of the compressed semantics which does not make any distinction between proper and improper blocks. To that end, we define the *semi-compressed semantics* (noted $\xrightarrow{\bullet}_{sc}$) similarly to the compressed semantics based on the rules from Figure 5.3 except Release and Release$_\perp$ that are replaced by the single following rule:

$$\text{Release}^s \qquad (\mathcal{P}; [P]^\ell; \Phi) \xrightarrow{[\mathtt{rel}]^\ell}_{sc} (\mathcal{P} \uplus \{[P]^\ell\}; \varnothing; \Phi) \quad \text{when } P \text{ is negative}$$

**Remark 5** (Relation with Focusing)**.** *The reader familiar with focused proof systems will have recognised the strong similarities: the phases of the compressed semantics are the same as those of focused proof systems, and more deeply, the choice of polarities for each process construction can actually be derived from encoding our process algebra into linear logic in such a way that*

[2]Remind that conditionals are executed by the internal reduction and their outcomes are only known "at runtime", so do the explored branches (either Then or Else).

*proof search corresponds to executions. While this encoding provides an intuitive guide, it would not be obvious to obtain directly our results from the completeness of focusing in linear logic — for instance, internal reduction would not naturally fit in. We give below (Lemma 4) a direct proof of completeness (*i.e. *all reachable states are explored by the compressed strategy) using the idea of the positive trunk argument of [MS07]. Besides being self-contained, this has the advantage of setting the stage for the next section, where we go beyond what focusing actually allows, but still exploit the positive trunk argument to carry out our analysis. In Section 7.3, we discuss at greater length related work in proof theory.*

### 5.3.3 Reachability

We now formalise the relationship between traces of the compressed and annotated semantics. In order to do so, we translate between configurations and enriched configurations as follows:

$$\lceil(\mathcal{P};\Phi)\rceil = (\mathcal{P};\varnothing;\Phi),\ \lfloor(\mathcal{P};\varnothing;\Phi)\rfloor = (\mathcal{P};\Phi) \text{ and } \lfloor(\mathcal{P};P;\Phi)\rfloor = (\mathcal{P}\uplus\{P\};\Phi).$$

Similarly, we map compressed traces to annotated ones:

$$\lfloor\epsilon\rfloor = \epsilon,\ \lfloor\mathtt{foc}(\alpha).\mathsf{tr}\rfloor = \alpha.\lfloor\mathsf{tr}\rfloor,\ \lfloor\mathtt{rel}.\mathsf{tr}\rfloor = \lfloor\mathsf{tr}\rfloor, \text{ and } \lfloor\alpha.\mathsf{tr}\rfloor = \alpha.\lfloor\mathsf{tr}\rfloor \text{ otherwise.}$$

We observe that we can map any execution in the compressed semantics to an execution in the annotated semantics. Indeed, a compressed execution that does not use $\text{Release}_\perp$ is simply an annotated execution with some extra annotations (*i.e.* `foc` and `rel`) indicating positive/negative phase changes. Moreover, even with possibly a rule $\text{Release}_\perp$ at the end, we obtain same reachable frames with $\rightarrow_a$.

**Lemma 3.** *Let $A$, $B$ be configurations and $\mathsf{tr}_c$ be a trace such that $A \xrightarrow{\mathsf{tr}_c}_c B$. If $\mathsf{tr}_c$ is proper then $\lfloor A\rfloor \xrightarrow{\lfloor\mathsf{tr}_c\rfloor}_a \lfloor B\rfloor$. Otherwise, there exists a configuration $B'$ such that $\lfloor A\rfloor \xrightarrow{\lfloor\mathsf{tr}_c\rfloor}_a B'$ with $\Phi(B') = \Phi(B)$.*

Going in the opposite direction is more involved. In general, mapping annotated executions to compressed ones requires to reorder actions. Compressed executions also force negative actions to be performed unconditionally and blocks to be fully executed. One way to handle this is to consider *complete* executions of a configuration, *i.e.* executions after which no more action can be performed except possibly the ones that consist in unfolding a replication (*i.e.* rule Repl).

Last but not least, compressed executions stop exploring when reaching an improper block while annotated semantics is able to continue. We first prove a completeness result for the semi-compressed semantics $\rightarrow_{sc}$ for which all complete, annotated, executions can be mapped to one compressed execution after permutations of independent actions. Second, we extend the first completeness result for the full compressed semantics. However, instead of mapping an annotated execution to a compressed execution, we only map an annotated execution to (possibly many) compressed executions each exploring a different improper block at the end.

Inspired by the positive trunk argument of [MS07], we show the first completeness lemma.

**Lemma 4.** *Let $A$, $A'$ be two configurations, $\mathsf{tr}$ a trace and $A \xrightarrow{\mathsf{tr}}_a A'$ a complete, annotated execution. There exists a trace $\mathsf{tr}_c$, such that $\lfloor \mathsf{tr}_c \rfloor$ can be obtained from $\mathsf{tr}$ by swapping independent labelled actions, and $\lceil A \rceil \xrightarrow{\mathsf{tr}_c}_{sc} \lceil A' \rceil$.*

*Proof.* Let $A = (\mathcal{P}; \Phi)$ be a configuration and $(\mathcal{P}; \Phi) \xrightarrow{\mathsf{tr}}_a A'$ a complete execution. We proceed by induction on the length of $\mathsf{tr}$, distinguishing two cases.

*Case 1.* We first consider the case where there is at least one process in $\mathcal{P}$ that is negative and non-replicated. Since we are considering a complete execution, at least one negative action $\alpha$ is performed on this process in $\mathsf{tr}$. This action may be an output, the decomposition of a parallel composition, or the removal of a zero. If there are more than one such action, we choose the one that can be performed using Neg, *i.e.* the one that is minimal according to our arbitrary order on labelled skeletons. Since our action can be performed initially by our process, and by well-labelling, the label of the action is independent with all labels of previously executed actions in $\mathsf{tr}$. Moreover, there cannot be any second-order dependency between $\alpha$ and one of those actions. Indeed, if $\alpha$ is an output, no input performed before $\alpha$ is able to use the handle of $\alpha$. It can thus be swapped before all the others by using Lemma 1, obtaining an execution of trace $\alpha.\mathsf{tr}'$ ending in the same configuration $A'$. The rule Neg can be performed in the compressed semantics to trigger the action $\alpha$, and by induction hypothesis on $\mathsf{tr}'$ we can complete our compressed execution towards $A'$.

*Case 2.* Otherwise, when $\mathcal{P}$ contains only positive or replicated processes, we must choose one process to focus on, start a positive phase and execute all its actions until we can finally release the focus. As long as all processes are positive or replicated, only input or session actions can be performed. In either case, the action yields a new process (the continuation of the input, or the new session) which may be negative or positive. We define the *positive prefix* of our execution as the prefix of actions for which all but the last transition yield a positive process. It is guaranteed to exist because $A'$ contains only negative processes.

The positive prefix is composed only of input and replication actions. Because session actions are performed by negative processes, and no new negative process is created in the positive prefix, session actions can be permuted at the beginning of the prefix thanks to Lemma 1. Thus, we assume without loss of generality that the prefix is composed of session actions, followed by input actions: we write $\mathsf{tr} = \mathsf{tr}_!.\mathsf{tr}_{\mathrm{in}}.\mathsf{tr}_0$. Note that actions in $\mathsf{tr}_!$ are pairwise independent since they correspond to unfoldings of replicated processes that were initially present in $\mathcal{P}$. In the portion of the execution where inputs of $\mathsf{tr}_{\mathrm{in}}$ are performed, there is an obvious bijective mapping between the processes of any configuration and its successor, allowing us to follow execution threads, and to freely permute inputs pertaining to different threads. Such permutations are made possible by Lemma 1. Indeed, they concern actions that are (i) sequentially independent (*i.e.* labels are independent) since two different threads involve actions performed by processes in parallel and (ii) recipe independent since there is no output action in $\mathsf{tr}_{\mathrm{in}}$.

The last action of the positive prefix releases a negative process $P^-$. Let $P$ be its antecedent (through its corresponding thread) in the configuration obtained after the execution of $\mathsf{tr}_!$. We

have that:

$$A = (\mathcal{P}; \Phi) \xrightarrow{\mathsf{tr}_1}_a (\mathcal{P}_1 \uplus \{P\}; \Phi) \xrightarrow{\mathsf{tr}_{\text{in}}}_a (\mathcal{P}_2 \uplus \{P^-\}; \Phi) \xrightarrow{\mathsf{tr}_0}_a A'$$

Now we can write $\mathcal{P} = \mathcal{P}_0 \uplus \{P_\mathsf{f}\}$ where $P_\mathsf{f}$ is either $P$ or a replicated process that gives rise to $P$ in one transition. By permuting actions pertaining to $P_\mathsf{f}$ before all others thanks to Lemma 1 and previous remarks, we obtain an execution of the form

$$A = (\mathcal{P}_0 \uplus \{P_\mathsf{f}\}; \Phi) \xrightarrow{\mathsf{tr}_1}_a (\mathcal{P}'_0 \uplus \{P^-\}; \Phi) \xrightarrow{\mathsf{tr}_2}_a (\mathcal{P}_2 \uplus \{P^-\}; \Phi) \xrightarrow{\mathsf{tr}_0}_a A'$$

where $\mathsf{tr}_1.\mathsf{tr}_2.\mathsf{tr}_0$ can be obtained from $\mathsf{tr}$ by swapping independent labelled actions, $\mathsf{tr}_1 = [\alpha]^\ell.\mathsf{tr}'_1$, and $\mathcal{P}'_0 = \mathcal{P}_0$ when $P_\mathsf{f} = P$, and $\mathcal{P}'_0 = \mathcal{P}_0 \uplus \{P_\mathsf{f}\}$ otherwise.

In the compressed semantics, if we initiate a focus on $P_\mathsf{f}$, we can execute the actions of $\mathsf{tr}_1$ and release the focus when reaching $P^-$ (using the rule Release$^s$), *i.e.* we have that:

$$(\mathcal{P}; \varnothing; \Phi) \xrightarrow{[\mathtt{foc}(\alpha)]^\ell}_{sc} \xrightarrow{\mathsf{tr}'_1}_{sc} (\mathcal{P}'_0; P^-; \Phi) \xrightarrow{[\mathtt{rel}]^{\ell'}}_{sc} (\mathcal{P}'_0 \uplus \{P^-\}; \varnothing; \Phi)$$

where $\ell'$ is the label of $P^-$. We can conclude by induction hypothesis on $\mathsf{tr}_2.\mathsf{tr}_0$. □

Next, we show the second completeness result covering the full compressed semantics. We say that two blocks $b_1$ and $b_2$ are independent when the actions from one and the other are independent.

**Lemma 5.** *Let $A$, $A'$ be two configurations, $\mathsf{tr}$ be a trace and $A \xrightarrow{\mathsf{tr}}_a A'$ be a complete, annotated, execution. There exists a configuration $A_0$, a proper trace $\mathsf{tr}_c$ and a (possibly empty) sequence of pairwise independent improper blocks $b_1, \ldots, b_k$, such that $\lfloor \mathsf{tr}_c.b_1.\ldots.b_k \rfloor$ can be obtained from $\mathsf{tr}$ by swapping independent labelled actions, and for all $1 \leq i \leq k$, there exists a configuration $A_i$ and a compressed execution $\lceil A \rceil \xrightarrow{\mathsf{tr}_c}_c A_0 \xrightarrow{b_i}_c A_i$ such that $\Phi(A_i) = \Phi(A') = \Phi(A_0)$.*

*Proof.* Applying Lemma 4, we obtain a trace $\mathsf{tr}'$, such that $\lfloor \mathsf{tr}' \rfloor$ can be obtained from $\mathsf{tr}$ by swapping independent labelled actions, and $\lceil A \rceil \xrightarrow{\mathsf{tr}'}_{sc} \lceil A' \rceil$. Let $b_1, \ldots, b_k$ be the improper blocks that occur in $\mathsf{tr}'$. We argue that all those blocks are pairwise independent and each block $b_i$ is independent with all actions occurring after $b_i$ in $\mathsf{tr}'$. Both claims follow from (i) the absence of recipe dependencies since blocks $b_i$ do not feature outputs and (ii) the fact that each block $b_i$ gives rise to a null process that disappears from the multiset along with the label it carried out preventing sequential dependencies.

Hence a proper trace $\mathsf{tr}_c$ such that $\mathsf{tr}_c.b_1.\ldots.b_k$ can be obtained from $\mathsf{tr}$ by swapping independent blocks. Those swaps can also be made along the former semi-compressed execution since before starting a new block, the semi-compressed semantics allows to choose any positive process to start a focus. We thus have an execution $\lceil A \rceil \xrightarrow{\mathsf{tr}_c.b_1.\ldots.b_k}_{sc} \lceil A' \rceil$.

Since blocks $b_i$ are pairwise independent, there exist initial configurations $A_0, A_1, \ldots, A_k$ such that $\lceil A \rceil \xrightarrow{\mathsf{tr}_c}_{sc} A_0$, and

$$A_0 \xrightarrow{b_1}_{sc} A_1,\ A_0 \xrightarrow{b_2}_{sc} A_2,\ \ldots, A_0 \xrightarrow{b_k}_{sc} A_k.$$

Since $\mathsf{tr}_c$ is a proper trace, the execution $\lceil A \rceil \xrightarrow{\mathsf{tr}_c}_{sc} A_0$ can easily be translated into a compressed execution $\lceil A \rceil \xrightarrow{\mathsf{tr}_c}_c A_0$ using rule Release instead of Release$^s$. Finally, remark that all executions

$A_0 \xrightarrow{b_i}_{sc} A_i$ can be done using the compressed execution instead of the semi-compressed execution: it suffices to trigger rule $\text{Release}_\perp$ instead of $\text{Release}^s$. We thus obtain executions $A_0 \xrightarrow{b_i}_c (\emptyset; \varnothing; \Phi(A_i))$ for all $1 \leq i \leq k$. □

**Remark 6** (Reachability properties verified with the compressed semantics)**.** *When checking a security property that can be decided by looking only at reachable frames obtained through complete executions (*e.g. *secrecy properties), lemmas 3 to 5 show that the semi-compressed semantics or the compressed semantics can be used instead of the regular one. More generally, the semi-compressed semantics is sound and complete for reachable predicates that are:*

- *monotone (*i.e. *if an execution $A \xrightarrow{\mathsf{tr}} (\mathcal{P}; \Phi)$ satisfies the predicate then so does $A \xrightarrow{\mathsf{tr}.\mathsf{tr}'} (\mathcal{Q}; \Phi \cup \Psi)$), and,*
- *invariant by permutations of actions (*i.e. *if $A \xrightarrow{\mathsf{tr}} B$ satisfies the predicate then so does any execution $A \xrightarrow{\mathsf{tr}'} B$ as long as $\mathsf{tr}'$ equals $\mathsf{tr}$ up to a permutation of actions).*

*For instance, secrecy is monotone and invariant by permutations.*

*While this is already useful to reduce the search space, we will go much further in the following sections. Most importantly, our final goal is to go beyond reachability properties, and to deal with trace equivalence, which is addressed in the next subsection.*

### 5.3.4 Equivalence

We now define the compressed trace equivalence ($\approx_c$) and prove that it coincides with the regular trace equivalence ($\approx$).

**Definition 23.** *Let $A$ and $B$ be two configurations. We say that $A \sqsubseteq_c B$ when, for any $A \xrightarrow{\mathsf{tr}}_c A'$ such that $bc(\mathsf{tr}) \cap fc(B) = \emptyset$, there exists $B \xrightarrow{\mathsf{tr}}_c B'$ such that $\Phi(A') \sim \Phi(B')$. They are* compressed trace equivalent*, denoted $A \approx_c B$, if $A \sqsubseteq_c B$ and $B \sqsubseteq_c A$.*

Compressed trace equivalence can be more efficiently checked than regular trace equivalence. Obviously, it explores fewer interleavings by relying on $\rightarrow_c$ rather than $\rightarrow$. It also requires that traces of one process can be played exactly by the other, including details such as unobservable actions, labels, and focusing annotations. The subtleties shown in Example 23 are crucial for the completeness of compressed equivalence w.r.t. regular equivalence. Since the compressed semantics forces to perform available outputs before *e.g.* input actions, some non-equivalences are only detected thanks to the labels and detailed unobservable actions of our annotated semantics (as illustrated by Example 23).

**Theorem 2.** *Let $A$ and $B$ be two action-deterministic configurations with $\mathsf{skl}(A) = \mathsf{skl}(B)$. We have $A \approx B$ if, and only if, $\lceil A \rceil \approx_c \lceil B \rceil$.*

*Proof sketch, complete proof in Subsection 5.3.5.* ($\Rightarrow$) Consider an execution $\lceil A \rceil \xrightarrow{\mathsf{tr}}_c A'$ where $\mathsf{tr}$ is a proper trace (the other case is similar). Using Lemma 3, we get $A \xrightarrow{\lfloor \mathsf{tr} \rfloor}_a \lfloor A' \rfloor$. Then, Lemma 2 yields $B \xrightarrow{\lfloor \mathsf{tr} \rfloor}_a B'$ for some $B'$ such that $\Phi(\lfloor A' \rfloor) \sim \Phi(B')$ and labelled skeletons are

equal all along the executions. Relying on those skeletons, one can show that positive/negative phases are synchronised, and thus $\lceil B \rceil \xrightarrow{\mathsf{tr}}_c B''$ for some $B''$ with $\lfloor B'' \rfloor = B'$.
($\Leftarrow$) We first deal with proper executions. Consider an execution $A \xrightarrow{\mathsf{tr}}_a A'$ where $\mathsf{tr}$ is a proper trace. We first observe that it suffices to consider only complete executions there. This allows us to get a compressed execution $\lceil A \rceil \xrightarrow{\mathsf{tr}_c}_c \lceil A' \rceil$ by Lemma 4. Since $\lceil A \rceil \approx_c \lceil B \rceil$, there exists $B'$ such that $\lceil B \rceil \xrightarrow{\mathsf{tr}_c}_c B'$ with $\Phi(\lceil A' \rceil) \sim \Phi(B')$. Thus we have $B \xrightarrow{\lfloor \mathsf{tr}_c \rfloor}_a \lfloor B' \rfloor$ but also $B \xrightarrow{\mathsf{tr}}_a \lfloor B' \rfloor$ thanks to Lemma 1.

When $\mathsf{tr}$ is not proper, we invoke Lemma 5 and obtain executions $\lceil A \rceil \xrightarrow{\mathsf{tr}_c}_c A_0 \xrightarrow{b_i}_c A_i$ for $1 \leq i \leq k$. Following similar arguments as above, we obtain executions $B \xrightarrow{\mathsf{tr}_c}_a B_0 \xrightarrow{b_i}_a B_i$ for $1 \leq i \leq k$ such that $\Phi(B_i) = \Phi(B_0) \sim \Phi(A_0) = \Phi(A_i)$. We then exploit independencies of blocks $b_i$ to construct $B \xrightarrow{\mathsf{tr}_c}_a B_0 \xrightarrow{b_1 \ldots . b_k}_a B'$ with $\Phi(B') = \Phi(B_0)$ and then $B \xrightarrow{\mathsf{tr}}_a B'$. □

**Remark 7.** *Note that the trace inclusions of the regular semantics (*i.e. $\sqsubseteq$*) and the compressed semantics (*i.e. $\sqsubseteq_c$*) do not coincide as witnessed by the following example. Let* $\mathcal{P} = \{\mathtt{in}(c, x)\}$ *and* $\mathcal{Q} = \{\mathtt{in}(c, x).\mathtt{out}(c, n)\}$ *accompanied with an arbitrary frame* $\Phi$*. We have* $(\mathcal{P}; \Phi) \sqsubseteq (\mathcal{Q}; \Phi)$ *but* $(\mathcal{P}; \Phi) \not\sqsubseteq_c (\mathcal{Q}; \Phi)$ *since* $(\mathcal{Q}; \Phi)$ *is not able to produce the trace* $\mathtt{foc}(\mathtt{in}(c, M)).\mathtt{rel}.\mathtt{zero}$ *in the compressed semantics. But we have* $(\mathcal{Q}; \Phi) \not\sqsubseteq (\mathcal{P}; \Phi)$ *and thus* $(\mathcal{P}; \Phi) \not\approx (\mathcal{Q}; \Phi)$*.*

### 5.3.5 Proof of Theorem 2

We prove below the two implications of Theorem 2, dealing first with soundness and then with the more involved completeness result.

Soundness can be established for both the semi-compressed and the compressed trace equivalence. Even though the semi-compressed semantics is weaker that the compressed semantics, we still introduce the semi-compressed trace equivalence and prove that it coincides with the regular trace equivalence for future proofs. Indeed, for the compressed (and the reduced semantics we eventually define in Section 5.4), we prove the completeness of the optimised trace equivalence w.r.t. regular trace equivalence in two steps: first the semi-optimised w.r.t. the regular one and then the optimised one w.r.t. the semi-optimised one.

**Definition 24.** *Let* $A$ *and* $B$ *be two configurations. We say that* $A \sqsubseteq_{sc} B$ *when, for any* $A \xrightarrow{\mathsf{tr}}_{sc} A'$ *such that* $bc(\mathsf{tr}) \cap fc(B) = \varnothing$*, there exists* $B \xrightarrow{\mathsf{tr}}_{sc} B'$ *such that* $\Phi(A') \sim \Phi(B')$*. They are* semi-compressed trace equivalent*, denoted* $A \approx_{sc} B$*, if* $A \sqsubseteq_{sc} B$ *and* $B \sqsubseteq_{sc} A$*.*

**Lemma 6** (Soundness)**.** *Let* $A$ *and* $B$ *be action-deterministic configurations such that* $\mathsf{skl}(A) = \mathsf{skl}(B)$*. We have that* $A \approx B$ *implies* $\lceil A \rceil \approx_c \lceil B \rceil$ *(and* $\lceil A \rceil \approx_{sc} \lceil B \rceil$ *as well).*

*Proof.* We only deal with the compressed trace equivalence, the case of the semi-compressed trace equivalence is identical. By symmetry it suffices to show $\lceil A \rceil \sqsubseteq_c \lceil B \rceil$. Consider an execution $\lceil A \rceil \xrightarrow{\mathsf{tr}}_c A'$ such that $bc(\mathsf{tr}) \cap fc(B) = \emptyset$. Thanks to Lemma 3, we know that $A \xrightarrow{\lfloor \mathsf{tr} \rfloor}_a K_A$ with $\Phi(K_A) = \Phi(A')$. Let $\lfloor \mathsf{tr} \rfloor = [\alpha_1]^{\ell_1} \ldots [\alpha_n]^{\ell_n}$, and we denote $A_1, \ldots, A_n$ the intermediate configurations that are reached during this execution. We have that:

$$A_0 = A \xrightarrow{[\alpha_1]^{\ell_1}}_a A_1 \xrightarrow{[\alpha_2]^{\ell_2}}_a \ldots \xrightarrow{[\alpha_n]^{\ell_n}}_a A_n = K_A = (\mathcal{P}; \Phi_A).$$

Applying Lemma 2, we deduce that $B$ can perform a very similar execution (same labels, same actions), *i.e.*

$$B_0 = B \xrightarrow{[\alpha_1]^{\ell_1}}_a B_1 \xrightarrow{[\alpha_2]^{\ell_2}}_a \dots \xrightarrow{[\alpha_n]^{\ell_n}}_a B_n = (\mathcal{Q}; \Phi_B).$$

with $\Phi(A_i) \sim \Phi(B_i)$ and $\mathsf{skl}(A_i) = \mathsf{skl}(B_i)$ for $0 \leq i \leq n$.

As a consequence of the equalities $\mathsf{skl}(A_i) = \mathsf{skl}(B_i)$ and the fact that $B$ produces the same labelled actions as $A$ does, we are sure that $\lceil B \rceil$ will be able to do this execution in the compressed semantics as well. In particular, the fact that a given configuration $B_i$ can start a positive phase or has to release the focus is determined by the set $\mathsf{skl}(B_i) = \mathsf{skl}(A_i)$ and the actions $[\alpha_1]^{\ell_1}, \dots, [\alpha_{i-1}]^{\ell_{i-1}}$ and the fact that it can keep the focus on a specific process while performing positive actions can be deduced from labels of $\mathsf{tr}$. Finally, we have shown that if $A_i$ can execute an action $\alpha$ using Neg rule then $B_i$ can as well. The only missing part is about the fact that Neg has been made non-branching using an arbitrary order on labelled skeletons. Let us say we can use Neg only for actions whose skeleton is minimal among others skeletons of available, negative actions. Using $\mathsf{skl}(A_i) = \mathsf{skl}(B_i)$, we easily show that this is symmetric for $A_i$ and $B_i$. This way, we obtain an execution $\lceil B \rceil \xrightarrow{\mathsf{tr}}_c B'$ with $\Phi(B') = \Phi(B_n)$ (we do not necessarily have $\lfloor B' \rfloor = B_n$ because $\mathsf{tr}$ may end with an improper block). Finally, we have $\Phi(B') = \Phi_B \sim \Phi_A = \Phi(A')$. □

One key ingredient of the completeness is the fact that exploring complete executions only is actually sufficient to establish annotated trace equivalence.

**Lemma 7.** *Let $A$ and $B$ be two action-deterministic configurations. If for any complete execution $A \xrightarrow{\mathsf{tr}}_a A'$ with $bc(\mathsf{tr}) \cap fc(B) = \emptyset$, there exists a trace $\mathsf{tr}'$ and an execution $B \xrightarrow{\mathsf{tr}'}_a B'$ such that $\Phi(A') \sim \Phi(B')$, then $A \sqsubseteq_a B$.*

*Proof.* Let $A \xrightarrow{\mathsf{tr}_0}_a A_0$ be an execution of $A$ with $bc(\mathsf{tr}_0) \cap fc(B) = \emptyset$. Firstly, we complete the latter execution in an arbitrary way $A \xrightarrow{\mathsf{tr}_0.\mathsf{tr}_1}_a A'$ such that any process of $A'$ is replicated and $bc(\mathsf{tr}_0.\mathsf{tr}_1) \cap fc(B) = \emptyset$ (it suffices to choose fresh channels for $B$ as well). By hypothesis, there exists an execution $B \xrightarrow{\mathsf{tr}_0.\mathsf{tr}_1}_a B'$ such that $\Phi(A') \sim \Phi(B')$. The latter execution of $B$ is thus of the form $B \xrightarrow{\mathsf{tr}_0}_a B_0 \xrightarrow{\mathsf{tr}_1}_a B'$. It remains to show that $\Phi(A_0) \sim \Phi(B_0)$. For the sake of contradiction, we assume that $\Phi(B_0) \not\sim \Phi(A_0)$. In other words, there is a test of equality or a computation test over $\mathrm{dom}(\Phi(B_0))$ that holds for $\Phi(A_0)$ and does not for $\Phi(B_0)$ (or the converse). Since $\mathrm{dom}(\Phi(B_0)) \subseteq \mathrm{dom}(\Phi(B')) = \mathrm{dom}(\Phi(A'))$, this same test can be used to conclude that $\Phi(A') \not\sim \Phi(B')$ leading to a contradiction. □

Finally, we prove the completeness of the compressed semantics w.r.t. trace equivalence in two steps. First (Lemma 8), we show the completeness for the semi-compressed trace equivalence defined above (see Definition 24). Second (Corollary 3), we show that both the full and the semi-compressed trace equivalence coincide.

**Lemma 8.** *Let $A$ and $B$ be two action-deterministic configurations satisfying $\mathsf{skl}(A) = \mathsf{skl}(B)$. Then $\lceil A \rceil \approx_{sc} \lceil B \rceil$ implies $A \approx B$.*

*Proof.* Assume $\lceil A \rceil \approx_{sc} \lceil B \rceil$, thanks to Theorem 1, it suffices to show $A \approx_a B$. Let us show the following intermediate result: *for any complete execution* $A \xrightarrow{\mathsf{tr}}_a A'$ *such that* $bc(\mathsf{tr}) \cap fc(B) = \emptyset$, *there is an execution* $B \xrightarrow{\mathsf{tr}}_a B'$ *such that* $\Phi(A') \sim \Phi(B')$. Thanks to Lemma 7 and by symmetry of $\approx_a$, this intermediate result implies the required conclusion $A \approx_a B$.

Let $A \xrightarrow{\mathsf{tr}}_a A'$ be a complete execution with $bc(\mathsf{tr}) \cap fc(B) = \emptyset$. We thus have that $A'$ is initial. Applying the Lemma 4, we obtain a trace $\mathsf{tr}_c$ such that $\lceil A \rceil \xrightarrow{\mathsf{tr}_c}_{sc} \lceil A' \rceil$ and $\lfloor \mathsf{tr}_c \rfloor$ can be obtained from $\mathsf{tr}$ by swapping independent actions. Since we have $\lceil A \rceil \approx_{sc} \lceil B \rceil$, we deduce that $\lceil B \rceil \xrightarrow{\mathsf{tr}_c}_{sc} \lceil B' \rceil$ with $\Phi(A') \sim \Phi(B')$. Lemma 3 gives us $B \xrightarrow{\lfloor \mathsf{tr}_c \rfloor}_a B'$. We can now apply Lemma 1 to obtain $B \xrightarrow{\mathsf{tr}}_a B'$ and conclude.

□

**Corollary 2.** *Let $A$ and $B$ be two action-deterministic configurations with $\mathsf{skl}(A) = \mathsf{skl}(B)$. We have $A \approx B$ if, and only if, $\lceil A \rceil \approx_{sc} \lceil B \rceil$.*

*Proof.* It follows from lemmas 6 and 8 □

**Lemma 9.** *Let $A$ and $B$ be two action-deterministic configurations. Then $\lceil A \rceil \sqsubseteq_c \lceil B \rceil$ implies $\lceil A \rceil \sqsubseteq_{sc} \lceil B \rceil$.*

*Proof.* Assume that $A \sqsubseteq_c B$. Let $A'$ be such that $A \xrightarrow{\mathsf{tr}}_{sc} A'$ for some $\mathsf{tr}$ such that $bc(\mathsf{tr}) \cap fc(B) = \emptyset$. Let $b_1, \ldots, b_k$ be the improper blocks that occur in $\mathsf{tr}$. We argue that all those blocks are pairwise independent and each block $b_i$ is independent with all actions occurring after $b_i$ in $\mathsf{tr}'$. Both claims follow from (i) the absence of recipe dependencies since blocks $b_i$ do not feature outputs and (ii) the fact that each block $b_i$ gives rise to a null process that disappears from the multiset along with the label it carried out preventing sequential dependencies.

Hence a proper trace $\mathsf{tr}_c$ such that $\mathsf{tr}_c.b_1.\ldots.b_k$ can be obtained from $\mathsf{tr}$ by swapping independent blocks. Those swaps can also be made along the former semi-compressed execution since before starting a new block, the semi-compressed semantics allows to choose any positive process to start a focus. We thus have an execution $A \xrightarrow{\mathsf{tr}_c.b_1.\ldots.b_k}_{sc} \lceil A' \rceil$.

Since blocks $b_i$ are pairwise independent, there exist initial configurations $A_0, A_1, \ldots, A_k$ such that $A \xrightarrow{\mathsf{tr}_c}_{sc} A_0$, and

$$A_0 \xrightarrow{b_1}_{sc} A_1,\ A_0 \xrightarrow{b_2}_{sc} A_2,\ \ldots, A_0 \xrightarrow{b_k}_{sc} A_k.$$

Since $\mathsf{tr}_c$ is a proper trace, the execution $\lceil A \rceil \xrightarrow{\mathsf{tr}_c}_{sc} A_0$ can easily be translated into a compressed execution $\lceil A \rceil \xrightarrow{\mathsf{tr}_c}_c A_0$ using rule Release instead of Release$^s$. Finally, remark that all executions $A_0 \xrightarrow{b_i}_{sc} A_i$ can be done using the compressed execution instead of the semi-compressed execution: it suffices to trigger rule Release$_\perp$ instead of Release$^s$. We thus obtain executions $A_0 \xrightarrow{b_i}_c (\emptyset; \varnothing; \Phi(A_i))$ for all $1 \leq i \leq k$.

Thanks to our hypothesis, we deduce that there exist configurations $B_0$, $B_1, \ldots, B_k$ such that $B \xrightarrow{\mathsf{tr}_c}_c B_0$ with $\Phi(A_0) \sim \Phi(B_0)$, and

$$B \xrightarrow{\mathsf{tr}_c.b_1}_c B_1,\ B \xrightarrow{\mathsf{tr}_c.b_2}_c B_2,\ \ldots, B_0 \xrightarrow{\mathsf{tr}_c.b_k}_c B_k.$$

Those executions can now be mapped back to semi-compressed executions: there exists $B \xrightarrow{\mathsf{tr}_c}_{sc} B_0$ (because $\mathsf{tr}_c$ is proper) and there exist $B \xrightarrow{\mathsf{tr}_c.b_i}_{sc} B'_i$ with $\Phi(B'_i) = \Phi(B_i) = \Phi(B_0)$ (because the block $b_i$ is improper). Since blocks $b_1, \ldots, b_k$ are pairwise independent, we deduce that there exists $B'$ such that $B \xrightarrow{\mathsf{tr}_c}_{sc} B_0 \xrightarrow{b_1,\ldots,b_k}_{c} B'$ with $\Phi(B') = \Phi(B_0)$. Then, permutations of blocks can be undone to retrieve $\mathsf{tr}$ (since swaps have been done between independent blocks). □

Finally, combining lemmas 8 and 9 yields the following corollary.

**Corollary 3** (Completeness)**.** *Let $A$ and $B$ be two action-deterministic configurations satisfying $\mathsf{skl}(A) = \mathsf{skl}(B)$. Then $\lceil A \rceil \approx_c \lceil B \rceil$ implies $A \approx B$.*

Theorem 2 follows from Corollary 3 and Lemma 6.

## 5.4 Reduction

The compressed semantics cuts down interleavings by using a simple focused strategy. However, this semantics does not analyse data dependency that happens when an input depends on an output, and is thus unable to exploit the independency of blocks to reduce interleavings as partly illustrated in the next example. We tackle this problem with the reduced semantics which we define in this section.

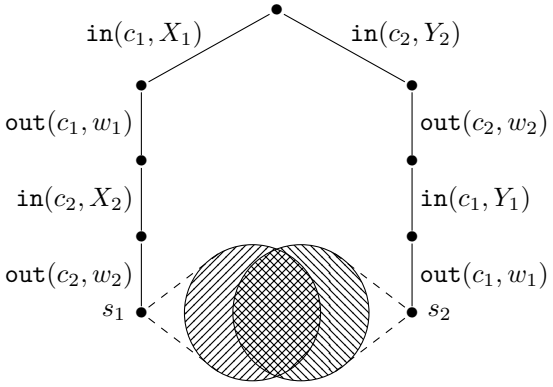


**Figure 5.4** Resuming Example 27, we informally depict two possible interleavings abstracting away recipes (*i.e.* $X_1, X_2, Y_1, Y_2$). A node in this tree represents all reachable states (from $A$ using the compressed semantics) that can be reached using the corresponding interleaving of actions and choosing recipes for $X_1, X_2$ (or $Y_1, Y_2$). We represent two of those sets of states (*i.e.* $s_1$ and $s_2$) by the hatch circles. The intersection of those circles illustrates to set of reachable states that can be reached by both interleavings.

**Example 27.** *For $i \in \{1, 2\}$, let $P_i = \mathtt{in}(c_i, x_i).\mathtt{out}(c_i, n_i).P'_i$ for some name $n_i \in \mathcal{N}$ and positive process $P'_i$. We consider the configuration $A = (\{P_1, P_2\}; \emptyset)$ and two types of interleavings of actions that the compressed semantics may explore. Essentially, the compressed semantics has two choices: either it starts to focus on process $P_1$ producing a block $b_1 = \mathtt{foc}(\mathtt{in}(c_1, M_1)).\mathtt{rel}.\mathtt{out}(c_1, w_1)$ for some recipe $M_1$ or on process $P_2$ producing a block $b_2 =$*

$\mathtt{foc}(\mathtt{in}(c_2, M_2)).\mathtt{rel}.\mathtt{out}(c_2, w_2)$ *for some recipe* $M_2$*. The compressed semantics explores both traces* $\mathsf{tr}_1 = b_1.b_2$ *and* $\mathsf{tr}_2 = b_2.b_1$*.*

*One could argue at this point that exploring those two traces may be redundant. For instance, if* $M_1 = M_2 = \mathsf{ok} \in \Sigma_c$ *(*i.e. *a constant), then both traces essentially lead to the same state and are thus redundant. Further, it seems that one can choose a priority order (*e.g. $P_1$ *has priority over* $P_2$*) and, when the compressed semantics has to choose a focus, it has to choose the process with highest priority. However, when recipes depend on previous blocks (*e.g. $M_1 = w_2$ *in* $\mathsf{tr}_2$*), both traces do not lead to the same state and must be explored separately. A diagram informally illustrating those two interleavings, the sets of reachable configurations following one or the other interleaving and the overlap (*i.e. *reachable states that can be reached using both interleavings) is depicted in Figure 5.4.*

*This is the crux of the problem: some interleavings explored by the compressed semantics are mostly redundant but they cannot be simply discarded because some specific traces following those interleavings actually exploit the specific ordering that violates the pre-defined priority order. Moreover, distinguishing those traces from redundant ones calls for a notion of* necessity *based on (strong) data-dependencies: "Is it the case that executing block* $b_2$ *before* $b_1$ *is* really necessary *to be able to execute block* $b_1$ *in a given way ?". Intuitively, the* necessity *notion expresses the necessity (for the environment) to execute a certain block after a certain trace for not losing completeness;* i.e. *still exploring all reachable states.*

### 5.4.1 Strong Independence

Before formally defining our reduction technique, we characterise the kind of independencies over which we seek to eliminate redundancy in the compressed semantics.

**Definition 25.** *Two blocks* $b_1$ *and* $b_2$ *are* sequentially independent *(resp.* recipe independent*), written* $b_1 \parallel^s b_2$ *(resp.* $b_1 \parallel^r b_2$*), when all labelled actions* $\alpha_1 \in b_1$ *and* $\alpha_2 \in b_2$ *are sequentially independent (resp. recipe independent).*

*They are* independent*, written* $b_1 \parallel b_2$ *when* $b_1 \parallel^s b_2$ *and* $b_1 \parallel^r b_2$*. Otherwise they are* dependent*, written* $b_1 \rightleftharpoons b_2$*.*

Obviously, Lemma 1 tells us that independent blocks can be permuted in a trace without affecting the executability and the result of executing that trace. But this notion is not very strong and does not well match the notion of *necessity* we need since it considers fixed recipes, which are irrelevant (in the end, only the derived messages matter) and can easily introduce spurious dependencies.

**Example 28** (Resuming Example 27)**.** *Let* $M_1 = \pi_1(\langle \mathsf{ok}, w_2 \rangle)$ *in the block* $b_1 = \mathtt{foc}(\mathtt{in}(c_1, M_1)).\mathtt{rel}.\mathtt{out}(c_1, w_1)$ *and* $M_2 = \mathsf{ok}$ *in the block* $b_2 = \mathtt{foc}(\mathtt{in}(c_2, M_2)).\mathtt{rel}.\mathtt{out}(c_2, w_2)$*, we obtain two blocks that are dependent (*i.e. $b_1 \rightleftharpoons b_2$*) because* $M_1$ *uses* $w_2$*. However, it is clearly not a necessity to execute one block before the other. Indeed, it suffices to replace* $M_1$ *by* $M_1' = \mathsf{ok}$ *yielding same messages and execution but also removing dependencies (the induced block* $b_1' = \mathtt{foc}(\mathtt{in}(c_1, \mathsf{ok})).\mathtt{rel}.\mathtt{out}(c_1, w_1)$ *satisfies* $b_1' \parallel b_2$*).*

*We now consider* $A' = (\{P_1, P_2\}; \Phi_0)$ *where* $\Phi_0 = \{w \mapsto \langle n_1, n_2 \rangle\}$. *Intuitively, we give to the attacker the two names* $n_1, n_2$ *that* $P_1$ *and* $P_2$ *may output. Let* $M_3 = w_2$ *in the block* $b_3 = \mathtt{foc}(\mathtt{in}(c_1, M_3)).\mathtt{rel}.\mathtt{out}(c_1, w_1)$ *and* $M_4 = \mathsf{ok}$ *in the block* $b_4 = \mathtt{foc}(\mathtt{in}(c_2, M_4)).\mathtt{rel}.\mathtt{out}(c_2, w_2)$, *we obtain two blocks that are also dependent (*i.e. $b_3 \rightleftharpoons b_4$*). However, it is not a necessity to execute one block before the other since one can replace* $M_3$ *by* $M_3' = \pi_2(w)$ *whilst preserving executability and resulting frame. As for the previous example, replacing* $M_3$ *by* $M_3'$ *also makes the previous dependency disappear.*

This calls for a stronger notion of equivalence over traces, which allows permutations of independent blocks but also changes of recipes that preserve messages. During these permutations, we will also require that traces remain *plausible*, which is defined as follows: $\mathsf{tr}$ is plausible if for any input $\mathtt{in}(c, M)$ such that $\mathsf{tr} = \mathsf{tr}_0.\mathtt{in}(c, M).\mathsf{tr}_2$ then $M \in \mathcal{T}(\Sigma_{\mathsf{pub}}, \mathcal{W}_0)$ where $\mathcal{W}_0$ is the set of all handles occurring in $\mathsf{tr}_0$. Given a block $b$, *i.e.* a sequence of the form $\mathtt{foc}(\alpha).\mathsf{tr}^+.\mathtt{rel}.\mathsf{tr}^-$, we denote by $b^+$ (resp. $b^-$) the part of $b$ corresponding to the positive (resp. negative) phase, *i.e.* $b^+ = \alpha.\mathsf{tr}^+$ (resp. $b^- = \mathsf{tr}^-$). We note $(b_1 =_{\mathsf{E}} b_2)\Phi$ when $b_1^+\Phi =_{\mathsf{E}} b_2^+\Phi$ and $b_1^- = b_2^-$.

**Definition 26.** *Given a frame* $\Phi$, *the relation* $\equiv_\Phi$ *is the smallest equivalence over plausible compressed traces such that* $\mathsf{tr}.b_1.b_2.\mathsf{tr}' \equiv_\Phi \mathsf{tr}.b_2.b_1.\mathsf{tr}'$ *when* $b_1 \parallel b_2$, *and* $\mathsf{tr}.b_1.\mathsf{tr}' \equiv_\Phi \mathsf{tr}.b_2.\mathsf{tr}'$ *when* $(b_1 =_{\mathsf{E}} b_2)\Phi$.

**Example 29** (Resuming Example 28)**.** *Consider a compressed execution of the trace* $b_2.b_1$ *(where* $\Phi = \{w_1 \mapsto n_1, w_2 \mapsto n_2\}$*):*

$$A \xrightarrow{b_2.b_1}_c (\{P_1'\{x_1 \mapsto \mathsf{ok}\}, P_2'\{x_2 \mapsto \mathsf{ok}\}\}; \Phi)$$

*The two blocks we defined were dependent (*i.e. $b_1 \rightleftharpoons b_2$*) but* $b_2.b_1 \equiv_\Phi b_2.b_1' \equiv_\Phi b_1'.b_2$ *witnessing the fact that* $b_1$ *and* $b_2$ *can actually be swapped and no block is actually necessary for the other to be executed. The traces* $b_1.b_2$ *and* $b_2.b_1$ *are thus redundant and exploring only one of them should be sufficient.*

Along the same lines as the proof of Lemma 1, we prove the following result.

**Lemma 10.** *Let* $A$ *and* $A'$ *be two initial configurations such that* $A \xrightarrow{\mathsf{tr}}_{sc} A'$. *We have that* $A \xrightarrow{\mathsf{tr}'}_{sc} A'$ *for any* $\mathsf{tr}' \equiv_{\Phi(A')} \mathsf{tr}$.

*Proof.* Thanks to Lemma 3, we have that $\lfloor A \rfloor \xrightarrow{|\mathsf{tr}|}_a (\mathcal{P}; \Phi)$. We first prove that $\mathsf{tr}'$ can be performed using $\rightarrow_a$. For this, it suffices to establish the implication for each of the two generators of $\equiv_\Phi$. The first case is given by Lemma 1. The second one is a common property of (derivatives of) the applied $\pi$-calculus that follows from a simple observation of the transition rules; *i.e.* executions rules and computations ($\Downarrow$) are insensitive to the choice of representative in $=_{\mathsf{E}}$-equivalence classes. Finally, we must prove that $\mathsf{tr}'$ can be played using $\rightarrow_{sc}$. Thanks to initiality of $A$ and $(\mathcal{P}; \varnothing; \Phi)$ we know that each block of $\mathsf{tr}$ starts when the configuration is initial and after performing it we get another initial configuration. This is still true in $\lfloor A \rfloor \xrightarrow{|\mathsf{tr}'|}_a (\mathcal{P}; \Phi)$. Finally, labels of blocks of $\mathsf{tr}'$ ensure that a single process is used in a positive part of any block.

Having proved those two facts, we can easily show that each block of $\mathsf{tr}'$ can be performed using $\rightarrow_{sc}$. □

Note that a similar lemma based on the compressed semantics (instead of the semi-compressed semantics) would not hold since $\equiv_\Phi$ notably allows to swap an improper block that was at the end to another position whereas the compressed semantics allows improper blocks to be executed at the end only.

### 5.4.2 Priority Order And Necessity

We now turn to defining our reduced semantics, which will achieve the elimination of redundancies identified above by only executing specific representatives in equivalence classes modulo $\equiv_\Phi$. More precisely, we shall only execute minimal traces according to some order, which we introduce next.

**Definition 27.** *We consider an arbitrary order $\prec$ on blocks that is insensitive to recipes and handles, and such that independent blocks are always strictly ordered in one way or the other. We finally define $\prec_{\mathsf{lex}}$ on compressed traces as the lexicographic extension of $\prec$ on blocks.*

In order to incrementally build representatives that are minimal with respect to $\prec_{\mathsf{lex}}$, we define a predicate that expresses whether a block $b$ should be *authorised* after a given trace $\mathsf{tr}$. Intuitively, this is the case only when, for any block $b' \succ b$ in $\mathsf{tr}$ (that is to say, executing $b$ would violate the priority order), dependencies forbid to move $b$ before $b'$ (meaning that violating the priority order is a *necessity*). We define this with recipe dependencies first, then quantify over all recipes to capture message dependencies.

**Definition 28** (Weak and Strong Authorization)**.** *A block $b$ is* weakly authorised *after $\mathsf{tr}$, noted $\mathsf{tr} \triangleright b$, when $\mathsf{tr} = \epsilon$; or $\mathsf{tr} = \mathsf{tr}_0.b_0$ and either (i) $b \rightleftharpoons b_0$ or (ii) $b \parallel b_0$, $b_0 \prec b$, and $\mathsf{tr}_0 \triangleright b$.*
*A block $b$ is* strongly authorised *after $\mathsf{tr}, \Phi$, noted $(\mathsf{tr}, \Phi) \blacktriangleright b$, when $\mathsf{tr} \triangleright b'$ for all $b'$ satisfying $(b' =_{\mathsf{E}} b)\Phi$.*

The *strong authorised* predicate is the formal counterpart of the intuitive notion of *necessity* we sketched before.

**Example 30** (Resuming Example 29)**.** *Assuming that $b_1 \prec b_2$, one has $(b_1, \Phi) \blacktriangleright b_2$ but $(b_2, \Phi) \not\blacktriangleright b_1$ because $b_2 \not\sim b_1'$ and $(b_1 =_{\mathsf{E}} b_1')\Phi$. In other words, $b_2.b_1$ is considered redundant and will not be explored by the reduced semantics since $b_1.b_2$ has priority over the other interleaving and leads to the same state.*

### 5.4.3 Reduced Semantics

We finally define $\rightarrow_r$ as the least relation such that:

$$\text{Init}\ \frac{}{A \xrightarrow{\epsilon}_r A} \qquad \text{Block}\ \frac{A \xrightarrow{\mathsf{tr}}_r (\mathcal{P}; \varnothing; \Phi) \quad (\mathcal{P}; \varnothing; \Phi) \xrightarrow{b}_c A'}{A \xrightarrow{\mathsf{tr}.b}_r A'}\ \text{if } (\mathsf{tr}, \Phi) \blacktriangleright b$$

Our reduced semantics only applies to initial configurations: otherwise, no block can be performed. This is not restrictive since we can, without loss of generality, pre-execute unobservable and output actions that may occur at top level.

Similarly to compression, we also introduce the *semi-reduced semantics* noted $\xrightarrow{\bullet}_{sr}$ that is defined as the reduced semantics but based on the semi-compressed semantics $\xrightarrow{\bullet}_{sc}$ instead of the compressed semantics $\xrightarrow{\bullet}_{c}$.

**Example 31** (Resuming Example 27)**.** *We now consider the configuration* $A = (\{P_1, P_2, P_3\}; \emptyset)$ *(remind that* $P_i = \mathtt{in}(c_i, x_i).\mathtt{out}(c_i, n_i).P_i'$ *and* $P_i$*'s are positive). We assume that blocks using only channel* $c_i$ *are smaller w.r.t.* $\prec$ *than blocks using only channel* $c_j$ *when* $j > i$*. We now illustrate and characterise the redundant traces that are not explored by the reduced semantics but that are explored by the (semi-)compressed semantics. To conduct this analysis, we shall group traces together depending on the interleaving of the first blocks of* $P_1, P_2, P_3$ *they follow. We note those blocks* $b_1, b_2, b_3$ *where* $b_i = \mathtt{foc}(\mathtt{in}(c_i, M_i)).\mathtt{rel}.\mathtt{out}(c_i, w_i)$*. There are 6 possible interleavings of those 3 blocks that are depicted in Figure 5.5. Note that many different traces can follow a given single interleaving since many different recipes can be chosen for* $M_1, M_2$ *and* $M_3$*. Moreover, all those traces will be explored by the (semi-)compressed semantics. We now illustrate that, thanks to the authorised predicate* $(\bullet, \bullet) \blacktriangleright \bullet$*, a strict subset of those traces are actually explored by the reduced semantics.*

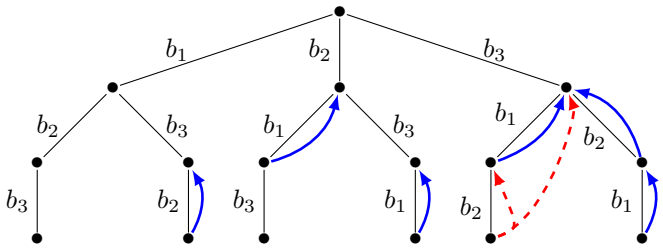


**Figure 5.5** Interleavings with 3 processes in parallel

*We informally depict the* necessity relation *by blue and red, dashed arrows. For instance, traces following the interleaving* $b_1.b_3.b_2$ *will be explored by the reduced semantics only if* $b_3$ *is necessary to* $b_2$*,* i.e. *for any* $(b_2' =_{\mathsf{E}} b_2)\Phi$ *where* $\Phi$ *is the resulting frame, one has* $b_2' \rightleftharpoons b_3$*. Indeed,* $(b_1.b_3, \Phi) \blacktriangleright b_2$ *implies that for any* $(b_2' =_{\mathsf{E}} b_2)\Phi$*, one has* $b_1.b_3 \triangleright b_2'$*. But since* $b_2 \prec b_3$ *and so* $b_2' \prec b_3$*, the latter entails* $b_2' \rightleftharpoons b_3$*. We represent this necessary condition by the left-most arrow (in blue). Note the many other arrows (in blue) corresponding to similar conditions.*

*Now, on the interleaving* $b_3.b_1.b_2$*, the authorised predicate* $(b_3.b_1, \Phi) \blacktriangleright b_2$ *requires that either* $b_1$ *or* $b_3$ *is necessary to* $b_2$ *represented by the dashed, red* 2*-arrow. Intuitively, the latter condition expresses that* $b_2$ *is only allowed to come after* $b_3$ *if it depends on it, possibly indirectly through* $b_1$ *(note that* $b_1$ *should also depends on* $b_3$ *as illustrated by the blue arrow).*

*Overall, the reduced semantics imposes many such conditions and does not explore the many traces violating them.*

**Example 32.** *We consider processes* $R_i := \mathtt{in}(c_i, x).\mathsf{if}\ x = \mathsf{ok}\ \mathsf{then}\ \mathtt{out}(c_i, \mathsf{ok})$ *where* ok *is a public constant, and then consider a parallel composition of* $n$ *such processes:* $P_n := \Pi\{R_i\}^{\#}_{1\leq i\leq n}$. *Thanks to compression, we will only consider traces made of blocks, and obtain a first exponential reduction of the state space. However, contrary to the case of a replicated process (see Example 25 in Subsection 5.3.1), we still have many interleavings to consider – blocks can be interleaved in all the possible ways. Our reduced semantics also cuts down these interleavings on this example. Assume that our order* $\prec$ *prioritises blocks on* $c_i$ *over those on* $c_j$ *when* $i < j$, *and consider a trace starting with* $\mathtt{foc}(\mathtt{in}(c_j, M_j)).\mathtt{rel}.\mathtt{out}(c_j, w_j)$. *Trying to continue the exploration with a block on* $c_i$ *with* $i < j$, *the authorisation predicate* $(\bullet, \bullet) \blacktriangleright \bullet$ *will impose that there is a dependency between the block on* $c_i$ *and the previous one on* $c_j$. *In this case it must be a data dependency: the recipe of the message passed as input on* $c_i$ *must make use of the previous output to derive* ok. *Since* ok *is a public constant, it is possible to derive it without using any previous output and thus the block on* $c_i$ *cannot be authorised by* $(\bullet, \bullet) \blacktriangleright \bullet$. *Thus, on this simple example, the reduced semantics will not explore any trace where a block on* $c_i$ *is performed after one on* $c_j$ *with* $i < j$.

*If one forgets about branching due to choices of recipes, there are* $(2n)!/2^n$ *different interleavings of size* $2n$ *(i.e., containing* $2n$ *observable actions) in the regular semantics. In the compressed semantics this number goes down to* $n!$. *Finally, in the reduced semantics, there is only one trace of size* $2n$.

### 5.4.4 Reachability

An easy induction on the compressed trace $\mathsf{tr}$ allows us to map an execution w.r.t. the reduced semantics to an execution w.r.t. the compressed semantics.

**Lemma 11.** *For any configurations* $A$ *and* $A'$, $A \xrightarrow{\mathsf{tr}}_r A'$ *implies* $A \xrightarrow{\mathsf{tr}}_c A'$ *and* $A \xrightarrow{\mathsf{tr}}_{sr} A'$ *implies* $A \xrightarrow{\mathsf{tr}}_{sc} A'$.

Next, we show that our reduced semantics only explores specific representatives. Given a frame $\Phi$, a plausible trace $\mathsf{tr}$ is $\Phi$-minimal if it is minimal (w.r.t. $\prec_{\mathsf{lex}}$) in its equivalence class modulo $\equiv_\Phi$.

**Lemma 12.** *Let* $A$ *be an initial configuration and* $A' = (\mathcal{P}; \varnothing; \Phi)$ *be a configuration such that* $A \xrightarrow{\mathsf{tr}}_c A'$ *(resp.* $A \xrightarrow{\mathsf{tr}}_{sc} A'$*). We have that* $\mathsf{tr}$ *is* $\Phi$*-minimal if, and only if,* $A \xrightarrow{\mathsf{tr}}_r A'$ *(resp.* $A \xrightarrow{\mathsf{tr}}_{sr} A'$*).*

*Proof.* We first deal with the semi-compressed and semi-reduced semantics case and then show that the compressed and reduced case follows from the former.

(**Semi-compressed and semi-reduced case**) Let $A$ and $(\mathcal{P}; \varnothing; \Phi)$ be two configurations such that $A \xrightarrow{\mathsf{tr}}_{sc} (\mathcal{P}; \varnothing; \Phi)$.

($\Rightarrow$) We first show that if $\mathsf{tr}$ is $\Phi$-minimal, then $A \xrightarrow{\mathsf{tr}}_{sr} (\mathcal{P}; \varnothing; \Phi)$ by induction on the trace $\mathsf{tr}$. The base case, *i.e.* $\mathsf{tr} = \epsilon$ is straightforward. Now, assume that $\mathsf{tr} = \mathsf{tr}_0.b$ for some block $b$ and $A \xrightarrow{\mathsf{tr}_0}_{sc} (\mathcal{P}_0; \varnothing; \Phi_0) \xrightarrow{b}_{sc} A'$. Since $\mathsf{tr}$ is $\Phi$-minimal, we also have that $\mathsf{tr}_0$ is $\Phi_0$-minimal and thus we obtain by induction hypothesis that $A \xrightarrow{\mathsf{tr}_0}_{sr} (\mathcal{P}_0; \varnothing; \Phi_0)$. To conclude, it remains to show that

$\mathsf{tr}_0 \triangleright b'$ for any $b'$ such that $(b' =_{\mathsf{E}} b)\Phi_0$. Assume that it is not the case, this means that for some $b'$ such that $(b =_{\mathsf{E}} b')\Phi_0$, the trace $\mathsf{tr}_0$ can be written $\mathsf{tr}'_0.b_0.\ldots.b_n$ with:

$$b_i \parallel b' \text{ and } b_i \prec b' \text{ for any } i > 0, \text{ as well as } b_0 \parallel b' \text{ and } b' \prec b_0.$$

Let $\mathsf{tr}' = \mathsf{tr}'_0.b'.b_0 \ldots b_n$. We have $\mathsf{tr}' \prec_{\mathsf{lex}} \mathsf{tr}$ and $\mathsf{tr}' \equiv_\Phi \mathsf{tr}_0.b'$, which contradicts the $\Phi$-minimality of $\mathsf{tr}$.

($\Leftarrow$) Now, we assume that $\mathsf{tr}$ is not $\Phi$-minimal, and we want to establish that $\mathsf{tr}$ cannot be executed in the semi-reduced semantics. Let $\mathsf{tr}_m$ be a $\Phi$-minimal trace of the equivalence class of $\mathsf{tr}$. We have in particular $\mathsf{tr}_m \equiv_\Phi \mathsf{tr}$ and $\mathsf{tr}_m \prec_{\mathsf{lex}} \mathsf{tr}$. Now, we let $\mathsf{tr}^s_m$ (resp. $\mathsf{tr}^s$) be the "trace of labelled skeletons" associated to $\mathsf{tr}_m$ (resp. $\mathsf{tr}$). Let $\mathsf{tr}^s_0$ be the longest common prefix of $\mathsf{tr}^s_m$ and $\mathsf{tr}^s$, and $\mathsf{tr}_0$ (resp. $\mathsf{tr}'_0$) be the corresponding prefix of $\mathsf{tr}$ (resp. $\mathsf{tr}_m$). We have a decomposition of the form $\mathsf{tr} = \mathsf{tr}_0.b.\mathsf{tr}_1$ and $\mathsf{tr}_m = \mathsf{tr}'_0.b_m.\mathsf{tr}'_1$ with $(\mathsf{tr}_0 =_{\mathsf{E}} \mathsf{tr}'_0)\Phi$ and $b_m \prec b$. Again, since when dropping recipes, the relation $\equiv_\Phi$ only swaps sequentially independent labelled skeletons, block $b_m$ must have a counterpart in $\mathsf{tr}$ and, more precisely, in $\mathsf{tr}_1$. We thus have a more precise decomposition of $\mathsf{tr}$: $\mathsf{tr} = \mathsf{tr}_0.b.\mathsf{tr}_{11}.b'_m.\mathsf{tr}_{12}$ such that $(b'_m =_{\mathsf{E}} b_m)\Phi$.

We now show that $b'_m$ cannot be executed after $\mathsf{tr}_0.b.\mathsf{tr}_{11}$ in the semi-reduced semantics (assuming that the trace has been executed so far in the reduced semantics). In other words, we show that $\mathsf{tr}_0.b.\mathsf{tr}_{11} \triangleright b'_m$ does *not* hold. We have seen that $(b'_m =_{\mathsf{E}} b_m)\Phi$ and $b_m \prec b$ so it suffices to show:

$$b_m \parallel b_i \text{ for any } b_i \in b.\mathsf{tr}_{11}$$

First, we prove $b_i \parallel^r b_m$ (*i.e.* they are recipe independent) for any $b_i \in b.\mathsf{tr}_{11}$. This comes from the fact that $\mathsf{tr}'_0.b_m.\mathsf{tr}'_1 = \mathsf{tr}_m$ is plausible, and thus the inputs of $b_m$ only use handles introduced in $\mathsf{tr}'_0$ which are the same as those introduced in $\mathsf{tr}_0$. In particular, the inputs of $b_m$ do not rely on the handles introduced in $b.\mathsf{tr}_{11}$. Similarly, using the fact that $\mathsf{tr}_0.b.\mathsf{tr}_{11}.b'_m.\mathsf{tr}_{12} = \mathsf{tr}$ is plausible and $b'^-_m = b^-_m$, we deduce that handles of outputs of $b_m$ are not used in $b.\mathsf{tr}_{11}$.

Second, we show that $b_i \parallel^s b_m$ (*i.e.* they are sequentially independent) for any $b_i \in b.\mathsf{tr}_{11}$. For this, we remark that for any traces $\mathsf{tr}_1.b.\mathsf{tr}_2 \equiv_\Phi \mathsf{tr}'_1.b'.\mathsf{tr}'_2$ such that $(b =_{\mathsf{E}} b')\Phi$, we have that $b \parallel^s b_s$ for all $b_s \in \mathsf{skl}(\mathsf{tr}'_1)\backslash\mathsf{skl}(\mathsf{tr}_1)$ where $\mathsf{skl}(\mathsf{tr})$ is the multiset of labelled skeletons of blocks of $\mathsf{tr}$, and $\backslash$ should be read as multiset substraction. This can be easily shown by induction on the relation $\equiv_\Phi$. By applying this helping remark to $\mathsf{tr}_0.b'_m.\mathsf{tr}'_2 \equiv_\Phi \mathsf{tr}_0.b.\mathsf{tr}_{11}.b'_m.\mathsf{tr}_{12}$, we obtain the required conclusion: $b'_m \parallel^s b.\mathsf{tr}_{11}$ and thus $b_m \parallel^s b.\mathsf{tr}_{11}$.

(**Compressed and reduced case**) We consider an execution $A \xrightarrow{\mathsf{tr}}_c (\mathcal{P};\Phi) = A'$.

($\Rightarrow$) We suppose that $\mathsf{tr}$ is $\Phi$-minimal. Even if it means modifying the resulting multiset, we also have an execution $A \xrightarrow{\mathsf{tr}'}_{sc} (\mathcal{Q};\Phi) = A'_s$ for some $\mathcal{Q}$. Indeed, it suffices to replace the final potential rule $\mathsf{Release}_\perp$ by $\mathsf{Release}^s$ followed by Neg (for removing the released null process). Applying the above result, we obtain an execution $A \xrightarrow{\mathsf{tr}'}_{sr} A'_s$. Now it suffices to revert the potential replacement made above yielding $A \xrightarrow{\mathsf{tr}}_r A'$.

($\Leftarrow$) We suppose that $A \xrightarrow{\mathsf{tr}}_r A'$. Applying the same modification explained above, we obtain an execution $A \xrightarrow{\mathsf{tr}'}_{sr} (\mathcal{Q};\Phi)$ for some $\mathcal{Q}$. Invoking the above result, we deduce that $\mathsf{tr}$ is $\Phi$-minimal. □

**Remark 8** (Reachability properties verified with the reduced semantics)**.** *We now characterise reachability predicates that are preserved by the reduced semantics the same way it was done for the compressed semantics (see Remark 6). The semi-reduced semantics is sound and complete for reachable predicates that are:*

- *monotone (*i.e. *if an execution* $A \xrightarrow{\mathsf{tr}} (\mathcal{P};\Phi)$ *satisfies the predicate then so does* $A \xrightarrow{\mathsf{tr}.\mathsf{tr}'} (\mathcal{Q};\Phi\cup\Psi)$*), and,*
- *stable by* $=_{\mathsf{E}}$ *(*i.e. *if* $A \xrightarrow{\mathsf{tr}} (\mathcal{P};\Phi)$ *satisfies the predicate then so does* $A \xrightarrow{\mathsf{tr}'} (\mathcal{Q};\Phi')$ *if* $\mathsf{tr}\Phi =_{\mathsf{E}} \mathsf{tr}'\Phi'$*),*
- *invariant by permutations of actions (*i.e. *if* $A \xrightarrow{\mathsf{tr}} B$ *satisfies the predicate for some configurations* $A, B$ *and a trace* $\mathsf{tr}$ *then for any trace* $\mathsf{tr}'$ *equals to* $\mathsf{tr}$ *up to a permutation of actions, any execution* $A \xrightarrow{\mathsf{tr}'} B$ *must satisfy the predicate).*

*The above simply follows from lemmas 11 and 12.*

*For instance, secrecy is monotone, stable by* $=_{\mathsf{E}}$*, and, invariant by permutations.*

### 5.4.5 Equivalence

The reduced semantics induces an equivalence $\approx_r$ that we define similarly to the compressed one, and we then establish its soundness and completeness w.r.t. $\approx_c$.

**Definition 29.** *Let* $A$ *and* $B$ *be two configurations. We say that* $A \sqsubseteq_r B$ *(resp.* $A \sqsubseteq_{sr} B$ *) when, for every* $A \xrightarrow{\mathsf{tr}}_r A'$ *(resp.* $A \xrightarrow{\mathsf{tr}}_{sr} A'$*) such that* $bc(\mathsf{tr}) \cap fc(B) = \emptyset$*, there exists* $B \xrightarrow{\mathsf{tr}}_r B'$ *(resp.* $B \xrightarrow{\mathsf{tr}}_{sr} B'$*) such that* $\Phi(A') \sim \Phi(B')$*. They are* reduced trace equivalent*, denoted* $A \approx_r B$*, if* $A \sqsubseteq_r B$ *and* $B \sqsubseteq_r A$*. They are* semi-reduced trace equivalent*, denoted* $A \approx_{sr} B$*, if* $A \sqsubseteq_{sr} B$ *and* $B \sqsubseteq_{sr} A$*.*

We first prove that semi-reduced trace equivalence coincides with the semi-compressed trace equivalence. To do so, we first show that $\equiv_\Phi$ is stable by static equivalence as stated and proved next.

**Proposition 13.** *For any static equivalent frames* $\Phi \sim \Psi$ *and compressed traces* $\mathsf{tr}$ *and* $\mathsf{tr}'$*, we have that* $\mathsf{tr} \equiv_\Phi \mathsf{tr}'$ *if, and only if,* $\mathsf{tr} \equiv_\Psi \mathsf{tr}'$*.*

*Proof.* The two implications are symmetric, we thus only show one implication. Considering the two generators of $\equiv_\Phi$, the only non-trivial step is to show that $\mathsf{tr}.b_1.\mathsf{tr}' \equiv_\Psi \mathsf{tr}.b_2.\mathsf{tr}'$ when $(b_1 =_{\mathsf{E}} b_2)\Phi$. But the latter condition, together with $\Phi \sim \Psi$, yields $(b_1 =_{\mathsf{E}} b_2)\Psi$ which allows us to conclude. □

Then we need to show that the the semi-compressed trace equivalence restricted to complete executions actually coincides with the semi-compressed trace equivalence.

**Lemma 13.** *Let* $A$ *and* $B$ *be two action-deterministic configurations. If for any complete execution of the form* $A \xrightarrow{\mathsf{tr}}_{sc} (\mathcal{P};\varnothing;\Phi)$ *with* $bc(\mathsf{tr})\cap fc(B) = \emptyset$*, there exists an execution* $B \xrightarrow{\mathsf{tr}}_{sc} (\mathcal{Q};\varnothing;\Psi)$ *such that* $\Phi \sim \Psi$*, then* $A \sqsubseteq_{sc} B$*.*

*Proof.* Let $A$ and $B$ be two action-deterministic configurations, and assume that for any complete execution $A \xrightarrow{\mathsf{tr}}_{sc} (\mathcal{P}; \varnothing; \Phi)$ with $bc(\mathsf{tr}) \cap fc(B) = \emptyset$, there exists an execution $B \xrightarrow{\mathsf{tr}}_{sc} (\mathcal{Q}; \varnothing; \Psi)$ such that $\Phi \sim \Psi$. Now, we have to establish that $A \sqsubseteq_{sc} B$.

Let $(\mathcal{P}'; \varnothing; \Phi')$ be a configuration such that $A \xrightarrow{\mathsf{tr}'}_{sc} (\mathcal{P}'; \varnothing; \Phi')$. First, we can complete this execution to reach a process $(\mathcal{P}; \varnothing; \Phi)$ such that each process $P \in \mathcal{P}$ is replicated, *i.e.*

$$A \xrightarrow{\mathsf{tr}'}_{sc} (\mathcal{P}'; \varnothing; \Phi') \xrightarrow{\mathsf{tr}^+}_{sc} (\mathcal{P}; \varnothing; \Phi) \text{ is a complete execution.}$$

Without loss of generality, we can choose $\mathsf{tr}^+$ so that it satisfies $bc(\mathsf{tr}^+) \cap fc(B) = \emptyset$. By hypothesis, we know that there exists an execution $B \xrightarrow{\mathsf{tr}'\mathsf{tr}^+}_{sc} (\mathcal{Q}; \varnothing; \Psi)$ such that $\Phi \sim \Psi$. Let $B'$ be the configuration reached along this execution after the execution of $\mathsf{tr}'$ and $\Psi'$ its frame. Similarly to the proof of Lemma 7, we prove that $\Phi \sim \Psi$ implies $\Phi' \sim \Psi'$. □

**Lemma 14.** *Let $A$ and $B$ be two initial, action-deterministic configurations.*

$$A \approx_{sc} B \text{ if, and only if, } A \approx_{sr} B$$

*Proof.* We prove the two directions separately.

($\Rightarrow$) *$A \sqsubseteq_{sc} B$ implies $A \sqsubseteq_{sr} B$.* Consider an execution of the form $A \xrightarrow{\mathsf{tr}}_{sr} (\mathcal{P}; \varnothing; \Phi)$ with $bc(\mathsf{tr}) \cap fc(B) = \emptyset$. Using Lemma 11, we know that $A \xrightarrow{\mathsf{tr}}_{sc} (\mathcal{P}; \varnothing; \Phi)$, and Lemma 12 tells us that $\mathsf{tr}$ is $\Phi$-minimal. Since $A \sqsubseteq_{sc} B$, we deduce that there exists $(\mathcal{Q}; \varnothing; \Psi)$ such that:

$$B \xrightarrow{\mathsf{tr}}_{sc} (\mathcal{Q}; \varnothing; \Psi) \text{ and } \Phi \sim \Psi.$$

Now, by Proposition 13 we obtain that $\mathsf{tr}$ is also $\Psi$-minimal, and so Lemma 12 tells us that the execution of $\mathsf{tr}$ by $B$ can also be performed in the semi-reduced semantics.

($\Leftarrow$) *$A \sqsubseteq_{sr} B$ implies $A \sqsubseteq_{sc} B$.* Relying on Lemma 13, it is actually sufficient to show that for any complete execution $A \xrightarrow{\mathsf{tr}}_{sc} (\mathcal{P}; \varnothing; \Phi)$ with $bc(\mathsf{tr}) \cap fc(B) = \emptyset$, there exists an execution of the form $B \xrightarrow{\mathsf{tr}}_{sc} (\mathcal{Q}; \varnothing; \Psi)$ such that $\Phi \sim \Psi$. We thus consider a complete execution $A \xrightarrow{\mathsf{tr}}_{sc} (\mathcal{P}; \varnothing; \Phi) = A'$ such that $bc(\mathsf{tr}) \cap fc(B) = \emptyset$. Note that since the given execution is complete, we have that $A'$ is initial (this will be crucial to invoke Lemma 12). Let $\mathsf{tr}'$ be a $\Phi$-minimal trace in the equivalence class of $\mathsf{tr}$. Applying the Lemma 12, we obtain an execution $A \xrightarrow{\mathsf{tr}'}_{sr} A'$. By hypothesis, there must be some $B'$ such that:

$$B \xrightarrow{\mathsf{tr}'}_{sr} B' \text{ and } \Phi(B') \sim \Phi.$$

Using Lemma 11, we obtain the same execution in the semi-compressed semantics. Finally, by Proposition 13 we obtain $\mathsf{tr}' \equiv_{\Phi(B')} \mathsf{tr}$, and by Lemma 10 we obtain:

$$B \xrightarrow{\mathsf{tr}}_{sc} B' \text{ and } \Phi(B') \sim \Phi$$

concluding the proof. □

Finally, we prove that the reduced trace equivalence coincides with the semi-reduced trace equivalence implying, as a corollary (stated in Section 5.5), that the reduced trace equivalence coincides with the regular trace equivalence.

**Lemma 15.** *Let $A$ and $B$ be two initial action-deterministic configurations.*

$$A \approx_{sr} B \text{ if, and only, if, } A \approx_{r} B$$

*Proof.* The direction ($\Rightarrow$) is easy: it suffices to distinguish the case of proper executions from improper ones and directly apply the hypothesis. We focus on ($\Leftarrow$). Assume that $A \sqsubseteq_r B$. Let $A'$ be such that $A \xrightarrow{\mathsf{tr}}_{sr} A'$ for some $\mathsf{tr}$ such that $bc(\mathsf{tr}) \cap fc(B) = \emptyset$. Lemma 11 tells us that $A \xrightarrow{\mathsf{tr}}_{sc} A'$, and thanks to Lemma 12, we have that $\mathsf{tr}$ is $\Phi(A')$-minimal.

Let $b_1, \ldots, b_k$ be the improper blocks that occur in $\mathsf{tr}$. We have that there exist $\mathsf{tr}_0, \mathsf{tr}_1, \ldots, \mathsf{tr}_k$ made of proper blocks such that $\mathsf{tr} = \mathsf{tr}_0.b_1.\mathsf{tr}_1.b_2.\ldots.\mathsf{tr}_{k-1}.b_k.\mathsf{tr}_k$. We have that $b_1, \ldots, b_k$ are pairwise independent, and also:

$$\mathsf{tr} \equiv_{\Phi(A')} \mathsf{tr}_0.\mathsf{tr}_1 \ldots \mathsf{tr}_k.b_1.\ldots.b_k$$

Because (i) $\mathsf{tr}_i$ are proper traces, (ii) there are no dependencies between $b_i$ and $\mathsf{tr}_j$ for $i < j$, and (iii) the $b_i$ do not have any output, we have that $A \xrightarrow{\mathsf{tr}_0.\mathsf{tr}_1.\ldots.\mathsf{tr}_k}_r A_0$, and also that:

$$A \xrightarrow{\mathsf{tr}_0.b_1}_r A_1,\ A \xrightarrow{\mathsf{tr}_0.\mathsf{tr}_1.b_2}_r A_2,\ \ldots,\ A \xrightarrow{\mathsf{tr}_0.\mathsf{tr}_1.\ldots.\mathsf{tr}_{k-1}.b_k}_r A_k.$$

Thanks to our hypothesis, we deduce that there exist $B_0, B_1, \ldots, B_k$ such that $B \xrightarrow{\mathsf{tr}_0.\mathsf{tr}_1.\ldots.\mathsf{tr}_k}_r B_0$ with $\Phi(A_0) \sim \Phi(B_0)$, and also that:

$$B \xrightarrow{\mathsf{tr}_0.b_1}_r B_1,\ B \xrightarrow{\mathsf{tr}_0.\mathsf{tr}_1.b_2}_r B_2,\ \ldots,\ B \xrightarrow{\mathsf{tr}_0.\mathsf{tr}_1.\ldots.\mathsf{tr}_{k-1}.b_k}_r B_k.$$

We deduce that there exists $B'$ such that $B \xrightarrow{\mathsf{tr}_0.\mathsf{tr}_1.\ldots.\mathsf{tr}_k.b_1.\ldots.b_k}_{sc} B'$. Next, we observe that $\Phi(A') = \Phi(A_0) \sim \Phi(B_0) = \Phi(B')$. From this we conclude $\mathsf{tr} \equiv_{\Phi(B')} \mathsf{tr}_0.\mathsf{tr}_1 \ldots \mathsf{tr}_k.b_1.\ldots.b_k$, hence $B \xrightarrow{\mathsf{tr}}_{sc} B'$ by Lemma 10. Since $\mathsf{tr}$ is $\Phi(A')$-minimal it is also $\Phi(B')$-minimal (by Proposition 13), and thus $B \xrightarrow{\mathsf{tr}}_{sr} B'$ by Lemma 12. □

## 5.5 Main Result and Discussions

The main theorem of this chapter states that the –highly optimised– reduced semantics can be used instead of the regular semantics to verify trace equivalence between action-deterministic configurations. In other words, we finally obtain that the *reduced trace equivalence* coincides with the *trace equivalence* for action-deterministic configurations having same skeletons.

**Theorem 3.** *Let $A$ and $B$ be two initial action-deterministic configurations such that* $\mathsf{skl}(A) = \mathsf{skl}(B)$. *It holds that:*

$$A \approx B \text{ if, and only, if, } \lceil A \rceil \approx_r \lceil B \rceil$$

*Proof.* It directly follows from Lemma 14, Lemma 15 and Corollary 2. □

The practical implications of this theorem are addressed in Chapter 6. In the remaining of this section, we discuss some theoretical aspects of the refined semantics we developed in this chapter and some possible future work.

**Discussion on Compression vs. Recursion.** We did not consider recursive processes in this chapter. There are fundamental reasons for that which we explain now.

First, remark that considering the unfolding of recursive processes as part of the internal reduction is not an option. Indeed, in such a case, infinite executions induced by recursive processes could make the compression get stuck in a negative phase (*e.g.* with $\mathrm{rec}X.\mathtt{out}(c,\mathsf{ok}).X$) or a positive phase (*e.g.* with $\mathrm{rec}X.\mathtt{in}(c,\mathsf{ok}).X$) without the ability to explore other processes in the multiset and thus losing completeness for reachability (*e.g.* losing Lemma 4). Therefore, recursive processes shall be given an action (not necessarily observable) and thus a skeleton. Such a skeleton should be different from the already defined ones since the compression rule for recursive processes (to be defined) must be different from the others (otherwise, the above problem arises again). But then, as stated by the Strong Symmetry Lemma (*i.e.* Lemma 2), the attacker should be able to deduce skeletons of active processes from initial skeletons and performed observable actions. This cannot be enforced in the general case; for instance, the attacker has no way to distinguish $P = \mathrm{rec}X.\mathtt{out}(c,\mathsf{ok}).X$ from $\mathtt{out}(c,\mathsf{ok}).P$ (essentially because he has no way to distinguish $\mathtt{out}(c,\mathsf{ok}).0$ from $\mathrm{rec}X.\mathtt{out}(c,\mathsf{ok}).0$). One could come up with restrictions on the class of processes we deal with to avoid this problem. However, in our opinion, those classes are really restrictive and would not reflect typical uses of recursion mainly because those classes would make the unfoldings of recursion observable to the attacker.

**Towards dropping the action-deterministic assumption.** It would be interesting to support processes that are not action-deterministic, which are commonplace when analysing *e.g.* anonymity or unlinkability scenarios. We tried to adapt our POR techniques towards this goal but encountered very difficult challenges that we explain now. We leave the task of addressing those challenges for future work.

For this discussion, we focus on the compression since challenges already arise for this first technique and we leave aside the design choices and the proof technique we followed in this chapter (*e.g.* annotations, the Strong Symmetry Lemma). What we seek for is a class of swaps of actions that can systematically be done for executions of processes that are not necessarily action-deterministic. The next step would be to design an exploration strategy in the flavor of the compression strategy that could then be proved complete thanks to those swaps.

Let us illustrate problems one encounters when seeking for possible swaps between output and input. We start with a minimal example abstracting away data to focus on the action's nature: $P_1^l = \mathtt{in}_c \mid \mathtt{out}_e.\mathtt{in}_d$ and $P_1^r = \mathtt{in}_d \mid \mathtt{out}_e.\mathtt{in}_c$. Those two processes are not trace equivalent (*i.e.* $(\{P_1^l\};\emptyset) \not\approx (\{P_1^r\};\emptyset)$) and a trace witnessing this would be $\mathtt{in}_c$ (or $\mathtt{in}_d$). However, if one imposes a strategy exploring outputs in priority, as the compression does, one would only explore traces starting by the output $\mathtt{out}_e$. But there is no witness of non-equivalence starting with $\mathtt{out}_e$ since after this output the two processes are structurally equal (*i.e.* $\mathtt{in}_c \mid \mathtt{in}_d$).

One could argue that there is a more direct way to detect the non-equivalence in the former example; indeed the immediately available actions are not the same in $P_1^l$ and $P_1^r$. However, the same problem arises for slightly more complex witnesses that require to "violate" the simple strategy of prioritising outputs for many actions. For instance, consider the following processes

$P_2^l = \mathtt{in}_\alpha.A \mid \mathtt{out}_e.\mathtt{in}_\alpha.B$ and $P_2^r = \mathtt{in}_\alpha.B \mid \mathtt{out}_e.\mathtt{in}_\alpha.A$ such that $A$ and $B$ are not equivalent but for "long" witnesses; *e.g.* $A$ made of $k$ inputs ($k$ can be chosen arbitrarily) followed by $\mathtt{out}_f$ while $B$ made of the same inputs but followed by $\mathtt{out}_g$. Again, $P_2^l$ and $P_2^r$ are not trace equivalent. This example shows that even when one verifies that the actions immediately enabled are the same on both sides, some witnesses can be found only by violating the simple strategy aforementioned for an arbitrary long time (because $k$ can be chosen arbitrary large).

The crux of the problem is that the backward swap (*i.e.* moving towards the beginning) of the action $\mathtt{out}_e$ with the other actions (*i.e.* $\mathtt{in}_c, \mathtt{in}_e$ in the first example and $\mathtt{in}_\alpha$ in the second example) do not violate any sequential dependency nor data dependency but are nevertheless impossible for a reason that only arises in the non-deterministic case. This novel reason is that performing the action $\mathtt{out}_e$ makes available an action that is related to the actions to be swapped (either they are dependent or, worse, they are equal). Note that there actually are other kinds of interactions preventing swaps that we do not describe here. Those new entangled interactions between actions to be swapped caused by the non-determinism are hard to predict and analyse. In such cases, the strategy cannot impose any specific ordering since it is not certain that all the necessary swaps, which are needed in order to prove that we do not lose completeness by not exploring other interleavings, are all possible and not prevented by one of the above reasons.

**Other Directions for Future Work.** First, we could investigate the role of the particular choice of the order $\prec$, to determine heuristics for maximising the impact of reduction. In particular, it could be chosen on a case-by-case basis after having analysed the shape of the processes to be verified.

Further, one may adapt our treatment of replication to bounded replication to obtain a first symmetry elimination scheme, which should provide a significant optimisation when studying security protocols with several (but finitely many) sessions. More precisely, if one wants to analyse $n$ sessions of a process $P$, he may consider the bounded replication $!^{[n]}_{\frac{a}{c,\overline{n}}}P$ that intuitively represents a standard replication $!^{a}_{\frac{}{c,\overline{n}}}P$ that can be unfolded at most $n$ times. The benefit stems from the fact that such a bounded replication will be better optimised using our POR techniques than the parallel composition $\Pi_{1\leq i\leq n}P$.

Finally, it is certainly possible to leverage our POR techniques for processes that are not fully action-deterministic but that feature *phases*, some of them being action-deterministic. This is certainly useful *e.g.* for improving the verification of e-voting protocols. For instance, in presence of strong phases (or synchronisation barriers) as defined in [BS16], we conjecture that our POR techniques can be leveraged inside action-deterministic phases without losing completeness. We claim that the same can be achieved with weak phases (*e.g.* defined in [DRS08]) provided that the semantics considers actions produced when moving to next phases as observable for the attacker.

*Chapter 6*

# Putting Reduced Semantics into Practice and Integration in Apte

In the previous chapter, we devised optimised semantics that only explore a selection of interleavings according to some execution strategy (*i.e.* compression, reduction). We notably defined the reduced semantics (see Subsection 5.4.3) combining our two POR techniques: compression and reduction. We eventually obtained the main theorem of the previous chapter (*i.e.* Theorem 3) that states that this reduced semantics can be used instead of the regular semantics to verify the trace equivalence of action-deterministic configurations.

We now seek to put this reduced semantics into practice. Since we target methods and tools dealing with bounded number of sessions only, we must restrict the class of configurations (by comparison with the previous chapter). We have chosen to work with *simple configurations* that lie in a strict sub-class of action-deterministic configurations (without replication) satisfying syntactical constraints that are easy to verify. We define this class and the instance of the model we shall work with in Section 6.1.

Then, we show how the compressed and the reduced semantics can be combined with constraint solving-based methods (Section 6.2). We shall define a (classic) symbolic semantics, restrict it to follow the compressed strategy and finally show how to refine it further by translating the authorised predicate into a new type of constraints. We also prove that the soundness and completeness results on the induced trace equivalences can be lifted to the symbolic setting easily. All those steps involve a high level of technicalities but do not represent a key contribution in themselves. However, the fact that it is possible to translate quite easily our POR techniques from a regular, concrete semantics to a symbolic semantics validates our design choices.

Next, we show how we followed this approach to integrate our POR techniques in the tool Apte (Section 6.3). More interestingly, we also prove the soundness of our integration; *i.e.* we show that the modified version of Apte that uses our optimised semantics still decides trace equivalence for simple processes. Finally, we provide benchmarks showing dramatic speedups brought by our optimised semantics (Section 6.4) before concluding with future works (Section 6.5).

## 6.1 Instantiation of the Model and Class of Processes

In this chapter, we work with a strict sub-class of action-deterministic processes we considered in the previous chapter (see Hypothesis 1). We also consider an instantiation of the term-algebra and the semantics we used in the previous chapter. We now formally define those instantiations and the class of processes we shall work with.

**Term Algebra.** We consider an arbitrary term-algebra made of an arbitrary signature $\Sigma$ and an equational theory $=_{\mathsf{E}}$. However, contrary to the previous chapter, we need to restrict the computation relations $\bullet \Downarrow \bullet$ we can deal with to be computation relations induced by an extension $\equiv_{\mathsf{E}_d}$ of $=_{\mathsf{E}}$ as defined in Chapter 2, Subsection 2.3.2. Remind that such computation relations are induced by equational theories $\equiv_{\mathsf{E}_d}$ over terms: a term $t$ computes a messages $u$ (*i.e.* $t \Downarrow u$) when $t \equiv_{\mathsf{E}_d} u$ and $t$ is *valid* (*i.e.* for all all sub-term $t'$ of $t$, there exists a message $u'$ such that $t' \equiv_{\mathsf{E}_d} u'$). We note $\mathsf{valid}(t)$ when $t$ is valid.

This restriction is imposed by the constraint solving approach where computations (whether it leads to a success or a failure) shall be modelled by (dis)equality constraints. Obviously this is in line with the theory behind the tool Apte we will discuss later in Section 6.3.

**Semantics.** Similarly to the previous chapter, we consider the internal reduction induced by the relation $\mathscr{R} = \mathscr{R}_{\text{test}} \cup \mathscr{R}_{\text{out}}$ (see Definition 11 in Section 4.1). Remind that this internal reduction evaluates conditionals ($R_{\text{test}}$) and removes blocked outputs ($R_{\text{out}}$) greedily.

**Class of Configurations.** In this chapter, we eventually define a symbolic semantics based on the constraint solving framework notably featuring deducibility and (dis)equality constraints. While this will be sufficient to represent equality over terms, it does not allow to describe arbitrary computations in let constructs (*e.g.* $\mathsf{let}\ x = \mathsf{sdec}(y, z); \mathtt{in} \ldots$). For this reason, we must restrict ourselves to use conditionals (*i.e.* let constructs) containing only computations corresponding to equality tests (*e.g.* $\mathsf{let}\ x = \mathsf{eq}(y, z)\ \mathtt{in} \ldots$). We thus introduce the following notation to ease the readability of the further developments.

**Notation 2.** *For some processes $P, Q$ with $x \notin \text{vars}(P)$, and terms $t_1, t_2$, we note* $\mathsf{if}\ t_1 = t_2\ \mathsf{then}\ P\ \mathsf{else}\ Q$ *the process* $\mathsf{let}\ x = \mathsf{eq}(t_1, t_2)\ \mathtt{in}\ P\ \mathsf{else}\ Q$ *where* eq *is described in Example 11 (in Subsection 2.3.2).*

In the class of processes we eventually define, we shall only consider conditionals of the form $\mathsf{if}\ t_1 = t_2\ \mathsf{then}\ P\ \mathsf{else}\ Q$. This is in line with restrictions made in the tool Apte that we eventually leverage.

We consider the fragment of *simple processes* without replication and recursion built on *basic processes* following [CCD13a]. A basic process represents a party in a protocol, which may sequentially perform actions such as waiting for a message, checking equality between terms (using the construct introduced above), or outputting a message. Then, a simple process is a parallel

composition of such basic processes playing on distinct channels. The simple configurations form a sub-class of action-deterministic configurations based on a simple syntactic criterion.

**Definition 30** (Class of configurations)**.** *The set of* basic processes *on $c \in \mathcal{C}$ is defined using the following grammar (where $t, t_1, t_2 \in \mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X})$ and $x \in \mathcal{X}$):*

$$\begin{array}{rcll} P, Q & := & 0 & \textit{null} \\ & | & \mathsf{if}\ t_1 = t_2\ \mathsf{then}\ P\ \mathsf{else}\ Q & \textit{conditional} \\ & | & \mathtt{in}(c, x).P & \textit{input} \\ & | & \mathtt{out}(c, t).P & \textit{output} \end{array}$$

*A* simple process *$\mathcal{P} = \{P_1, \ldots, P_n\}^{\#}$ is a multiset of basic processes $P_i$ on pairwise distinct channels $c_i$. We consider the class of* simple configurations*;* i.e. *configurations whose first components are simple ground processes.*

**Example 33.** *The Private Authentication protocol defined in Subsection 3.3.2, which will be our running example in this chapter, lies in our class: $P(\mathsf{sk}_A, \mathsf{pk}(skb)), Q(\mathsf{sk}_B, \mathsf{pk}(\mathsf{sk}_A)), Q_0(\mathsf{sk}_B, \mathsf{pk}(\mathsf{sk}_A))$ are basic processes, $\mathcal{P}$ is a simple process and $(\mathcal{P}; \Phi_0)$ is a simple configuration.*

We do not consider recursion and replication since the constraint solving approach is not compatible with unbounded executions. Additionally, we dot not consider creation of names. The latter is without loss of generality though, because they can be replaced by private constants as already explained in Section 5.1. We do not consider parallel composition either to ease the presentation and the (already) complex proofs. We let the extension of our results to a larger class of processes featuring parallel compositions as future work.

Altogether, the instance of the semantics for the class of configurations under consideration (see Definition 30) is depicted in Figure 6.1. Note that it can be seen as an instantiation of the semantics we considered in the previous chapter (see Section 5.1) for a sub-class of configurations.

$$\begin{array}{lll} \mathsf{In} & (\{\mathtt{in}(c, x).P\} \uplus \mathcal{P}; \Phi) \xrightarrow{\mathtt{in}(c, R)} (\{P\{x \mapsto u\}\} \uplus \mathcal{P}; \Phi) & \\ & & \text{where } R \text{ is a recipe such that } R\Phi \Downarrow u \\ \mathsf{Out} & (\{\mathtt{out}(c, t).P\} \uplus \mathcal{P}; \Phi) \xrightarrow{\mathtt{out}(c, w)} (\{P\} \uplus \mathcal{P}; \Phi \uplus \{w \mapsto u\}) & \\ & & \text{where } w \text{ is a fresh variable and } t \Downarrow u \end{array}$$

**Figure 6.1** Instance of the Semantics for Chapter 6

## 6.2 Combining Compression and Reduction with Constraint Solving

In this section, we propose a symbolic semantics that we iteratively refine in order to embed our two POR techniques.

### 6.2.1 Symbolic Semantics

We define a symbolic semantics following, *e.g.* [MS01, Bau05]. Such a symbolic semantics avoids potentially infinite branching of concrete semantics due to inputs from the environment. Correctness is maintained by associating with each process a set of constraints on terms.

#### Constraint systems

Following the notations of [Bau05], we consider a new set $\mathcal{X}^2$ of *second-order variables*, denoted by $X$, $Y$, etc. We shall use those variables to abstract over recipes. We denote by $\mathrm{vars}^2(o)$ the set of free second-order variables of an object $o$, typically a constraint system. To prevent ambiguities, we shall use $\mathrm{vars}^1(\bullet)$ instead of $\mathrm{vars}(\bullet)$ for free first-order variables.

**Definition 31** (constraint system)**.** *We consider three kinds of* constraints*:*

$$D \vdash^?_X x \qquad u =^? v \qquad u \neq^? v$$

*where $D \subseteq \mathcal{W}$, $X \in \mathcal{X}^2$, $x \in \mathcal{X}$ and $u, v \in \mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X})$. We call* symbolic frame *(noted $\Phi, \Psi$) substitutions from $\mathcal{W}$ to terms. A* constraint system *$\mathcal{C} = (\Phi; \mathcal{S})$ consists of a symbolic frame $\Phi$, and a set of constraints $\mathcal{S}$.*

The first kind of constraint expresses that a second-order variable $X$ has to be instantiated by a recipe that uses only variables from a certain set $D$, and that the obtained term should be $x$. The handles in $D$ represent terms that have been previously outputted by the process.

We are not interested in general constraint systems, but only consider constraint systems that are *well-formed*. Given a constraint system $\mathcal{C}$, we define a dependency order on first-order variables in $\mathrm{vars}^1(\mathcal{C}) \cap \mathcal{X}$ by declaring that $x$ depends on $y$ if, and only if, $\mathcal{S}$ contains a deduction constraint $D \vdash^?_X x$ with $y \in \mathrm{vars}^1(\Phi(D))$. A constraint system $\mathcal{C}$ is *well-formed* if:

- the dependency relationship is acyclic, and
- for every $x \in \mathrm{vars}^1(\mathcal{C}) \cap \mathcal{X}$ (resp. $X \in \mathrm{vars}^2(\mathcal{C})$) there is a unique constraint $D \vdash^?_X x$ in $\mathcal{S}$.

For $X \in \mathrm{vars}^2(\mathcal{C})$, we write $D_{\mathcal{C}}(X)$ for the domain $D \subseteq \mathcal{W}$ of the deduction constraint $D \vdash^?_X x$ associated to $X$ in $\mathcal{C}$.

**Example 34** (Continuing Example 33)**.** *Let $\Phi = \Phi_0 \uplus \{w_3 \mapsto \mathsf{aenc}(\langle \pi_2(N), \langle n_b, \mathsf{pk}(skb)\rangle\rangle, \mathsf{pk}(ska))\}$ with $N = \mathsf{adec}(y, skb)$, and $\mathcal{S}$ be a set containing two constraints:*

$$\{w_0, w_1, w_2\} \vdash^?_Y y \text{ and } \pi_2(N) =^? \mathsf{pk}(ska).$$

*We have that $\mathcal{C} = (\Phi; \mathcal{S})$ is a well-formed constraint system. There is only one first-order variable $y \in \mathrm{vars}^1(\mathcal{C}) \cap \mathcal{X}$, and it does not occur in $\mathrm{vars}^1(\Phi(\{w_0, w_1, w_2\}))$, which is empty. Moreover, there is indeed a unique constraint that introduces $y$.*

Our notion of well-formed constraint systems is in line with what is used, *e.g.* in [MS01, Bau05]. We use a simpler variant here that is sufficient for our purpose.

**Definition 32** (solution)**.** *A* solution *of a constraint system $\mathcal{C} = (\Phi; \mathcal{S})$ is a substitution $\theta$ such that* $\mathrm{dom}(\theta) = \mathrm{vars}^2(\mathcal{C})$*, and $X\theta \in \mathcal{T}(\Sigma_{\mathsf{pub}}, D_{\mathcal{C}}(X))$ for any $X \in \mathrm{dom}(\theta)$. Moreover, we require that there exists a ground substitution $\lambda : \mathrm{vars}^1(\mathcal{C}) \to \mathcal{T}(\Sigma_c, \mathcal{N})$ such that:*

- *for every $D \vdash^?_X x$ in $\mathcal{S}$, we have $(X\theta)(\Phi\lambda) \Downarrow x\lambda$;*
- *for every $u =^? v$ in $\mathcal{S}$, we have $u\lambda \equiv_{\mathsf{E}_d} v\lambda$, $\mathsf{valid}(u\lambda)$, and $\mathsf{valid}(v\lambda)$; and*
- *for every $u \neq^? v$ in $\mathcal{S}$, we have $u\lambda \not\equiv_{\mathsf{E}_d} v\lambda$, or $\neg\mathsf{valid}(u\lambda)$, or $\neg\mathsf{valid}(v\lambda)$.*

*Moreover, we require that all the terms occurring in $\Phi\lambda$ are valid. The set of solutions of a constraint system $\mathcal{C}$ is denoted $\mathsf{Sol}(\mathcal{C})$. Since we consider constraint systems that are well-formed, the substitution $\lambda$ is unique modulo $=_{\mathsf{E}}$ given $\theta \in \mathsf{Sol}(\mathcal{C})$. We denote it by $\lambda_\theta$ when $\mathcal{C}$ is clear from the context.*

The validity condition on $\Phi\lambda$ ensures that all outputted terms actually compute messages. Indeed, we have by definition of $\Downarrow$ (see Section 6.1) that for any term $t$, $t \Downarrow u$ for some $u$ implies that $t$ must be valid. Similarly, an equality test holds only when it relates terms that compute the same message implying that the related terms must be valid as well.

**Example 35.** *Consider again the constraint system $\mathcal{C}$ given in Example 34. We have that $\theta = \{Y \mapsto \mathsf{aenc}(\langle w_1, w_1\rangle, w_2)\}$ is a solution of $\mathcal{C}$. Its associated first-order solution is $\lambda_\theta = \{y \mapsto \mathsf{aenc}(\langle \mathsf{pk}(ska), \mathsf{pk}(ska)\rangle, \mathsf{pk}(skb))\}$.*

#### Symbolic processes: syntax and semantics

Given a simple configuration $(\mathcal{P}; \Phi)$, we compute the constraint systems capturing its possible executions, starting from the symbolic configuration $(\mathcal{P}; \Phi; \emptyset)$. Note that we are now manipulating processes that are not ground anymore, but may contain free variables. For this reason, symbolic configurations are no longer subject to the internal reduction $\rightsquigarrow_{\mathscr{R}}$. Indeed, symbolic configurations represent different (concrete) configurations that could trigger different internal reduction rules. The semantics we eventually define features additional rules for conditionals and blocked outputs. They are the counterpart of internal reductions.

**Definition 33** (Symbolic Configuration)**.** *A symbolic configuration is a tuple $(\mathcal{P}; \Phi; \mathcal{S})$ where $(\Phi; \mathcal{S})$ is a constraint system, $\mathrm{vars}^1(\mathcal{P}) \subseteq (\mathrm{vars}^1(\mathcal{S}) \cap \mathcal{X})$ and $\mathcal{P}$ is a simple process.*

When considering a symbolic configuration $(\mathcal{P}; \Phi; \mathcal{S})$ and one of its solutions $\theta$, we may write $A = (\mathcal{P}\theta; \Phi\lambda_\theta \Downarrow)$ the (concrete) configuration associated that is not necessarily in normal form w.r.t. $\rightsquigarrow_{\mathscr{R}}$. The notation $\Phi\lambda_\theta \Downarrow$ denotes a frame $\Psi$ (we may also note $\Phi\lambda \Downarrow \Psi$) that is formally defined as follows: $\Psi = \{w \mapsto u \mid w \in \mathrm{dom}(\Phi), w\Phi \Downarrow u\}$. Note that computations never fail by definition of a solution (*i.e.* terms in $\Phi\lambda$ must be valid). Remark also that there may be more than one frame $\Psi$ such that $\Phi\lambda \Downarrow \Psi$ but they all are statically equivalent because equal modulo $=_{\mathsf{E}}$. When the choice of representative in $\sim$-equivalence class or in $=_{\mathsf{E}}$-class is unimportant, we

| | |
|---|---|
| In | $(\{\mathtt{in}(c,y).P\} \uplus \mathcal{P}; \Phi; \mathcal{S}) \overset{\mathtt{in}(c,X)}{\Longmapsto} (\{P\{y \mapsto x\}\} \uplus \mathcal{P}; \Phi; \mathcal{S} \cup \{\mathrm{dom}(\Phi) \vdash^?_X x\})$ where $X$ (resp. $x$) is a fresh second-order (resp. first-order) variable |
| Out | $(\{\mathtt{out}(c,t).P\} \uplus \mathcal{P}; \Phi; \mathcal{S}) \overset{\mathtt{out}(c,w)}{\Longmapsto} (\{P\} \uplus \mathcal{P}; \Phi \uplus \{w \mapsto t\}; \mathcal{S})$ where $w$ is a fresh variable |
| Then | $(\{\mathsf{if}\ t_1 = t_2\ \mathsf{then}\ P\ \mathsf{else}\ Q\} \uplus \mathcal{P}; \Phi; \mathcal{S}) \overset{\tau_{\mathsf{then}}}{\Longmapsto} (\{P\} \uplus \mathcal{P}; \Phi; \mathcal{S} \cup \{t_1 =^? t_2\})$ |
| Else | $(\{\mathsf{if}\ t_1 = t_2\ \mathsf{then}\ P\ \mathsf{else}\ Q\} \uplus \mathcal{P}; \Phi; \mathcal{S}) \overset{\tau_{\mathsf{else}}}{\Longmapsto} (\{Q\} \uplus \mathcal{P}; \Phi; \mathcal{S} \cup \{t_1 \neq^? t_2\})$ |
| Blocked-Out | $(\{\mathtt{out}(c,t).P\} \uplus \mathcal{P}; \Phi; \mathcal{S}) \overset{\tau_o}{\Longmapsto} (\{0\} \uplus \mathcal{P}; \Phi; \mathcal{S} \cup \{t \neq^? t\})$ |

**Figure 6.2** Symbolic semantics for symbolic configurations

may just write $\Phi\lambda \Downarrow$ without making the choice explicit. Finally, we explicitly write $\mathrm{NF}_{\mathscr{R}}(A)$ to refer to its normal form (*i.e.* $(\mathrm{NF}_{\mathscr{R}}(\mathcal{P}\theta); \Phi\lambda_\theta \Downarrow)$ where $\mathrm{NF}_{\mathscr{R}}$ has been defined in Chapter 2).

We give in Figure 6.2 the symbolic semantics for symbolic configurations. Remark the one-to-one mapping from symbolic rules (Figure 6.2) on one hand and concrete rules (Figure 6.1) and $\mathscr{R}_{\mathrm{test}}, \mathscr{R}_{\mathrm{out}}$ reduction rules (Definition 11 in Subsection 4.1.1) on the other hand. The additional rules Then,Else and Blocked-Out play the role of the internal reduction rules of $\mathscr{R}_{\mathrm{test}}, \mathscr{R}_{\mathrm{out}}$. Notably, Blocked-Out allows to symbolically remove blocked outputs adding a constraint (*i.e.* $t \neq^? t$) making sure that any instantiation of this output cannot be triggered because the underlying term do not reduce to a message or equivalently is not valid). Note that for a process $\mathtt{out}(c,t).P$ in a symbolic configuration, the rules Out and Blocked-Out cover mutually exclusive concrete executions: the former cover all instantiations for which $t$ is valid (since $\Phi\lambda_\theta$ must contain only valid terms) and the latter cover all instantiations for which $t$ is not valid (since $t \neq^? t$ holds iff $t$ is not valid).

The operator $\mathsf{obs}(\bullet)$ is extended to symbolic actions straightforwardly; *i.e.* it just removes all $\tau$ actions.

**Example 36** (Resuming Example 34)**.** *We have that* $(\{Q_0(\mathsf{sk}_B, \mathsf{pk}(\mathsf{sk}_A))\}; \Phi_0; \emptyset) \overset{\mathsf{tr}}{\Longmapsto} (\emptyset; \Phi; \mathcal{S})$ *where:*

- $\mathsf{tr} = \mathtt{in}(c_B, Y).\tau_{\mathsf{then}}.\mathtt{out}(c_B, w_3)$*, and*
- $\mathcal{C} = (\Phi; \mathcal{S})$ *is the constraint system defined in Example 34.*

We are now able to define the notion of equivalence associated to the symbolic semantics, namely symbolic trace equivalence (denoted $\approx^s$ ).

**Definition 34.** *Let* $A = (\mathcal{P}; \Phi)$ *and* $B = (\mathcal{Q}; \Psi)$ *be two simple configurations. We have that* $A \sqsubseteq^s B$ *when, for every trace* $\mathsf{tr}$ *such that* $(\mathcal{P}; \Phi; \emptyset) \overset{\mathsf{tr}}{\Longmapsto} (\mathcal{P}'; \Phi'; \mathcal{S}_A)$*, for every* $\theta \in \mathsf{Sol}(\Phi'; \mathcal{S}_A)$*, we have that:*

- $(\mathcal{Q}; \Psi; \emptyset) \overset{\mathsf{tr}'}{\Longmapsto} (\mathcal{Q}'; \Psi'; \mathcal{S}_B)$ *where* $\mathsf{obs}(\mathsf{tr}') = \mathsf{obs}(\mathsf{tr})$ *with* $\theta \in \mathsf{Sol}(\Psi'; \mathcal{S}_B)$*, and*

- $\Phi' \lambda_\theta^A \Downarrow \sim \Psi' \lambda_\theta^B \Downarrow$ *where* $\lambda_\theta^A$ *(resp.* $\lambda_\theta^B$*) is the substitution associated to* $\theta$ *w.r.t.* $(\Phi'; \mathcal{S}_A)$ *(resp.* $(\Psi'; \mathcal{S}_B)$*).*

*We have that* $A$ *and* $B$ *are in* trace equivalence w.r.t. $\Rightarrow$*, denoted* $A \approx^s B$*, if* $A \sqsubseteq^s B$ *and* $B \sqsubseteq^s A$*.*

**Example 37.** *We have that* $(\{Q_0(\mathsf{sk}_B, \mathsf{pk}(\mathsf{sk}_A))\}; \Phi_0) \not\sqsubseteq^s (\{Q_0(\mathsf{sk}_B, \mathsf{pk}(\mathsf{sk}'_A))\}; \Phi_0)$*. Continuing Example 36, we have seen that:*

- $(\{Q_0(\mathsf{sk}_B, \mathsf{pk}(\mathsf{sk}_A))\}; \Phi_0; \emptyset) \overset{\mathsf{tr}}{\Longrightarrow} (\emptyset; \Phi; \mathcal{S})$*, and*
- $\theta = \{Y \mapsto \mathsf{aenc}(\langle w_1, w_1 \rangle, w_2)\} \in \mathsf{Sol}(\Phi; \mathcal{S})$ *(see examples 34 and 35).*

*The only symbolic process that is reachable from* $(\{Q_0(\mathsf{sk}_B, \mathsf{pk}(\mathsf{sk}'_A))\}; \Phi_0; \emptyset)$ *using* $\mathsf{tr}$ *(up to* $\mathsf{obs}(\bullet)$*) is* $(\emptyset; \Phi'; \mathcal{S}')$ *with:*

- $\Phi' = \Phi_0 \uplus \{w_3 \mapsto \mathsf{aenc}(\langle \pi_2(N), \langle n_b, \mathsf{pk}(skb) \rangle \rangle, \mathsf{pk}(ska'))\}$*, and*
- $\mathcal{S}' = \left\{ \{w_0, w_1, w_2\} \vdash^?_Y y,\ \pi_2(N) =^? \mathsf{pk}(ska') \right\}$*.*

*One can check that* $\theta$ *is not a solution of* $(\Phi'; \mathcal{S}')$*.*

For processes without replication, the symbolic transition system induced by $\Rightarrow$ is essentially finite. Indeed, the choice of fresh names for handles and second-order variables does not matter, and therefore the relation $\Rightarrow$ is essentially finitely branching. Moreover, the length of traces of a simple configuration is obviously bounded. Thus, deciding (symbolic) trace equivalence between processes boils down to the problem of deciding a notion of equivalence between sets of constraint systems. This problem is well-studied and several procedures already exist [Bau05, CR12], *e.g.* Apte [CCD11] (see Section 6.3).

#### Soundness and completeness

It is well-known that such symbolic semantics are sound and complete w.r.t. concrete semantics, and therefore that they capture the same notion of trace equivalence. This has been proved for instance in [Bau05, CCD13a]. Using the same approach, we can show soundness and completeness of the symbolic semantics (Figure 6.2) w.r.t. the concrete semantics (Figure 6.1). We have:

- *Soundness*: each transition in the symbolic semantics represents a set of transitions that can be done in the concrete semantics.
- *Completeness*: each transition in the concrete semantics can be matched by a transition in the symbolic semantics.

These results are formally expressed in propositions 14 and 15 below. Note that the mapping between the two semantics is up to $\sim_{\mathscr{R}}$ and computations on frames.

**Proposition 14.** *Let $(\mathcal{P};\Phi)$ be a simple configuration such that $(\mathcal{P};\Phi;\emptyset) \overset{\mathsf{tr}}{\Longmapsto} (\mathcal{P}_s;\Phi_s;\mathcal{S}_s)$, and $\theta \in \mathsf{Sol}(\Phi_s;\mathcal{S}_s)$. We have that $(\mathcal{P};\Phi) \xrightarrow{\mathsf{obs}(\mathsf{tr}\theta)} (\mathrm{NF}_{\mathscr{R}}(\mathcal{P}_s\lambda);\Phi')$ where $\Phi_s\lambda \Downarrow \Phi'$ for $\lambda$ the first-order solution of $(\mathcal{P}_s;\Phi_s;\mathcal{S}_s)$ associated to $\theta$.*

*Proof.* We proceed by induction on $\mathsf{tr}$. If $\mathsf{tr} = \epsilon$, then we immediately conclude because $\theta = \lambda = \emptyset$, $\mathcal{P}_s = \mathcal{P}$ and $\Phi' = \Phi$ so an empty concrete execution suits. If $\mathsf{tr} = \mathsf{tr}^0.\alpha$, then we have $(\mathcal{P};\Phi;\emptyset) \overset{\mathsf{tr}^0}{\Longmapsto} (\mathcal{P}^0_s;\Phi^0_s;\mathcal{S}^0) \overset{\alpha}{\Longmapsto} (\mathcal{P}_s;\Phi_s;\mathcal{S})$ with $\theta \in \mathsf{Sol}(\Phi_s;\mathcal{S})$. We note $\lambda$ the associated first-order solution. We also have that $\theta^0 = \theta_{|\mathrm{vars}^2(\mathcal{S}^0)} \in \mathsf{Sol}(\Phi^0_s;\mathcal{S}^0)$ with $\lambda^0 = \lambda_{|\mathrm{vars}^1(\mathcal{S}^0)}$ the associated first-order solution. By inductive hypothesis, we thus obtain an execution $(\mathcal{P};\Phi) \xrightarrow{\mathsf{obs}(\mathsf{tr}^0\theta^0)} (\mathrm{NF}_{\mathscr{R}}(\mathcal{P}^0_s\lambda^0);\Phi^0)$ where $\Phi^0_s\lambda^0 \Downarrow \Phi^0$ and $\lambda^0$ is the first-order solution of $(\mathcal{P}^0_s;\Phi^0_s;\mathcal{S}^0_s)$ associated to $\theta^0$. We now reason by case analysis on $\alpha$.

*Case* $\alpha = \mathtt{out}(c,w)$. Then, there exists a process $P_s = \mathtt{out}(c,t).P' \in \mathcal{P}^0_s$. More precisely, we have that $\mathcal{P}^0_s = \{P_s\} \uplus \mathcal{P}'_s$, $\mathcal{P}_s = \{P'\} \uplus \mathcal{P}'_s$, and $\Phi_s = \Phi^0_s \uplus \{w \mapsto t\}$. Since $\theta^0$ is a solution with $\lambda^0$ as the associated first-order solution, it holds that $t\lambda^0$ is valid and thus $P_s\lambda^0 \in \mathcal{P}^0_s\lambda^0$ is not "blocked" and cannot be removed by the internal reduction. Hence $P_s\lambda^0 \in \mathrm{NF}_{\mathscr{R}}(\mathcal{P}^0_s\lambda^0)$. We thus have a decomposition $\mathrm{NF}_{\mathscr{R}}(\mathcal{P}^0_s\lambda^0) = \mathrm{NF}_{\mathscr{R}}(\{\mathcal{P}'_s\lambda^0\}) \uplus \{P_s\lambda^0\}$. By validity, there must be a message $u$ such that $t\lambda^0 \Downarrow u$. Finally, we can construct an execution $(\mathrm{NF}_{\mathscr{R}}(\mathcal{P}^0_s\lambda^0);\Phi^0) \xrightarrow{\mathtt{out}(c,w)} (\mathrm{NF}_{\mathscr{R}}(\{P'\lambda^0\}) \uplus \mathrm{NF}_{\mathscr{R}}(\mathcal{P}'_s\lambda^0);\Phi^0 \uplus \{w \mapsto u\})$. This can be done using $\lambda$ instead of $\lambda^0$ and one can easily verify that the resulting concrete execution satisfies all requirements.

*Case* $\alpha = \mathtt{in}(c,X)$. This case can be proved similarly to the output case.

*Case $\alpha$ is unobservable.* In such cases, we remark that $\mathrm{NF}_{\mathscr{R}}(\mathcal{P}_s\lambda) = \mathrm{NF}_{\mathscr{R}}(\mathcal{P}^0_s\lambda)$ and $\Phi_s = \Phi^0_s$. Thus, the execution given by the inductive hypothesis allows to conclude. □

**Proposition 15.** *Let $(\mathcal{P};\Phi)$ be a simple configuration such that $(\mathcal{P};\Phi) \xrightarrow{\mathsf{tr}} (\mathcal{P}';\Phi')$. There exists a symbolic configuration $(\mathcal{P}_s;\Phi_s;\mathcal{S})$, a solution $\theta \in \mathsf{Sol}(\Phi_s;\mathcal{S})$, and a sequence $\mathsf{tr}_s$ such that:*

- $(\mathcal{P};\Phi;\emptyset) \overset{\mathsf{tr}_s}{\Longmapsto} (\mathcal{P}_s;\Phi_s;\mathcal{S})$;
- $\mathcal{P}' = \mathcal{P}_s\lambda$, $\Phi_s\lambda \Downarrow \Phi'$; *and*
- $\mathsf{tr} = \mathsf{obs}(\mathsf{tr}_s\theta)$

*where $\lambda$ is the first-order solution of $(\mathcal{P}_s;\Phi_s;\mathcal{S})$ associated to $\theta$.*

*Proof.* We proceed by induction on $\mathsf{tr}$. If $\mathsf{tr} = \epsilon$ then we we immediately conclude by letting $\theta = \lambda = \emptyset$, $\mathcal{P}_s = \mathcal{P}$, $\mathsf{tr}_s = \epsilon$, $\Phi_s = \Phi$, and, $\mathcal{S} = \emptyset$. If $\mathsf{tr} = \mathsf{tr}^0.\alpha$ then we have $(\mathcal{P};\Phi) \xrightarrow{\mathsf{tr}^0} (\mathcal{P}^0;\Phi^0) \xrightarrow{\alpha} (\mathcal{P}';\Phi')$. By inductive hypothesis, there must be a symbolic configuration $(\mathcal{P}^0_s;\Phi^0_s;\mathcal{S}^0)$ and a solution $\theta^0 \in \mathsf{Sol}(\Phi_s;\mathcal{S}^0)$ (with $\lambda^0$ the associated first-order solution) such that $(\mathcal{P};\Phi;\emptyset) \overset{\mathsf{tr}^0_s}{\Longmapsto} (\mathcal{P}^0_s;\Phi^0_s;\mathcal{S}^0)$, $\mathcal{P}^0 = \mathcal{P}^0_s\lambda^0$, $\Phi^0_s\lambda^0 \Downarrow \Phi'$, and, $\mathsf{obs}(\mathsf{tr}^0_s\theta^0) = \mathsf{tr}^0$. We now reason by case analysis on $\alpha$.

*If* $\alpha = \mathtt{out}(c,w)$. In that case, there must be a process $P = \mathtt{out}(c,t).P' \in \mathcal{P}_0$ and $t$ must be valid. We thus also have a process $P_s = \mathtt{out}(c,t_s).P'_s \in \mathcal{P}^0_s$ and $t_s\lambda^0$ is valid. More precisely, one has the following decomposition: $\mathcal{P}^0_s = \{P_s\} \uplus \mathcal{P}'_s$. It is thus possible to extend the given symbolic execution: $(\mathcal{P}^0_s;\Phi^0_s;\mathcal{S}^0) \overset{\mathtt{out}(c,w)}{\Longmapsto} (\{P'\} \uplus \mathcal{P}'_s;\Phi^0_s \uplus \{w \mapsto t_s\};\mathcal{S}^0) = K_s$. Remark that

$\theta^0$ (along with $\lambda^0$) is also a solution of $K_s$ since $t_s\lambda^0$ is valid. Further, we need to mimic the internal reduction by executing $\tau$-actions from $K_s$ (indeed, we do not necessarily have that $\mathcal{P}' = (\{P'\} \uplus \mathcal{P}'_s)\lambda^0$). More precisely, we trigger all available Then, Else, Blocked-Out rules as long as they produce new constraint systems of which $\theta^0$ is still a solution. One can easily prove that the resulting execution satisfies all requirements.

*If* $\alpha = \mathtt{in}(c, X)$. This case can be proved similarly to the previous case. □

Finally, relying on these two results, we can establish that symbolic trace equivalence ($\approx^s$) exactly captures trace equivalence ($\approx$). Actually, both inclusions can be established separately.

**Proposition 16.** *For any simple configurations $A$ and $B$, we have that:*

$$A \sqsubseteq B \iff A \sqsubseteq^s B.$$

*Proof.* We consider two simple configurations $A = (\mathcal{P}; \Phi)$ and $B = (\mathcal{Q}; \Psi)$. We prove the two directions separately.

($\Rightarrow$) Let $(\mathcal{P}; \Phi; \emptyset) \overset{\mathsf{tr}_s}{\Longmapsto} (\mathcal{P}_s; \Phi_s; \mathcal{S}_s)$ and $\theta \in \mathsf{Sol}(\Phi_s; \mathcal{S}_s)$. Applying Proposition 14, we obtain an execution $A \xrightarrow{\mathsf{obs}(\mathsf{tr}_s\theta)} (\mathcal{P}'; \Phi')$ for some $\Phi'$ such that $\Phi_s\lambda \Downarrow \Phi'$. By hypothesis, we deduce $B \xrightarrow{\mathsf{obs}(\mathsf{tr}_s\theta)} (\mathcal{Q}'; \Psi')$ with $\Psi' \sim \Phi' \sim \Phi_s\lambda \Downarrow$. Now, Proposition 15 implies an execution $(\mathcal{Q}; \Psi; \emptyset) \overset{\mathsf{tr}'_s}{\Longmapsto} (\mathcal{Q}_s; \Psi_s; \mathcal{S}^B_s)$ and a solution $\theta^B \in \mathsf{Sol}(\Psi_s; \mathcal{S}^B_s)$ such that $\mathsf{obs}(\mathsf{tr}_s\theta) = \mathsf{obs}(\mathsf{tr}'_s\theta^B)$ and $\Psi_s\lambda^B \Downarrow \Psi'$ (where $\lambda^B$ is the first-order solution associated to $\theta^B$ and $(\Psi_s, \mathcal{S}^B_s)$). Remark that, since the choice of fresh second-order variable is irrelevant, we can w.l.o.g. and up to a bijection of second-order variables choose the ones in $\mathsf{tr}'_s$ in such a way that $\mathsf{obs}(\mathsf{tr}'_s) = \mathsf{obs}(\mathsf{tr}_s)$ (remind that $\mathsf{obs}(\mathsf{tr}_s\theta) = \mathsf{obs}(\mathsf{tr}'_s\theta^B)$ so they may differ only on the second order variables). A quick inspection at the notion of solutions and at the only rule from the symbolic semantics which add second-order-variables to set of constraints shows that

$$\mathrm{dom}(\theta^B) = \mathrm{vars}^2((\mathcal{S}^B_s; \Psi_s)) = \mathrm{vars}^2(\mathsf{tr}'_s) = \mathrm{vars}^2(\mathsf{tr}_s) = \mathrm{vars}^2((\mathcal{S}_s; \Phi_s)) = \mathrm{dom}(\theta).$$

Further, for each $X \in \mathrm{dom}(\theta)$, $X\theta$ can be found as a recipe in $\mathsf{obs}(\mathsf{tr}_s\theta)$. Since $\mathsf{obs}(\mathsf{tr}_s\theta) = \mathsf{obs}(\mathsf{tr}'_s\theta^B)$, it holds that $X\theta = X\theta^B$ and thus $\theta = \theta^B$ concluding the proof of this direction.

($\Leftarrow$) Let $(\mathcal{P}; \Phi) \xrightarrow{\mathsf{tr}} (\mathcal{P}'; \Phi')$. Applying Proposition 15, we obtain a symbolic execution $(\mathcal{P}; \Phi; \emptyset) \overset{\mathsf{tr}_s}{\Longmapsto} (\mathcal{P}_s; \Phi_s; \mathcal{S}_s)$ and $\theta \in \mathsf{Sol}(\Phi_s; \mathcal{S}_s)$ such that $\Phi_s\lambda \Downarrow \Phi'$ and $\mathsf{obs}(\mathsf{tr}_s\theta) = \mathsf{tr}$. By hypothesis, we obtain an execution $(\mathcal{Q}; \Psi; \emptyset) \overset{\mathsf{tr}'_s}{\Longmapsto} (\mathcal{Q}_s; \Psi_s; \mathcal{S}^B_s)$ such that $\mathsf{obs}(\mathsf{tr}_s) = \mathsf{obs}(\mathsf{tr}'_s)$, $\theta \in \mathsf{Sol}(\Psi_S, \mathcal{S}^B_s)$ and $\Psi_s\lambda^B_\theta \Downarrow \sim \Phi_s\lambda \Downarrow$. Next, we apply Proposition 14 and obtain $B \xrightarrow{\mathsf{obs}(\mathsf{tr}'_s\theta)} (\mathcal{Q}'; \Psi')$ for some $\Psi'$ such that $\Psi_s\lambda^B_\theta \Downarrow \Psi'$. We thus have $\Psi' \sim \Psi_s\lambda^B_\theta \Downarrow \sim \Phi_s\lambda \Downarrow \sim \Phi'$. Finally, remark that $\mathsf{obs}(\mathsf{tr}'_s\theta) = \mathsf{obs}(\mathsf{tr}'_s)\theta = \mathsf{obs}(\mathsf{tr}_s)\theta = \mathsf{obs}(\mathsf{tr}_s\theta) = \mathsf{tr}$. □

### 6.2.2 Embedding Compression into Symbolic Semantics

In order to define the symbolic, compressed semantics, we proceed as we did for the concrete semantics in Chapter 5. There is no fundamental difficulty in doing so since compression only leverages syntactical information (*i.e.* nature of available observable actions) that is similarly

available at the symbolic level. The only technical difficulty arises from unobservable actions that were treated using internal reduction and are now treated using unobservable actions.

We shall start with the symbolic semantics with unobservable actions. Formally, we define $\overset{\bullet}{\mapsto}$ as follows: $A \overset{\mathsf{tr}}{\overset{\bullet}{\mapsto}} B$ when $A \overset{\mathsf{tr}'}{\Longmapsto} B$ and $\mathsf{obs}(\mathsf{tr}') = \mathsf{tr}$. We shall state parts of the next results only for configurations with no conditional at top-level as they can be evaluated transparently with $\overset{\bullet}{\mapsto}$. We formally define such configurations below.

**Definition 35.** *A basic process $P$ is said to be* quiescent *when it starts with either an input (*i.e. *$P = \mathtt{in}(c, x)$ for some $c, x, P$) or with an output (*i.e. *$P = \mathtt{out}(c, t).Q$ for some $c, t, Q$). A simple process is said to be* quiescent *when all of its processes are quiescent. This is lifted to symbolic configurations in the obvious way.*

### Annotated Symbolic Semantics

One can define the annotated semantics by building on $\overset{\bullet}{\mapsto}$ as it was defined by building on $\overset{\bullet}{\to}$ in Chapter 5. The notions of skeletons and annotations are extended straightforwardly to respectively symbolic, quiescent processes and symbolic actions of $\overset{\bullet}{\mapsto}$ (*i.e.* of the form $\mathtt{in}(c, X), \mathtt{out}(c, w)$). Note that we do not define skeletons for processes of the form $\mathsf{if}\ t_1 = t_2\ \mathsf{then}\ P\ \mathsf{else}\ Q$ (*i.e.* non-quiescent processes) as we will require that such conditionals are executed before other observable actions.

We now equip the symbolic semantics (Figure 6.2) with annotations as it was done in Chapter 5 (see Figure 5.2). However, doing so for the symbolic configurations is simpler since they do not feature parallel composition nor replication. Labels are thus just handed over and extended with a $\boxdot$ from processes to their continuations. The resulting semantics is depicted in Figure 6.3. Labels on processes of the form $\mathsf{if}\ t_1 = t_2\ \mathsf{then}\ P\ \mathsf{else}\ Q$ are handed over to $P$ or $Q$ when evaluating the conditional.

$$\mathrm{Act}_\alpha \quad \frac{(\{P\}; \Phi; \mathcal{S}) \overset{\alpha}{\mapsto} (\{P'\}; \Phi'; \mathcal{S}')}{(\{[P]^\ell\} \uplus \mathcal{P}; \Phi; \mathcal{S}) \overset{\lceil\alpha\rceil^\ell}{\longmapsto}_a (\{[P']^{\ell \cdot \boxdot}\} \uplus \mathcal{P}; \Phi'; \mathcal{S}')} \ \alpha \in \{\mathtt{in}(\bullet, \bullet), \mathtt{out}(\bullet, \bullet)\}$$

$$\mathsf{Zero} \quad (\{[0]^\ell\} \uplus \mathcal{P}; \Phi; \mathcal{S}) \overset{\lceil\mathtt{zero}\rceil^\ell}{\longmapsto}_a (\mathcal{P}; \Phi; \mathcal{S})$$

**Figure 6.3** Annotated symbolic semantics for symbolic configurations

### Compression Strategy

As done before for configurations, we enrich symbolic configurations with a focus position. The symbolic, compressed semantics deals with enriched symbolic configurations of the form $(\mathcal{P}; F; \Phi; \mathcal{S})$ that denotes a symbolic configuration $(\mathcal{P}'; \Phi; \mathcal{S})$ (where $\mathcal{P}' = \mathcal{P} \uplus \{F\}$ when $F$ is a process and $\mathcal{P}' = \mathcal{P}$ otherwise) with (possibly) a distinguished process $F$ that is said to be *under focus*. We may write $\mathsf{Sol}(A)$ for an enriched symbolic configuration $A = (\mathcal{P}; F; \Phi; \mathcal{S})$ to denote the set $\mathsf{Sol}(\Phi; \mathcal{S})$. Operators $\lceil\bullet\rceil$ and $\lfloor\bullet\rfloor$ are lifted in the obvious way:

$$\lceil(\mathcal{P}; \Phi; \mathcal{S})\rceil = (\mathcal{P}; \varnothing; \Phi; \mathcal{S}),\ \lfloor(\mathcal{P}; \varnothing; \Phi; \mathcal{S})\rfloor = (\mathcal{P}; \Phi; \mathcal{S}) \text{ and } \lfloor(\mathcal{P}; F; \Phi; \mathcal{S})\rfloor = (\mathcal{P} \uplus \{F\}; \Phi; \mathcal{S}).$$

$$\text{Start/In} \quad \frac{\mathcal{P} \text{ is initial} \quad (P;\Phi;\mathcal{S}) \overset{\mathtt{in}(c,M)}{\longmapsto}_a (P';\Phi;\mathcal{S}')}{(\mathcal{P} \uplus \{P\};\varnothing;\Phi;\mathcal{S}) \overset{\mathtt{foc}(\mathtt{in}(c,M))}{\longmapsto}_c (\mathcal{P};P';\Phi;\mathcal{S}')}$$

$$\text{Pos/In} \quad \frac{(P;\Phi;\mathcal{S}) \overset{\mathtt{in}(c,M)}{\longmapsto}_a (P';\Phi;\mathcal{S}')}{(\mathcal{P};P;\Phi;\mathcal{S}) \overset{\mathtt{in}(c,M)}{\longmapsto}_c (\mathcal{P};P';\Phi;\mathcal{S}')}$$

$$\text{Neg} \quad \frac{\mathcal{P},P \text{ are quiescent} \quad (P;\Phi;\mathcal{S}) \overset{\alpha}{\longmapsto}_a (\mathcal{P}';\Phi';\mathcal{S}')}{(\mathcal{P} \uplus \{P\};\varnothing;\Phi;\mathcal{S}) \overset{\alpha}{\longmapsto}_c (\mathcal{P} \uplus \mathcal{P}';\varnothing;\Phi';\mathcal{S}')} \ \alpha \in \{\mathtt{zero}, \mathtt{out}(\bullet,\bullet)\}$$

$$\text{Release} \quad (\mathcal{P};[\mathtt{out}(c,t).P]^\ell;\Phi;\mathcal{S}) \overset{[\mathtt{rel}]^\ell}{\longmapsto}_c (\{[\mathtt{out}(c,t).P]^\ell\} \uplus \mathcal{P};\varnothing;\Phi;\mathcal{S} \uplus \{t =^? t\})$$

$$\text{Release}_\perp \quad (\mathcal{P};[0]^\ell;\Phi;\mathcal{S}) \overset{[\mathtt{rel}]^\ell.[\mathtt{zero}]^\ell}{\longmapsto}_c (\emptyset;\varnothing;\Phi;\mathcal{S})$$

Labels are implicitly set in the same way as in the annotated semantics. Neg is made non-branching by imposing an arbitrary order on labelled skeletons of available actions.

**Figure 6.4** Symbolic compressed semantics

Definitions of positive processes and initial configurations are lifted to symbolic processes and symbolic multiset of processes straightforwardly. In particular, non-quiescent processes are not positive (conditionals at top level shall be executed in the next negative phase) and non-quiescent multisets of symbolic processes are not initial. We can now derive the compressed symbolic semantics $\overset{\mathsf{tr}}{\longmapsto}_c$ following the same pattern as for the concrete semantics leading to the rules depicted in Figure 6.4. The only differences are in rules Neg and Release. In Neg, we explicitly require that the underlying configuration is quiescent. This is in line with the concrete, compressed semantics for which configurations are necessarily quiescent since they are in normal form. This guard condition (*i.e.* the configuration should be quiescent) is needed to make the Neg rule non-branching based on an arbitrary order on labelled skeletons of available actions. In Release, we make sure that the output that popped out in the focused position is not blocked by adding the $t =^? t$ constraint. By doing so, we are sure that the two ways to release a focus through rules Release and Release$_\perp$ are mutually exclusive: either the negative process in the focused position is a non-blocked output (that can be triggered using Release) or it is a blocked output (it can be triggered using the rules Blocked-Out and then Release$_\perp$) or it is a null process (it can directly be triggered using Release).

We derive similarly the notion of trace equivalence induced by $\mapsto_c$. We do not have to take care of the $\tau$ actions since they are performed implicitly in the compressed semantics.

**Definition 36.** *Let $A = (\mathcal{P};F_A;\Phi)$ and $B = (\mathcal{Q};F_B;\Psi)$ be two simple configurations. We have that $A \sqsubseteq_c^s B$ when, for every trace $\mathsf{tr}$ such that $(\mathcal{P};F_A;\Phi;\emptyset) \overset{\mathsf{tr}}{\longmapsto}_c A'$, for every $\theta \in \mathsf{Sol}(A')$, we have that:*

- *$(\mathcal{Q};F_B;\Psi;\emptyset) \overset{\mathsf{tr}}{\longmapsto}_c B'$ with $\theta \in \mathsf{Sol}(B')$, and*
- *$\Phi(A')\lambda_\theta^A \Downarrow \sim \Phi(B')\lambda_\theta^B \Downarrow$ where $\lambda_\theta^A$ (resp. $\lambda_\theta^B$) is the substitution associated to $\theta$ w.r.t. $A$ (resp. $B$).*

*We have that $A$ and $B$ are in* trace equivalence w.r.t. $\mapsto_c$*, denoted $A \approx_c^s B$, if $A \sqsubseteq_c^s B$ and $B \sqsubseteq_c^s A$.*

**Example 38.** *We have that* $\lceil(\{Q_0(\mathsf{sk}_B, \mathsf{pk}(\mathsf{sk}_A))\}; \Phi_0)\rceil \not\sqsubseteq_c^s \lceil(\{Q_0(\mathsf{sk}_B, \mathsf{pk}(\mathsf{sk}'_A))\}; \Phi_0)\rceil$. *Continuing Example 36, we have that:*

- $(\{Q_0(\mathsf{sk}_B, \mathsf{pk}(\mathsf{sk}_A))\}; \varnothing; \Phi_0; \emptyset) \xmapsto{\mathsf{tr}_c}_c (\emptyset; \varnothing; \Phi; \mathcal{S})$ *where* $\mathsf{obs}(\mathsf{tr}_c) = \mathsf{obs}(\mathsf{tr})$, *and*
- $\theta \in \mathsf{Sol}(\Phi; \mathcal{S})$ *(see Example 34).*

*The only symbolic process that is reachable from* $(\{Q_0(\mathsf{sk}_B, \mathsf{pk}(\mathsf{sk}'_A))\}; \varnothing; \Phi_0; \emptyset)$ *using* $\mathsf{obs}(\mathsf{tr})$ *is* $(\emptyset; \varnothing; \Phi'; \mathcal{S}')$ *with:*

- $\Phi' = \Phi_0 \uplus \{w_3 \mapsto \mathsf{aenc}(\langle \pi_2(N), \langle n_b, \mathsf{pk}(skb)\rangle\rangle, \mathsf{pk}(ska'))\}$, *and*
- $\mathcal{S}' = \left\{\{w_0, w_1, w_2\} \vdash^?_Y y;\ \ \pi_2(N) =^? \mathsf{pk}(ska')\right\}$.

*One can check that* $\theta$ *is not a solution of* $(\Phi'; \mathcal{S}')$.

#### Soundness and Completeness

To prove that the symbolic, compressed trace equivalence coincides with the trace equivalence, we follow the same proof technique as the one developed in Section 6.2 when proving that the trace equivalence coincides with the symbolic trace equivalence (Proposition 16).

Thus, we show below (Theorem 4) that the compressed trace equivalence coincides with the compressed trace equivalence relying on the mappings between the symbolic, compressed semantics and the compressed semantics. Both directions of this mapping (propositions 17 and 18) follow from the already established mappings between symbolic executions and concrete executions (see propositions 14 and 15) and the fact that the compressed, symbolic strategy corresponds exactly to the compressed strategy (both exploration strategies are identical and both rely on syntactical information about processes equally available at the symbolic level or at the regular level).

**Proposition 17.** *Let* $(\mathcal{P}; \Phi)$ *be a simple configuration such that* $\lceil(\mathcal{P}; \Phi; \emptyset)\rceil \xmapsto{\mathsf{tr}}_c (\mathcal{P}_s; F_A; \Phi_s; \mathcal{S}_s)$, *and* $\theta \in \mathsf{Sol}(\Phi_s; \mathcal{S}_s)$. *We have that* $\lceil(\mathcal{P}; \Phi)\rceil \xrightarrow{\mathsf{tr}\theta}_c (\mathrm{NF}_{\mathscr{R}}(\mathcal{P}_s\lambda); F'; \Phi')$ *where* $F' = \mathrm{NF}_{\mathscr{R}}(F\lambda)$ *if* $F$ *is a process and* $F' = \varnothing$ *otherwise,* $\lambda$ *is the first-order solution of* $(\Phi_s; \mathcal{S}_s)$ *associated to* $\theta$, *and,* $\Phi'$ *is such that* $\Phi_s\lambda \Downarrow \Phi'$.

**Proposition 18.** *Let* $(\mathcal{P}; \Phi)$ *be a simple configuration such that* $\lceil(\mathcal{P}; \Phi)\rceil \xrightarrow{\mathsf{tr}}_c (\mathcal{P}'; F'; \Phi')$. *There exists a symbolic configuration* $(\mathcal{P}_s; F_s; \Phi_s; \mathcal{S})$, *a solution* $\theta \in \mathsf{Sol}(\Phi_s; \mathcal{S})$, *and a sequence* $\mathsf{tr}_s$ *such that:*

- $\lceil(\mathcal{P}; \Phi; \emptyset)\rceil \xmapsto{\mathsf{tr}_s}_c (\mathcal{P}_s; F_s; \Phi_s; \mathcal{S})$;
- $\mathcal{P}' = \mathcal{P}_s\lambda$, $F' = F_s\lambda$, $\Phi_s\lambda \Downarrow \Phi'$; *and*
- $\mathsf{tr} = \mathsf{tr}_s\theta$

*where* $\lambda$ *is the first-order solution of* $(\Phi_s; \mathcal{S})$ *associated to* $\theta$.

Applying exactly the same proof of Proposition 16 but invoking propositions 17 and 18 instead of propositions 14 and 15, we obtain the following theorem.

**Theorem 4.** *For any simple configurations $A$ and $B$, we have that:*

$$\lceil A \rceil \sqsubseteq_c \lceil B \rceil \iff \lceil A \rceil \sqsubseteq_c^s \lceil B \rceil.$$

As an immediate consequence of theorems 2 and 4, we obtain that the relations $\approx$ and $\approx_c^s$ coincide.

**Corollary 4.** *For any simple processes $A$ and $B$ such that $\mathsf{skl}(A) = \mathsf{skl}(B)$, we have that:*

$$A \approx B \iff \lceil A \rceil \approx_c^s \lceil B \rceil.$$

### 6.2.3 Embedding Reduction into Symbolic Semantics

Unlike compression, which is essentially based on the input/output nature of actions, the reduction takes into account the exchanged messages and considers the notion of *necessity* introduced in Section 5.4. Remind that the reduced semantics can be seen as the compressed semantics with an additional constraint on the explored blocks: a block $b$ can be explored after a trace $\mathsf{tr}$ only if, each block $b'$ in $\mathsf{tr}$ that has less priority over $b$ (violating the priority order), strong data dependencies forbid us to swap $b$ before $b'$ (to comply with that priority order). Formally, this is when $b$ is *authorised* after $\mathsf{tr}$ (see Definition 28). The idea for lifting this to the symbolic semantics consists in replacing the *authorised* predicate by some additional constraints.

#### Priority Order

Remind that, in Section 5.4, we defined the priority order $<$ as an order over blocks that is insensitive to recipes and handles. Since, we only deal with simple (symbolic) configurations in this chapter, we can refine this notion and let $<$ be an order over channels. This is lifted to blocks by comparing the (single) channel of each block.

**Example 39.** *We consider the symbolic version of Example 27. Let us consider the configuration $A = (\{P_1, P_2\}; \varnothing; \Phi_0; \emptyset)$ where $\Phi_0 = \{w_0 \mapsto n\}$ and $P_i$ are defined in Example 27. We consider a priority order such that $c_1 < c_2$ (*i.e. *blocks produced by $P_1$ have priority over blocks produced by $P_2$). We now examine the two following symbolic, compressed traces: $\mathsf{tr}_1 = b_1.b_2$ and $\mathsf{tr}_2 = b_2.b_1$ where $b_1 = \mathtt{foc}(\mathtt{in}(c_1, X_1)).\mathtt{rel}.\mathtt{out}(c_1, w_1)$ and $b_2 = \mathtt{foc}(\mathtt{in}(c_2, X_2)).\mathtt{rel}.\mathtt{out}(c_2, w_2)$ for some second-order variables $X_1, X_2$. The two symbolic configurations resulting from the compressed executions of $\mathsf{tr}_1$ and $\mathsf{tr}_2$ are of the form $(\{P_1', P_2'\}; \varnothing; \Phi; \mathcal{S}_i)$ where $\Phi = \Phi_0 \cup \{w_1 \mapsto n_1, w_2 \mapsto n_2\}$,*

$$\mathcal{S}_1 = \left\{\{w_0\} \vdash^?_{X_1} x_1;\ \{w_0, w_1\} \vdash^?_{X_2} x_2\right\},\ \text{and}\ \mathcal{S}_2 = \left\{\{w_0\} \vdash^?_{X_2} x_2;\ \{w_0, w_2\} \vdash^?_{X_1} x_1\right\}.$$

*Remind that the sets of concrete processes that these two symbolic processes represent are different and some deeply rely on the specific order of blocks, which means that we cannot discard any of*

*those symbolic, compressed traces. However, these sets have a significant overlap corresponding to concrete instances of the interleaved blocks that are actually independent,* i.e. *where the output of one block is not necessary to obtain the input of the next block. This problem has been discussed in Example 27 and depicted in Figure 5.4 (both in Section 5.4). In order to avoid considering such concrete processes twice, we may add a* dependency constraint *"$X_1 \ltimes w_2$" in $\mathcal{C}_2$, whose purpose is to discard all solutions $\theta$ such that the message $x_1\lambda_\theta$ can be derived without using $w_2 \mapsto n_2$. For instance, the concrete trace $\mathtt{foc}(\mathtt{in}(c_2, w_0)).\mathtt{rel}.\mathtt{out}(c_2, w_2).\mathtt{foc}(\mathtt{in}(c_1, w_0)).\mathtt{rel}.\mathtt{out}(c_1, w_1)$ would be discarded thanks to this new constraint. The constraint "$X_1 \ltimes w_2$" would be the symbolic counterpart of the conditions imposed by the authorised predicate $(b_2, \Phi) \blacktriangleright b_1$.*

### Dependency Constraints

We now introduce a new kind of constraints that we call *dependency constraints*. They will be used to represent the *authorised* predicate at the symbolic level.

**Definition 37.** *A* dependency constraint *is a constraint of the form $\overline{X} \ltimes \overline{w}$ where $\overline{X}$ is a vector of second-order variables in $\mathcal{X}^2$, and $\overline{w}$ is a vector of handles,* i.e. *variables in $\mathcal{W}$.*

*Given a constraint system $\mathcal{C} = (\Phi; \mathcal{S})$, a set $\mathcal{S}_D$ of dependency constraints, and $\theta \in \mathsf{Sol}(\mathcal{C})$. We write $\theta \models_{(\Phi;\mathcal{S})} \mathcal{S}_D$ when $\theta$ also satisfies the dependency constraints in $\mathcal{S}_D$,* i.e. *when:*

*for each $\overline{X} \ltimes \overline{w} \in \mathcal{S}_D$ there is some $X_i \in \overline{X}$ such that for any recipe $M \in \mathcal{T}(\Sigma_{\mathsf{pub}}, D_{\mathcal{C}}(X_i))$ satisfying $M(\Phi\lambda_\theta \Downarrow) \Downarrow =_{\mathsf{E}} (X_i\theta)(\Phi\lambda_\theta \Downarrow) \Downarrow$ and $\mathsf{valid}(M(\Phi\lambda_\theta \Downarrow))$, we have $\mathrm{vars}^1(M) \cap \overline{w} \neq \emptyset$*

*where $\lambda_\theta$ is the substitution associated to $\theta$ w.r.t. $(\Phi; \mathcal{S})$.*

Intuitively, a dependency constraint $\overline{X} \ltimes \overline{w}$ is satisfied as soon as at least one message among those in $(\overline{X}\theta)(\Phi\lambda_\theta \Downarrow)$ can only be deduced by using at least a message stored in $\overline{w}$ even when one allows to modify recipes (as long as it is still a solution yielding same messages). Note that the representative $\Phi'$ such that $\Phi\lambda_\theta \Downarrow \Phi'$ is unimportant since the validity predicate is stable by $=_{\mathsf{E}}$.

**Example 40.** *Continuing Example 39, assume that $n_1 = n_2 = n$ and let $\theta = \{X_1 \mapsto w_2; X_2 \mapsto w_0\}$. We have that $\theta \in \mathsf{Sol}(\mathcal{C}_2)$ and the substitution associated to $\theta$ w.r.t. $\mathcal{C}_2$ is $\lambda^2_\theta = \{x_1 \mapsto n; x_2 \mapsto n\}$. However, $\theta$ does not satisfy the dependency constraint $X_1 \ltimes w_2$. Indeed, we have that $w_0(\Phi\lambda^2_\theta) = (X_1\theta)(\Phi\lambda^2_\theta) = n$ and $\mathsf{valid}(w_0(\Phi\lambda^2_\theta))$ whereas $\{w_0\} \cap \{w_2\} = \emptyset$. Intuitively, this means that there is no good reason to postpone the execution of the block on channel $c_1$ if the output on $c_2$ is not useful to build the message used in input on $c_1$.*

We shall now define formally how dependency constraints will be added to our constraint systems. Below, we define a function $\mathrm{Deps}(\bullet)$ that takes a compressed symbolic trace as input and returns necessary dependency constraints making sure that all concrete instantiations satisfy the authorised predicates for any prefix of the trace. To simplify the presentation, for some $\overline{X} = (X_1, \ldots, X_\ell)$ and $\overline{w} = (w_1, \ldots, w_k)$, we use the notation $\mathtt{io}_c(\overline{X}, \overline{w})$ as a shortcut for $\mathtt{foc}(\mathtt{in}(c, X_1)).\mathtt{in}(c, X_2) \ldots \mathtt{in}(c, X_\ell).\mathtt{rel}.\mathtt{out}(c, w_1) \ldots \mathtt{out}(c, w_k)$ when $\overline{w} \neq \emptyset$ and for $\mathtt{foc}(\mathtt{in}(c, X_1)).\mathtt{in}(c, X_2) \ldots \mathtt{in}(c, X_\ell).\mathtt{rel}.0$ otherwise.

**Definition 38** (Generation of dependency constraints)**.** *Let $c$ be a channel, and $\mathsf{tr} = \mathtt{io}_{c_1}(\overline{X_1}, \overline{w_1}).\ldots.\mathtt{io}_{c_n}(\overline{X_n}, \overline{w_n})$ be a symbolic, compressed trace. If there exists a rank $k \leq n$ such that $c \prec c_k$ and $c_i \prec c$ for all $k < i \leq n$, then $\mathrm{dep}(\mathsf{tr}, c) = \{\ w \mid w \in \overline{w_i}$ with $k \leq i \leq n\}$. Otherwise, we have that $\mathrm{dep}(\mathsf{tr}, c) = \emptyset$.*

*Then, given a symbolic, compressed trace $\mathsf{tr}$, we define $\mathrm{Deps}(\mathsf{tr})$ by $\mathrm{Deps}(\epsilon) = \emptyset$ and*

$$\mathrm{Deps}(\mathsf{tr}.\mathtt{io}_c(\overline{X}, \overline{w})) = \begin{cases} \mathrm{Deps}(\mathsf{tr}) \cup \{\overline{X} \ltimes \mathrm{dep}(\mathsf{tr}, c)\} & \text{if } \mathrm{dep}(\mathsf{tr}, c) \neq \emptyset \\ \mathrm{Deps}(\mathsf{tr}) & \text{otherwise} \end{cases}$$

Intuitively, $\mathrm{Deps}(\mathsf{tr})$ corresponds to the accumulation of the dependency constraints generated for all prefixes of $\mathsf{tr}$.

**Example 41.** *We consider the symbolic version of Example 31 (in Subsection 5.4.3 with its associated Figure 5.5). We let $\prec$ be such that $c_1 \prec c_2 \prec c_3$.*

*For instance, after executing the symbolic, compressed trace $\mathsf{io}_1.\mathsf{io}_2.\mathsf{io}_3$, a dependency constraint of the form $X_2 \ltimes w_3$ (represented by the left-most arrow in Figure 5.5) is generated. Further, on the symbolic, compressed trace $\mathsf{io}_3.\mathsf{io}_1.\mathsf{io}_2$ we add $X_1 \ltimes w_3$ after the second transition, and $X_2 \ltimes \{w_3, w_1\}$ (represented by the dashed 2-arrow in Figure 5.5) after the third transition.*

#### Symbolic Reduced Semantics

Dependency constraints give rise to a new notion of trace equivalence based on a symbolic, reduced semantics, which is the symbolic counterpart of the reduced semantics ($\xrightarrow{\bullet}_r$) where $\mathrm{Deps}(\bullet)$ plays the role of $(\bullet, \bullet) \blacktriangleright \bullet$. Intuitively, the symbolic, reduced semantics explores all symbolic executions but is equipped with the stronger notion of solution taking $\bullet \models_{(\bullet;\bullet)} \mathrm{Deps}(\bullet)$ into account. We prefer directly defining the reduced trace equivalence below.

**Definition 39** (reduced trace equivalence)**.** *Let $A = (\mathcal{P}; \Phi)$ and $B = (\mathcal{Q}; \Psi)$ be two extended simple processes. We have that $A \sqsubseteq_r^s B$ when, for every sequence $\mathsf{tr}$ such that $(\mathcal{P}; \Phi; \emptyset) \xmapsto{\mathsf{tr}}_c (\mathcal{P}'; \Phi'; \mathcal{S}_A)$, for every $\theta \in \mathsf{Sol}(\Phi'; \mathcal{S}_A)$ such that $\theta \models_{(\Phi';\mathcal{S}_A)} \mathrm{Deps}(\mathsf{tr})$, we have that:*

- *$(\mathcal{Q}; \Psi; \emptyset) \xmapsto{\mathsf{tr}}_c (\mathcal{Q}'; \Psi'; \mathcal{S}_B)$ with $\theta \in \mathsf{Sol}(\Psi'; \mathcal{S}_B)$, and $\theta \models_{(\Psi';\mathcal{S}_B)} \mathrm{Deps}(\mathsf{tr})$;*
- *$\Phi'\lambda_\theta^A \Downarrow \sim \Psi'\lambda_\theta^B \Downarrow$ where $\lambda_\theta^A$ (resp. $\lambda_\theta^B$) is the substitution associated to $\theta$ w.r.t. $(\Phi'; \mathcal{S}_A)$ (resp. $(\Psi'; \mathcal{S}_B)$).*

*We have that $A$ and $B$ are in* reduced trace equivalence*, denoted $A \approx_r^s B$, if $A \sqsubseteq_r^s B$ and $B \sqsubseteq_r^s A$.*

#### Soundness and Completeness

We first prove that the constraints generated by the $\mathrm{Deps}(\bullet)$ operator plays exactly the same role as the authorised predicate of the reduced semantics $(\bullet, \bullet) \blacktriangleright \bullet$.

**Proposition 19.** *Let $A$ be an initial, symbolic configuration, $A \xmapsto{\mathsf{tr}}_c (\mathcal{P}; F; \Phi; \mathcal{S})$ be a symbolic, compressed execution, and, $\theta \in \mathsf{Sol}(\Phi, \mathcal{S})$ a solution of the resulting symbolic configuration. It*

*holds that* $\theta \models_{(\Phi;\mathcal{S})} \mathrm{Deps}(\mathsf{tr})$ *if, and only if,* $(\mathsf{tr}_0\theta, \Psi) \blacktriangleright b\theta$ *for any prefix* $\mathsf{tr}_0.b$ *of* $\mathsf{tr}$ *and some frame* $\Psi$ *such that* $\Phi\lambda \Downarrow \Psi$.

*Proof.* We proceed by induction on the number of blocks in $\mathsf{tr}$. If $\mathsf{tr} = \epsilon$ then $\mathcal{S}' = \emptyset$, $\mathrm{Deps}(\mathsf{tr}) = \emptyset$ and there is no prefix of $\mathsf{tr}$ of the form $\mathsf{tr}_0.b$. The equivalence is thus trivial. Otherwise, $\mathsf{tr} = \mathsf{tr}_0.b$. We let $c$ be the channel of the block $b$ and $\overline{X}$ its second-order variables. We distinguish two cases whether $\mathrm{dep}(\mathsf{tr}_0, c)$ is empty or not. If it is empty, then either $\mathsf{tr}_0 = \epsilon$ or the last blocks of $\mathsf{tr}_0$ on channels different from $c$ are lower (w.r.t. $\prec$) than $b$. In both cases, we trivially have $(\mathsf{tr}_0\theta, \Psi) \blacktriangleright b\theta$ for any $\Psi$ such that $\Phi\lambda \Downarrow \Psi$ and conclude by inductive hypothesis. Otherwise, $\mathrm{dep}(\mathsf{tr}_0, c) = \mathcal{W}_0$. By inductive hypothesis, it suffices to show the following:

(i) $\theta \models_{(\Phi;\mathcal{S})} \overline{X} \ltimes \mathcal{W}_0$ if, and only if, (ii) $(\mathsf{tr}_0\theta, \Psi) \blacktriangleright b\theta$ for some frame $\Psi$ such that $\Phi\lambda \Downarrow \Psi$.

Firstly, we remark that the choice of $\Psi$ such that $\Phi\lambda \Downarrow \Psi$ is irrelevant since $(\bullet, \bullet) \blacktriangleright \bullet$ is stable by $=_{\mathsf{E}}$ on the second argument. We thus fix such a $\Psi$. We note $b_i$ (for $i$ from 0 to $n$) the successive blocks of $\mathsf{tr}_0$ and note $c_i$ their respective channels. Since $\mathrm{dep}(\mathsf{tr}_0, c) \neq \emptyset$, there exists a rank $k < n$ such that $c \prec b_k$ and $c_i \prec c$ for all $k < i \leq n$. We now prove the two implications separately.

$((i) \Rightarrow (ii))$ For the sake of the contradiction, we assume that $(\mathsf{tr}_0\theta, \Psi) \blacktriangleright b\theta$ does not hold. Hence a block $b'$ such that $(b' =_{\mathsf{E}} b\theta)\Psi$ and $\mathsf{tr}_0\theta \triangleright b'$ does not hold. Since $b_i \prec b'$ for all $k < i$ and $b' \prec c_k$, the block $b'$ must be independent of all blocks $b_k, \ldots, b_n$. We now prove that it contradicts $\theta \models_{(\Phi;\mathcal{S})} \overline{X} \ltimes \mathcal{W}_0$. The latter implies the existence of some $X_i \in \overline{X}$ such that one has for any recipe $M \in \mathcal{T}(\Sigma_{\mathsf{pub}}, D_{\mathcal{C}}(X_i))$ such that $M(\Phi\lambda_\theta \Downarrow) \Downarrow =_{\mathsf{E}} (X_i\theta)(\Phi\lambda_\theta \Downarrow) \Downarrow$ and $\mathsf{valid}(M(\Phi\lambda_\theta \Downarrow))$, it holds that $\mathrm{vars}^1(M) \cap \mathcal{W}_0 \neq \emptyset$. However, since $(b' =_{\mathsf{E}} b\theta)\Psi$, one has a recipe $M$ (counterpart of $X\theta$ in $b'$) such that $M(\Phi\lambda_\theta \Downarrow) \Downarrow =_{\mathsf{E}} (X_i\theta)(\Phi\lambda_\theta \Downarrow) \Downarrow$ (remind that $\Phi\lambda_\theta \Downarrow \Psi$). But since $b'$ and $b_i$ are independent, $\mathrm{vars}^1(M) \cap \mathcal{W}_0 = \emptyset$ which is absurd.

$((ii) \Rightarrow (i))$ By $(i)$ and $c_i \prec c \prec c_k$ for all $i$ such that $k < i$, we have that any block $b'$ such that $(b' =_{\mathsf{E}} b)\Psi$ must be recipe-dependent with at least a block $b_j$ for some $k \leq j \leq n$. For the sake of the contradiction, we assume that $\theta \not\models_{(\Phi;\mathcal{S})} \overline{X} \ltimes \mathcal{W}_0$. Hence, for any $X_i \in \overline{X}$, there exists a recipe $M_i$ computing the same message as $X_i\theta$ for $\Phi\lambda \Downarrow$ such that $\mathrm{vars}^1(M_i) \cap \mathcal{W}_0 = \emptyset$. By replacing $X_i$ by $M_i$ in the block $b\theta$ we thus obtain a block $b'$ such that $(b' =_{\mathsf{E}} b\theta)\Psi$ and $\mathrm{vars}^1(b') \cap \mathcal{W}_0 = \emptyset$ which contradicts the fact that $b'$ must be recipe-dependent with at least a block $b_j$ for some $k \leq j \leq n$. □

To prove that the symbolic, reduced trace equivalence coincides with the reduced trace equivalence, we now follow the same proof technique as the one developed for the compressed symbolic trace equivalence (w.r.t. compressed trace equivalence) in Subsection 6.2.2 and for the symbolic trace equivalence (w.r.t. trace equivalence) in Subsection 6.2.1. Both directions of this mapping (propositions 20 and 21) directly follow from the already established mapping between symbolic, compressed executions and compressed executions (see propositions 17 and 18) in addition to Proposition 19 stating that $\mathrm{Deps}(\bullet)$ exactly mimics $(\bullet, \bullet) \blacktriangleright \bullet$.

**Proposition 20.** *Let* $(\mathcal{P}; \Phi)$ *be an initial simple configuration such that* $\lceil(\mathcal{P}; \Phi; \emptyset)\rceil \xrightarrow{\mathsf{tr}}_r (\mathcal{P}_s; F_A; \Phi_s; \mathcal{S}_s)$, *and* $\theta \in \mathsf{Sol}(\Phi_s; \mathcal{S}_s)$. *We have that* $\lceil(\mathcal{P}; \Phi)\rceil \xrightarrow{\mathsf{tr}\theta}_r (\mathrm{NF}_{\mathscr{R}}(\mathcal{P}_s\lambda); F'; \Phi')$ *where*

$F' = \mathrm{NF}_{\mathscr{R}}(F\lambda)$ *if* $F$ *is a process and* $F' = \varnothing$ *otherwise,* $\lambda$ *is the first-order solution of* $(\Phi_s; \mathcal{S}_s)$ *associated to* $\theta$*, and,* $\Phi'$ *is such that* $\Phi_s\lambda \Downarrow \Phi'$*.*

**Proposition 21.** *Let* $(\mathcal{P}; \Phi)$ *be an initial, simple configuration such that* $\lceil(\mathcal{P}; \Phi)\rceil \xrightarrow{\mathsf{tr}}_r (\mathcal{P}'; F'; \Phi')$*. There exists a symbolic configuration* $(\mathcal{P}_s; F_s; \Phi_s; \mathcal{S})$*, a solution* $\theta \in \mathsf{Sol}(\Phi_s; \mathcal{S})$*, and a sequence* $\mathsf{tr}_s$ *such that:*

- $\lceil(\mathcal{P}; \Phi; \emptyset)\rceil \overset{\mathsf{tr}_s}{\longmapsto}_r (\mathcal{P}_s; F_s; \Phi_s; \mathcal{S})$*;*
- $\mathcal{P}' = \mathcal{P}_s\lambda$*,* $F' = F_s\lambda$*,* $\Phi_s\lambda \Downarrow \Phi'$*; and*
- $\mathsf{tr} = \mathsf{tr}_s\theta$

*where* $\lambda$ *is the first-order solution of* $(\Phi_s; \mathcal{S})$ *associated to* $\theta$*.*

As already argued in Subsection 6.2.2, the two propositions above (propositions 20 and 21) are the two only ingredients necessary to obtain the theorem below (exactly the same proof of Proposition 16 applies).

**Theorem 5.** *For any initial, simple configurations* $A$ *and* $B$*, we have that:*

$$\lceil A\rceil \sqsubseteq_r \lceil B\rceil \iff \lceil A\rceil \sqsubseteq_r^s \lceil B\rceil.$$

*Proof.* Same proof as the one of Proposition 16. □

As an immediate consequence of theorems 3 and 5, we obtain that the relations $\approx$ and $\approx_r^s$ coincide.

**Corollary 5.** *For any initial, simple processes* $A$ *and* $B$ *such that* $\mathsf{skl}(A) = \mathsf{skl}(B)$*, we have that:*

$$A \approx B \iff \lceil A\rceil \approx_r^s \lceil B\rceil.$$

## 6.3 Integration in Apte

We validate our approach by integrating our refined semantics in the Apte tool. As already mentioned, Apte is a tool deciding trace equivalence for a fixed set of cryptographic primitives (*i.e.* "standard primitives") and bounded processes (no replication, no recursion).

As we shall see, the compressed semantics can easily be used as a replacement for the usual semantics in verification algorithms. However, exploiting the reduced semantics is not trivial, and requires to adapt the constraint resolution procedure. Fortunately, this can be done in a relatively superficial way, which avoids having to enter into the complex details of Apte's algorithm.

It is beyond the scope of this thesis to provide a detailed summary of how the verification tool Apte actually works. A paper of 50 pages long describing solely the constraint resolution procedure implemented in Apte (the part of the procedure corresponding to $\mapsto^{\mathsf{A2}}$ in our presentation) is now available [CCLD16]. Detailed proofs of the soundness, completeness and termination of this algorithm are available in a long and technical appendix (more than 100 pages).

In order to show how our reduced semantics have been integrated in the constraint solving procedure of Apte, we choose to provide a high-level axiomatic presentation of the Apte's algorithm. This allows us to prove that our integration is correct in a relatively superficial way, which avoids having to enter into the complex details of Apte's algorithm (which manipulates matrices of constraint systems and additional kinds of constraints). These statements are consequences of results stated and proved in [Che12] and have been written in concertation with Vincent Cheval (developer of Apte). However, due to some changes in the presentation, proving them will require to adapt most of the proofs. It is therefore beyond the scope of this thesis to provide formal proofs of these axioms.

In Subsection 6.3.1, we start with a high-level presentation of Apte's algorithm, following the original procedure [CCD11] but assuming public channels only. The purpose of this presentation is to provide the reader enough details about Apte to explain him how our optimisations have been integrated, leaving out unimportant details. We axiomatize the procedure in Subsection 6.3.2 and prove it correct w.r.t. trace equivalence in Subsection 6.3.3. Then we explain how compressed semantics can be used to enhance the procedure (Section 6.3.4). We finally describe how our reduction technique can be integrated (Section 6.3.5).

### 6.3.1 Apte in a Nutshell

Apte has been designed for a fixed equational theory $\mathsf{E}_{\mathsf{apte}}$ (formally defined in Example 12 from Chapter 2) containing standard cryptographic primitives whose the signatutre is recalled next (where $\Sigma_0$ is a user-defined set of constants not involved in any equation):

$$\begin{array}{rcl} \Sigma_c & = & \Sigma_0 \cup \{\mathsf{aenc}, \mathsf{pk}, \mathsf{enc}, \mathsf{hash}, \mathsf{sign}, \mathsf{vk}, \langle\rangle\} \\ \Sigma_d & = & \{\mathsf{adec}, \mathsf{dec}, \mathsf{check}, \pi_1, \pi_2\}. \end{array}$$

We now give a high-level description of the algorithm that is implemented in Apte. The main idea is to perform all possible symbolic executions of the processes, keeping together the processes that can be reached using the same sequence of symbolic actions. Then, at each step of this symbolic execution, the procedure checks that for every solution of every process on one side, there is a corresponding solution for some process on the other side so that the resulting frames are in static equivalence. This check for *symbolic equivalence* is not obviously decidable. To achieve it, Apte's procedure relies on a set of rules for simplifying sets of constraint systems. These rules are used to put constraint systems in a *solved form* that enables the efficient verification of symbolic equivalence.

The symbolic execution used in Apte is essentially the same as $\overset{\bullet}{\mapsto}$ described in Section 6.2. However, Apte's constraint resolution procedure introduces new kinds of constraints. Fortunately, we do not need to enter into the details of those constraints and how they are manipulated. Instead, we treat them axiomatically.

**Definition 40** (extended constraint system/symbolic process)**.** *An* extended constraint system $\mathcal{C}^+ = (\Phi; \mathcal{S}; \mathcal{S}^+)$ *consists of a constraint system* $\mathcal{C} = (\Phi; \mathcal{S})$ *together with an additional set* $\mathcal{S}^+$ *of* extended constraints*. We treat this latter set abstractly, only assuming an associated*

*satisfaction relation, written* $\theta \models \mathcal{S}^+$*, such that* $\theta \models \emptyset$ *always holds, and* $\theta \models \mathcal{S}_1^+$ *implies* $\theta \models \mathcal{S}_2^+$ *when* $\mathcal{S}_2^+ \subseteq \mathcal{S}_1^+$*. We define the set of solutions of* $\mathcal{C}^+$ *as* $\mathsf{Sol}^+(\mathcal{C}^+) = \{\ \theta \in \mathsf{Sol}(\mathcal{C}) \mid \theta \models \mathcal{S}^+\ \}$.

*An* extended symbolic process $(\mathcal{P}; \Phi; \mathcal{S}; \mathcal{S}^+)$ *is a symbolic process with an additional set of extended constraints* $\mathcal{S}^+$*.*

We shall denote extended constraint systems by $\mathcal{S}^+$, $\mathcal{S}_1^+$, etc. Extended symbolic processes will be denoted by $A^+$, $B^+$, etc. Sets of extended symbolic processes will simply be denoted by $\mathbf{A}$, $\mathbf{B}$, etc. For convenience, we extend $\mathsf{Sol}$ and $\mathsf{Sol}^+$ to symbolic processes and extended symbolic processes in the natural way:

$$\mathsf{Sol}(\mathcal{P}; \Phi; \mathcal{S}) = \mathsf{Sol}(\Phi; \mathcal{S}) \quad \text{and} \quad \mathsf{Sol}^+(\mathcal{P}; \Phi; \mathcal{S}; \mathcal{S}^+) = \mathsf{Sol}^+(\Phi; \mathcal{S}; \mathcal{S}^+).$$

We may also use the following notation to translate back and forth between symbolic processes and extended symbolic processes:

$$⟦(\mathcal{P}; \Phi; \mathcal{S})⟧ = (\mathcal{P}; \Phi; \mathcal{S}; \emptyset) \quad \text{and} \quad ⟦(\mathcal{P}; \Phi; \mathcal{S}; \mathcal{S}^+)⟧ = (\mathcal{P}; \Phi; \mathcal{S}).$$

We can now introduce the key notion of symbolic equivalence between sets of extended symbolic processes, or more precisely between their underlying extended constraint systems.

**Definition 41** (symbolic equivalence)**.** *Given two sets of extended symbolic processes* $\mathbf{A}$ *and* $\mathbf{B}$*, we have that* $\mathbf{A} \prec^+ \mathbf{B}$ *if for every* $A^+ = (\mathcal{P}_A; \Phi_A; \mathcal{S}_A; \mathcal{S}_A^+) \in \mathbf{A}$*, for every* $\theta \in \mathsf{Sol}^+(A^+)$*, there exists* $B^+ = (\mathcal{P}_B; \Phi_B; \mathcal{S}_B; \mathcal{S}_B^+) \in \mathbf{B}$ *such that* $\theta \in \mathsf{Sol}^+(B^+)$ *and* $\Phi_A \lambda_\theta^A \sim \Phi_B \lambda_\theta^B$ *where* $\lambda_\theta^A$ *(resp.* $\lambda_\theta^B$*) is the substitution associated to* $\theta$ *w.r.t.* $(\Phi_A; \mathcal{S}_A)$ *(resp.* $(\Phi_B; \mathcal{S}_B)$*). We say that* $\mathbf{A}$ *and* $\mathbf{B}$ *are in* symbolic equivalence*, denoted by* $\mathbf{A} \sim^+ \mathbf{B}$*, if* $\mathbf{A} \prec^+ \mathbf{B}$ *and* $\mathbf{B} \prec^+ \mathbf{A}$*.*

The whole trace equivalence procedure can finally be abstractly described by means of a transition system $\mapsto^{\mathsf{A}}$ on pairs of sets of extended symbolic processes, labelled by observable symbolic actions. Informally, the intent is that a pair of processes is in trace equivalence iff only symbolically equivalent pairs may be reached from the initial pair using $\mapsto^{\mathsf{A}}$.

We now define $\mapsto^{\mathsf{A}}$ formally. A transition $(\mathbf{A}; \mathbf{B}) \mapsto^{\mathsf{A}}$ can take place iff $\mathbf{A}$ and $\mathbf{B}$ are in symbolic equivalence[1].

Each transition for some observable action $\alpha$ consists of two steps, *i.e.* $(\mathbf{A}; \mathbf{B}) \overset{\alpha}{\mapsto}^{\mathsf{A}} (\mathbf{A''}; \mathbf{B''})$ iff $(\mathbf{A}; \mathbf{B}) \overset{\alpha}{\mapsto}^{\mathsf{A1}} (\mathbf{A'}; \mathbf{B'})$ and $(\mathbf{A'}; \mathbf{B'}) \mapsto^{\mathsf{A2}} (\mathbf{A''}; \mathbf{B''})$, where the latter transitions are described below:

1. The first part of the transition consists in performing an observable symbolic action $\alpha$ (either $\mathtt{in}(c, X)$ or $\mathtt{out}(c, w)$) followed by all available unobservable ($\tau$) actions. This is done for each extended symbolic process that occurs in the pair of sets, and each possible transition of one such process generates a new element in the target set. Formally, we have $(\mathbf{A}; \mathbf{B}) \overset{\alpha}{\mapsto}^{\mathsf{A1}} (\mathbf{A'}; \mathbf{B'})$ if

$$\mathbf{A'} = \bigcup_{(\mathcal{P}; \Phi; \mathcal{S}; \mathcal{S}^+) \in \mathbf{A}} \left\{\ (\mathcal{P}'; \Phi'; \mathcal{S}'; \mathcal{S}^+) \mid (\mathcal{P}; \Phi; \mathcal{S}) \overset{\alpha.\tau^*}{\Longrightarrow} (\mathcal{P}'; \Phi'; \mathcal{S}') \not\overset{\tau}{\Longrightarrow} \right\},$$

[1] This definition yields infinite executions for $\mapsto^{\mathsf{A}}$ if no inequivalent pair is met. Each such execution eventually reaches $(\emptyset; \emptyset)$ while, in practice, executions are obviously not explored past empty pairs. We chose to introduce this minor gap to make the theory more uniform.

and correspondingly for $\mathbf{B}'$. Note that elements of $(\mathbf{A};\mathbf{B})$ that cannot perform $\alpha$ are simply discarded, and that the constraint systems of individual processes are enriched according to their own transitions whereas the extended part of constraint systems are left unchanged. For a fixed symbolic action $\alpha$, the $\overset{\alpha}{\mapsto}^{\mathsf{A1}}$ transition is deterministic. The choice of names for handles and second-order variables does not matter, and therefore the relation $\mapsto^{\mathsf{A1}}$ is also finitely branching.

2. The second part consists in simplifying the constraint systems of $(\mathbf{A}';\mathbf{B}')$ until reaching *solved forms*. This part of the transition is non-deterministic, *i.e.* several different $(\mathbf{A}'';\mathbf{B}'')$ may be reached depending on various choices, *e.g.* whether a message is derived by using a function symbol or one of the available handles. Although branching, this part of the transition is finitely branching. Moreover, only extended constraints may change: for any $(\mathcal{P};\Phi;\mathcal{S};\mathcal{S}_1^+) \in \mathbf{A}''$ there must be a $\mathcal{S}_0^+$ such that $(\mathcal{P};\Phi;\mathcal{S};\mathcal{S}_0^+) \in \mathbf{A}'$, and similarly for $\mathbf{B}''$.

An important invariant of this construction is that all the processes occurring in any of the two sets of processes have constraint systems that share a common structure. More precisely the transitions maintain that for any $(\mathcal{P}_1;\Phi_1;\mathcal{S}_1;\mathcal{S}_1^+), (\mathcal{P}_2;\Phi_2;\mathcal{S}_2;\mathcal{S}_2^+) \in \mathbf{A}\cup\mathbf{B}$, $\mathrm{vars}^2(\mathcal{S}_1) = \mathrm{vars}^2(\mathcal{S}_2)$ and $D \vdash^?_X x$ occurs in $\mathcal{S}_1$ iff it occurs in $\mathcal{S}_2$.

**Example 42.** *Consider the simple basic processes* $R_i = \mathtt{in}(c_i, x_i).\mathsf{if}\ x_i = \mathsf{ok}\ \mathsf{then}\ \mathtt{out}(c_i, n_i)$ *for* $i \in \mathbb{N}, x_i \in \mathcal{X}, n_i \in \mathcal{N}$, $\mathsf{ok}$ *a public constant. We illustrate the roles of* $\mapsto^{\mathsf{A1}}$ *and* $\mapsto^{\mathsf{A2}}$ *on the pair* $(\{Q_0\};\{Q_0\})$ *where* $Q_0 = (\{R_1, R_2\};\emptyset;\emptyset;\emptyset)$. *We have that*

$$(\{Q_0\};\{Q_0\}) \overset{\mathtt{in}(c_2,X_2)}{\mapsto}{}^{\mathsf{A1}} (\{Q_0^t, Q_0^e\};\{Q_0^t, Q_0^e\})$$

*where* $Q_0^t$ *and* $Q_0^e$ *are the two symbolic processes one may obtain by executing the observable action* $\mathtt{in}(c_2, X_2)$*, depending on the conditional after that input. Specifically, we have:*

- $Q_0^t = (\{R_1, \mathtt{out}(c_2, n_2)\};\emptyset;\{\emptyset \vdash^?_{X_2} x_2, x_2 =^? \mathsf{ok}\};\emptyset)$
- $Q_0^e = (\{R_1\};\emptyset;\{\emptyset \vdash^?_{X_2} x_2, x_2 \neq^? \mathsf{ok}\};\emptyset)$

*After this first step,* $\mapsto^{\mathsf{A2}}$ *is going to non-deterministically solve the constraint systems. From the latter pair, it will produce only two alternatives. Indeed, if* $x_2 =^? \mathsf{ok}$ *holds then* Apte *infers that the only recipe that it needs to consider is the recipe* $R = \mathsf{ok}$*. In that case, the only considered solution is* $\{X_2 \mapsto \mathsf{ok}\}$*. Otherwise,* $x_2 \neq^? \mathsf{ok}$ *holds but, at this point, no more information is inferred on* $X_2$*. Formally,*

$$(\{Q_0^t, Q_0^e\};\{Q_0^t, Q_0^e\}) \mapsto^{\mathsf{A2}} (\{Q_1^t\};\{Q_1^t\})$$
$$(\{Q_0^t, Q_0^e\};\{Q_0^t, Q_0^e\}) \mapsto^{\mathsf{A2}} (\{Q_1^e\};\{Q_1^e\}) \quad \textit{where}$$

- $Q_1^t = (\{R_1, \mathtt{out}(c_2, n_2)\};\emptyset;\{\emptyset \vdash^?_{X_2} x_2, x_2 =^? \mathsf{ok}\};\mathcal{S}_1^t)$ *and* $\mathsf{Sol}^+(Q_1^t) = \{\theta_1^t\}$ *where* $\theta_1^t = \{X_2 \mapsto \mathsf{ok}\}$*;*
- $Q_1^e = (\{R_1\};\emptyset;\{\emptyset \vdash^?_{X_2} x_2, x_2 \neq^? \mathsf{ok}\};\mathcal{S}_1^e)$.

*The content of $\mathcal{S}_1^t$ and $\mathcal{S}_1^e$ is not important. Note that after $\mapsto^{\mathsf{A2}}$, only one alternative remains in this example (*i.e. *there is only one extended symbolic process on each side of the resulting pair) because only one of the two processes $Q_0^t, Q_0^e$ complies with the choices made in each branch.*

**Definition 42** ($\approx^{\mathsf{A}}$)**.** *Let $A = (\mathcal{P}_A; \Phi_A)$ and $B = (\mathcal{P}_B; \Phi_B)$ be two processes. We say that $A \approx^{\mathsf{A}} B$ when $\mathbf{A} \sim^{+} \mathbf{B}$ for any pair $(\mathbf{A}; \mathbf{B})$ such that $((\mathcal{P}_A; \Phi_A; \emptyset; \emptyset); (\mathcal{P}_B; \Phi_B; \emptyset; \emptyset)) \overset{\mathsf{tr}}{\mapsto}{}^{\mathsf{A}} (\mathbf{A}; \mathbf{B})$.*

As announced above, we expect $\approx^{\mathsf{A}}$ to coincide with trace equivalence. We shall actually prove it (see Section 6.3.3), after having introduced a few axioms (Section 6.3.2). We note, however, that this can only hold under some minor assumptions on processes. In practice, Apte does not need those assumptions but they allow for a more concise presentation.

**Definition 43.** *A simple process (resp. symbolic process) $A$ is said to be* quiescent *when $A \overset{\tau}{\not\rightarrow}$ (resp. $A \overset{\tau}{\not\Longrightarrow}$). An extended symbolic process $A^+$ is* quiescent *when $[\![A^+]\!] \overset{\tau}{\not\Longrightarrow}$.*

In $\overset{\alpha}{\mapsto}{}^{\mathsf{A1}}$ transitions, processes must start by executing an observable action $\alpha$ and possibly some $\tau$ actions after that. Hence, it does not make sense to consider $\mapsto^{\mathsf{A}}$ transitions on processes that can still perform $\tau$ actions. We shall thus establish that $\approx^{\mathsf{A}}$ and $\approx^{s}$ coincide only on quiescent processes, which is not a significant restriction since it is always possible to pre-execute all available $\tau$-actions before testing equivalences.

### 6.3.2 Specification of the Procedure

We now list and comment the specification satisfied by the exploration performed by Apte. These statements are consequences of results stated and proved in [Che12] and it is beyond the scope of this thesis to prove them.

**Soundness and completeness of constraint resolution.** The $\mapsto^{\mathsf{A2}}$ step, corresponding to Apte's constraint resolution procedure, only makes sense under some assumptions on the (common) structure of the processes that are part of the pairs of sets under consideration. Rather than precisely formulating these conditions (which would be at odds with the abstract treatment of extended constraint systems) we start by defining an over-approximation of the set of pairs on which we may apply $\mapsto^{\mathsf{A2}}$ at some point. We choose this over-approximation sufficiently large to cover pairs produced by the compressed semantics, and we then formulate our specifications in that domain. This over-approximation have to cover two things:

1. we have to consider additional disequalities of the form $t \neq^{?} t$ in constraint systems since they are eventually added by our compressed symbolic semantics (see Figure 6.4);

2. we have to allow the removal of some extended symbolic process from the original sets since they are eventually discarded by our compressed (resp. reduced) symbolic semantics.

Given an extended symbolic process $A^+ = (\mathcal{P}; \Phi; \mathcal{S}; \mathcal{S}^+)$, we denote by $\mathrm{add}(A^+)$ the set of extended symbolic processes obtained from $A^+$ by adding into $\mathcal{S}$ a number of disequalities of

the form $u \neq^? u$ with $\mathrm{vars}^1(u) \subseteq \mathrm{vars}^1(\mathcal{S})$. This is then extended to sets of extended symbolic processes as follows: $\mathrm{add}(\{A_1^+, \ldots, A_n^+\}) = \{\{B_1^+, \ldots, B_n^+\} \mid B_i^+ \in \mathrm{add}(A_i^+)\}$.

**Definition 44** (valid and intermediate valid pairs)**.** *The set of* valid pairs *is the least set such that:*

- *For all quiescent, symbolic processes* $A = (\mathcal{P}; \Phi; \emptyset)$ *and* $B = (\mathcal{Q}; \Psi; \emptyset)$, $(\{\lceil A \rceil\}; \{\lceil B \rceil\})$ *is valid.*
- *If* $(\mathbf{A}; \mathbf{B})$ *is valid and* $\mathbf{A} \sim^+ \mathbf{B}$, $(\mathbf{A}; \mathbf{B}) \overset{\alpha_\wedge}{\mapsto}^{\mathsf{A1}} (\mathbf{A}_1; \mathbf{B}_1)$, $\mathbf{A}_2 \subseteq \mathbf{A}_1$, $\mathbf{B}_2 \subseteq \mathbf{B}_1$, $\mathbf{A}_3 \in \mathrm{add}(\mathbf{A}_2)$, $\mathbf{B}_3 \in \mathrm{add}(\mathbf{B}_2)$, *and* $(\mathbf{A}_3; \mathbf{B}_3) \mapsto^{\mathsf{A2}} (\mathbf{A}'; \mathbf{B}')$ *then* $(\mathbf{A}'; \mathbf{B}')$ *is valid. In that case, the pair* $(\mathbf{A}_3; \mathbf{B}_3)$ *is called an* intermediate valid pair.

It immediately follows that $(\{\lceil A \rceil\}; \{\lceil B \rceil\}) \overset{\mathrm{tr}}{\mapsto}^{\mathsf{A}} (\mathbf{A}; \mathbf{B})$ implies that $(\mathbf{A}; \mathbf{B})$ is valid and only made of quiescent, extended symbolic processes. But the notion of validity accommodates more pairs: it will cover pairs accessible under refinements of $\mapsto^{\mathsf{A}}$ based on subset restrictions of $\mapsto^{\mathsf{A1}}$. We may note that these pairs are actually pairs that would have been explored by Apte when starting with another pair of processes (*e.g.* a process that makes explicit the use of trivial conditionals of the form $\mathsf{let}\ u = t\ \mathtt{in}\ P\ \mathsf{else}\ Q$ ). Therefore, those pairs do not cause any trouble when they have to be handled by Apte.

**Axiom 1** (soundness of constraint resolution)**.** *Let* $(\mathbf{A}'; \mathbf{B}')$ *be an intermediate valid pair such that* $(\mathbf{A}'; \mathbf{B}') \mapsto^{\mathsf{A2}} (\mathbf{A}''; \mathbf{B}'')$. *Then, for all* $A'' \in \mathbf{A}''$ *(resp.* $B'' \in \mathbf{B}''$*) there exists some* $A' \in \mathbf{A}'$ *(resp.* $B' \in \mathbf{B}'$*) such that* $\llbracket A' \rrbracket = \llbracket A'' \rrbracket$ *(resp.* $\llbracket B' \rrbracket = \llbracket B'' \rrbracket$*) and* $\mathsf{Sol}^+(A'') \subseteq \mathsf{Sol}^+(A')$ *(resp.* $\mathsf{Sol}^+(B'') \subseteq \mathsf{Sol}^+(B')$*).*

Apte almost treats symmetrically the two components of the pair of sets on which transitions take place. This is reflected by the fact that axioms concern both sides and are completely symmetric as in Axiom 1. In order to make the following specifications more concise and readable, we may state properties only for one of the two sets and consider "symmetrically" as well.

The completeness specification is in two parts: it first states that no first-order solution is lost in the constraint resolution process, and then that the branching of $\mapsto^{\mathsf{A2}}$ corresponds to different second-order solutions.

**Axiom 2** (first-order completeness of constraint resolution)**.** *Let* $(\mathbf{A}; \mathbf{B})$ *be an intermediate valid pair. For all* $A^+ \in \mathbf{A}$ *and* $\theta \in \mathsf{Sol}(A^+)$ *there exists* $(\mathbf{A}; \mathbf{B}) \mapsto^{\mathsf{A2}} (\mathbf{A}_2; \mathbf{B}_2)$, $A_2^+ \in \mathbf{A}_2$ *and* $\theta^+ \in \mathsf{Sol}^+(A_2^+)$ *such that* $\llbracket A_2^+ \rrbracket = \llbracket A^+ \rrbracket$ *and* $\lambda_\theta^A =_{\mathsf{E}} \lambda_{\theta^+}^A$*, where* $\lambda_\theta^A$ *(resp.* $\lambda_{\theta^+}^A$*) is the substitution associated to* $\theta$ *(resp. to* $\theta^+$*) w.r.t.* $\llbracket A^+ \rrbracket$*. Symmetrically for* $B^+ \in \mathbf{B}$*.*

To express that the branching of $\mapsto^{\mathsf{A2}}$ corresponds to different second-order solutions, we state in the next Axiom that if a branch of $\mapsto^{\mathsf{A2}}$ keeps a second-order solution for one process then it keeps it *uniformly* for all processes.

**Axiom 3** (second-order consistency of constraint resolution)**.** *Let* $(\mathbf{A}; \mathbf{B})$ *be an intermediate valid pair such that* $(\mathbf{A}; \mathbf{B}) \mapsto^{\mathsf{A2}} (\mathbf{A}_2; \mathbf{B}_2)$, $\theta \in \mathsf{Sol}^+(A^+)$ *for some* $A^+ \in \mathbf{A}$ *and* $\theta \in \mathsf{Sol}^+(C_2^+)$

*for some* $C_2^+ \in \mathbf{A}_2 \cup \mathbf{B}_2$. *Then there exists some* $A_2^+ \in \mathbf{A}_2$ *such that* $\llbracket A^+ \rrbracket = \llbracket A_2^+ \rrbracket$ *and* $\theta \in \mathsf{Sol}^+(A_2^+)$. *Symmetrically for* $B^+ \in \mathbf{B}$.

**Partial solution.** In order to avoid performing some explorations when dependency constraints of our reduced semantics are not satisfied, we shall be interested in knowing when all solutions of a given constraint system assign a given recipe to some variable. Such information is generally available in the solved forms computed by Apte, but not always in a complete fashion. We reflect this by introducing an abstract function that represents the information that can effectively be inferred by the procedure.

**Definition 45** (partial solution)**.** *We assume a* partial solution[2] *function* $\mathsf{ps}$ *which maps sets of extended constraints* $\mathcal{S}^+$ *to a substitution, such that for any* $\theta \in \mathsf{Sol}^+(\mathcal{P};\Phi;\mathcal{S};\mathcal{S}^+)$, *there exists* $\theta'$ *such that* $\theta = \mathsf{ps}(\mathcal{S}^+) \sqcup \theta'$. *We extend* $\mathsf{ps}$ *to extended symbolic processes:* $\mathsf{ps}(\mathcal{P};\Phi;\mathcal{S};\mathcal{S}^+) = \mathsf{ps}(\mathcal{S}^+)$.

Intuitively, given an extended constraint system, the function $\mathsf{ps}$ returns the value of some of its second-order variables (those for which their instantiation is already completely determined). Our specification of the partial solution shall postulate that the partial solution returned by Apte is the same for each extended symbolic process occurring in a pair $(\mathbf{A};\mathbf{B})$ reached during the exploration. Moreover, there is a monotonicity property that ensures that this partial solution becomes more precise along the exploration.

**Axiom 4.** *We assume the following about the partial solution:*

*(1) For any valid pair* $(\mathbf{A};\mathbf{B})$, *we have that* $\mathsf{ps}(A) = \mathsf{ps}(B)$ *for any* $A, B \in \mathbf{A} \cup \mathbf{B}$. *This allows us to simply write* $\mathsf{ps}(\mathbf{A};\mathbf{B})$ *when* $\mathbf{A} \cup \mathbf{B} \neq \emptyset$.

*(2) For any intermediate valid pair* $(\mathbf{A};\mathbf{B})$ *such that* $(\mathbf{A};\mathbf{B}) \mapsto^{\mathsf{A2}} (\mathbf{A}';\mathbf{B}')$ *and* $\mathbf{A}' \cup \mathbf{B}' \neq \emptyset$, *we have* $\mathsf{ps}(\mathbf{A}';\mathbf{B}') = \mathsf{ps}(\mathbf{A};\mathbf{B}) \sqcup \theta$ *for some* $\theta$.

**Example 43.** *Continuing Example 42, we first note that* $(\{Q_0\};\{Q_0\})$ *is a valid pair. Second, the exploration* $(\{Q_0\};\{Q_0\}) \xmapsto{\mathtt{in}(c_2,X_2),}{}^{\mathsf{A}} (\{Q_1^t\};\{Q_1^t\})$ *covers all executions of the form* $\llbracket Q_0 \rrbracket \xmapsto{\mathtt{in}(c_2,X_2).\tau_{\mathtt{then}},} \llbracket Q_1^t \rrbracket$ *going to the* `then` *branch even though the only solution of* $Q_1^t$ *is* $\theta_1^t$. *Indeed, if* $\theta \in \mathsf{Sol}(\llbracket Q_1^t \rrbracket)$ *then the message computed by* $X_2\theta$ *should be equal to* $\mathsf{ok}$ *and thus no first-order solution is lost as stated by* Axiom 2. *Moreover, because the value of* $X_2$ *is already known in* $Q_1^t$, *we may have* $\mathsf{ps}(Q_1^t) = \mathsf{ps}(\mathcal{S}_1^+) = \{X_2 \mapsto \mathsf{ok}\}$.

### 6.3.3 Proof of the Original Procedure

The procedure, axiomatized as above, can be proved correct w.r.t the regular symbolic semantics $\Rightarrow$ and its induced trace equivalence $\approx^s$ as defined in Section 6.2.1. Of course, Axiom 4 is unused in this first result. It will be used later on when implementing our reduced semantics. We

[2]We use the notation $\sigma_1 \sqcup \sigma_2$ to emphasise the fact that the two substitutions do not interact together. They have disjoint domain, i.e. $\mathrm{dom}(\sigma_1) \cap \mathrm{dom}(\sigma_2) = \emptyset$, and no variable of $\mathrm{dom}(\sigma_i)$ occurs in $\mathrm{img}(\sigma_j)$ with $\{i,j\} = \{1,2\}$.

first start by establishing that all the explorations performed by Apte correspond to symbolic executions.

This result is not new and has been established from scratch (*i.e.* without relying on the axioms stated in the previous section) in [Che12]. Nevertheless, we found it useful to establish that our axioms are sufficient to prove correctness of the original Apte procedure. The proofs provided in the following sections to establish correctness of our optimised procedure follow the same lines as the ones presented below.

**Lemma 16.** *Let* $(\mathbf{A};\mathbf{B})$ *be a valid pair such that* $(\mathbf{A};\mathbf{B}) \overset{\mathsf{tr}}{\mapsto}^{\mathsf{A}} (\mathbf{A}';\mathbf{B}')$*. Then, for all* $A' \in \mathbf{A}'$ *there is some* $A \in \mathbf{A}$ *such that* $\llbracket A \rrbracket \overset{\mathsf{tr}'}{\Longmapsto} \llbracket A' \rrbracket$ *for some* $\mathsf{tr}'$ *with* $\mathsf{obs}(\mathsf{tr}') = \mathsf{tr}$*. Symmetrically for* $B' \in \mathbf{B}'$*.*

*Proof.* We proceed by induction on $\mathsf{tr}$. When $\mathsf{tr}$ is empty, we have that $(\mathbf{A};\mathbf{B}) = (\mathbf{A}';\mathbf{B}')$, and the result trivially holds. Otherwise we have that:

$$(\mathbf{A};\mathbf{B}) \overset{\alpha}{\mapsto}^{\mathsf{A1}} (\mathbf{A}_1;\mathbf{B}_1) \mapsto^{\mathsf{A2}} (\mathbf{A}_2;\mathbf{B}_2) \overset{\mathsf{tr}_0}{\mapsto}^{\mathsf{A}} (\mathbf{A}';\mathbf{B}') \text{ with } \mathsf{tr} = \alpha.\mathsf{tr}_0.$$

Let $A'$ be a process of $\mathbf{A}'$. By induction hypothesis we have some $A_2 \in \mathbf{A}_2$ such that $\llbracket A_2 \rrbracket \overset{\mathsf{tr}'_0}{\Longmapsto} \llbracket A' \rrbracket$ with $\mathsf{obs}(\mathsf{tr}'_0) = \mathsf{tr}_0$. By Axiom 1 there is some $A_1 \in \mathbf{A}_1$ such that $\llbracket A_1 \rrbracket = \llbracket A_2 \rrbracket$, and by definition of $\mapsto^{\mathsf{A1}}$ we finally find some $A \in \mathbf{A}$ such that $\llbracket A \rrbracket \overset{\alpha.\tau^k}{\Longmapsto} \llbracket A_1 \rrbracket$. To sum up, we have $A \in \mathbf{A}$ such that $\llbracket A \rrbracket \overset{\mathsf{tr}}{\Longmapsto} \llbracket A' \rrbracket$ with $\mathsf{obs}(\mathsf{tr}') = \mathsf{tr}$. □

We now turn to completeness results. Assuming that processes under study are in equivalence $\approx^{\mathsf{A}}$ (so that Apte will not stop its exploration prematurely), we are able to show that any valid symbolic execution (*i.e.* a symbolic execution with a solution in its resulting constraint system) is captured by an exploration performed by Apte. Actually, since Apte discards some second-order solution during its exploration, we can only assume that another second-order solution with the same associated first-order solution will be found. We prove the former in the next lemma.

**Lemma 17.** *Let* $A = (\mathcal{P};\Phi;\emptyset)$*,* $B = (\mathcal{Q};\Psi;\emptyset)$ *and* $A' = (\mathcal{P}';\Phi';\mathcal{S}')$ *be three quiescent, symbolic processes such that* $(\mathcal{P};\Phi) \approx^{\mathsf{A}} (\mathcal{Q};\Psi)$*,* $A \overset{\mathsf{tr}}{\Longmapsto} A'$*, and* $\theta \in \mathsf{Sol}(A')$*. Then there exists an* Apte *exploration* $(\{\lceil A \rceil\};\{\lceil B \rceil\}) \overset{\mathsf{tr}_o}{\rightarrowtail}^{\mathsf{A}} (\mathbf{A}';\mathbf{B}')$ *and some* $A^+ \in \mathbf{A}'$*,* $\theta^+ \in \mathsf{Sol}^+(A^+)$ *such that* $\mathsf{obs}(\mathsf{tr}) = \mathsf{tr}_o$*,* $\llbracket A^+ \rrbracket = A'$ *and* $\lambda_\theta =_{\mathsf{E}} \lambda_{\theta^+}$*, where* $\lambda_\theta$ *(resp.* $\lambda_{\theta^+}$*) is the substitution associated to* $\theta$ *(resp. to* $\theta^+$*) with respect to* $(\Phi';\mathcal{S}')$*. Symmetrically for* $B \overset{\mathsf{tr}}{\Longmapsto} B'$*.*

*Proof.* By hypothesis, we have that $A \overset{\mathsf{tr}}{\Longmapsto} A'$. We will first reorganise this derivation to ensure that $\tau$ actions are always performed as soon as possible. Then, we proceed by induction on $\mathsf{obs}(\mathsf{tr})$. When $\mathsf{obs}(\mathsf{tr})$ is empty, we have that $A' = A$ since $A$ is quiescent. Let $(\mathbf{A}';\mathbf{B}') = (\{\lceil A \rceil\};\{\lceil B \rceil\})$, $A^+ = \lceil A \rceil$, $\theta^+ = \theta$. We have that $\theta \in \mathsf{Sol}(A)$ and therefore $\theta \in \mathsf{Sol}^+(\lceil A \rceil)$, *i.e.* $\theta^+ = \theta \in \mathsf{Sol}^+(A^+)$. We easily conclude.

Otherwise, consider $A \overset{\mathsf{tr}_0}{\Longmapsto} A_1 \overset{\alpha.\tau^k}{\Longmapsto} A'$ with $\theta \in \mathsf{Sol}(A')$. Let $A' = (\mathcal{P}';\Phi';\mathcal{S}')$ and $A_1 = (\mathcal{P}_1;\Phi_1;\mathcal{S}_1)$. We have that $\mathcal{S}_1 \subseteq \mathcal{S}'$. Since $\theta \in \mathsf{Sol}(A')$, we also have $\theta|_V \in \mathsf{Sol}(A_1)$ where $V = \mathrm{vars}^2(\mathcal{S}_1)$. Therefore, we apply our induction hypothesis and we obtain that there exists an Apte exploration $(\{\lceil A \rceil\};\{\lceil B \rceil\}) \overset{\mathsf{tr}'_0}{\rightarrowtail}^{\mathsf{A}} (\mathbf{A}_1;\mathbf{B}_1)$ and some $A_1^+ \in \mathbf{A}_1$, $\theta_1^+ \in \mathsf{Sol}^+(A_1^+)$ such that

$\mathsf{obs}(\mathsf{tr}_0) = \mathsf{tr}'_0$, $\llbracket A_1^+ \rrbracket = A_1$, and the first-order substitutions associated to $\theta|_V$ and $\theta_1^+$ with respect to $(\Phi_1; \mathcal{S}_1)$ are identical. By hypothesis we have $(\mathcal{P}; \Phi) \approx^{\mathsf{A}} (\mathcal{Q}; \Psi)$, thus $\mathbf{A}_1 \sim^+ \mathbf{B}_1$. Hence a $\mapsto^{\mathsf{A1}}$ transition can take place on that pair. By definition of $\mapsto^{\mathsf{A1}}$ and since $\llbracket A_1^+ \rrbracket = A_1 \stackrel{\alpha.\tau^k}{\Longmapsto} A'$ with $A'$ quiescent, there must be some $(\mathbf{A}_1; \mathbf{B}_1) \stackrel{\alpha}{\mapsto}^{\mathsf{A1}} (\mathbf{A}_2; \mathbf{B}_2)$ with $A_2^+ \in \mathbf{A}_2$, $\llbracket A_2^+ \rrbracket = A'$. Thus $\theta \in \mathsf{Sol}(A_2^+)$ and we can apply Axiom 2. There exists $(\mathbf{A}_2; \mathbf{B}_2) \mapsto^{\mathsf{A2}} (\mathbf{A}'; \mathbf{B}')$, $A^+ \in \mathbf{A}'$, $\llbracket A^+ \rrbracket = \llbracket A_2^+ \rrbracket$ and $\theta^+ \in \mathsf{Sol}^+(A^+)$ such that $\llbracket A_2^+ \rrbracket = \llbracket A^+ \rrbracket$, and the substitutions associated to $\theta$ (resp. $\theta^+$) w.r.t. $(\Phi'; \mathcal{S}')$ coincide. To sum up, the exploration

$$(\{\lceil A \rceil\}; \{\lceil B \rceil\}) \stackrel{\mathsf{tr}'_0}{\longmapsto}^{\mathsf{A}} (\mathbf{A}_1; \mathbf{B}_1) \stackrel{\alpha}{\mapsto}^{\mathsf{A1}} (\mathbf{A}_2; \mathbf{B}_2) \mapsto^{\mathsf{A2}} (\mathbf{A}'; \mathbf{B}')$$

together with $A^+ \in \mathbf{A}'$, and $\theta^+ \in \mathsf{Sol}^+(A^+)$ satisfy all the hypotheses. □

We now prove that Apte discards the same second-order solutions in a given exploration. To put in other words, if an execution with an associated second-order solution $\theta$ is captured by an exploration and $\theta$ is not discarded by this exploration (possibly for another execution) then the former is captured by the latter *with this* $\theta$.

**Lemma 18.** *Let $A, B, A'$ be quiescent symbolic processes such that $A \stackrel{\mathsf{tr}}{\Longmapsto} A' = (\mathcal{P}'; \Phi'; \mathcal{S}')$, $\theta \in \mathsf{Sol}(A')$ and $(\{\lceil A \rceil\}; \{\lceil B \rceil\}) \stackrel{\mathsf{tr}_o}{\longmapsto}^{\mathsf{A}} (\mathbf{A}'; \mathbf{B}')$ with $\mathsf{obs}(\mathsf{tr}) = \mathsf{tr}_o$ and $\theta \in \mathsf{Sol}^+(C)$ for some $C \in \mathbf{A}' \cup \mathbf{B}'$. Then there exists some $A^+ \in \mathbf{A}'$ such that $\llbracket A^+ \rrbracket = A'$ and $\theta \in \mathsf{Sol}^+(A^+)$. Symmetrically for $B \stackrel{\mathsf{tr}}{\Longmapsto} B'$.*

*Proof.* We proceed by induction on $\mathsf{tr}_o$. When $\mathsf{tr}_o$ is empty, we have that $A' = A$ (because $A$ is quiescent), $\mathbf{A}' = \{\lceil A \rceil\}$, and $\mathbf{B}' = \{\lceil B \rceil\}$. Let $A^+$ be $\lceil A \rceil = \lceil A' \rceil$. We deduce that $\theta \in \mathsf{Sol}^+(A^+)$ from the fact that $\theta \in \mathsf{Sol}(A)$ and $A^+ = \lceil A \rceil$.

We consider now the case of a non-empty execution:

$$(\{\lceil A \rceil\}; \{\lceil B \rceil\}) \stackrel{\mathsf{tr}_o}{\longmapsto}^{\mathsf{A}} (\mathbf{A}_1; \mathbf{B}_1) \stackrel{\alpha}{\mapsto}^{\mathsf{A1}} (\mathbf{A}_2; \mathbf{B}_2) \mapsto^{\mathsf{A2}} (\mathbf{A}_3; \mathbf{B}_3) \text{ and } A \stackrel{\mathsf{tr}}{\Longmapsto} A_1 \stackrel{\alpha.\tau^k}{\Longmapsto} A_3.$$

Note that, by reordering $\tau$ actions, we can assume $A_1$ to be quiescent. By assumption we have $\theta \in \mathsf{Sol}(A_3)$, $\mathsf{obs}(\mathsf{tr}) = \mathsf{tr}_o$ and $\theta \in \mathsf{Sol}^+(C_3)$ for some $C_3 \in \mathbf{A}_3 \cup \mathbf{B}_3$. By Axiom 1, there exists some $C_2 \in \mathbf{A}_2 \cup \mathbf{B}_2$ such that $\theta \in \mathsf{Sol}^+(C_2)$. By definition of $\mapsto^{\mathsf{A1}}$ we obtain $C_1 \in \mathbf{A}_1 \cup \mathbf{B}_1$ such that $\llbracket C_1 \rrbracket \stackrel{\alpha.\tau^k}{\Longmapsto} \llbracket C_2 \rrbracket$ and $\mathcal{S}^+(C_1) = \mathcal{S}^+(C_2)$ (*i.e.* the sets of extended constraints of $C_1$ and $C_2$ coincide). The first fact implies $\theta|_V \in \mathsf{Sol}(C_1)$ by monotonicity (where $V = \mathrm{vars}^2(\mathcal{S}(C_1))$, *i.e.* second-order variables that occur in the set of non-extended constraints of $C_1$), and the second allows us to conclude more strongly that $\theta|_V \in \mathsf{Sol}^+(C_1)$. Since we also have $\theta|_V \in \mathsf{Sol}(A_1)$ by monotonicity, the induction hypothesis applies and we obtain some $A_1^+ \in \mathbf{A}_1$ with $\llbracket A_1^+ \rrbracket = A_1$ and $\theta|_V \in \mathsf{Sol}^+(A_1^+)$.

By definition of $\mapsto^{\mathsf{A1}}$, and since $\llbracket A_1^+ \rrbracket \stackrel{\alpha.\tau^k}{\Longmapsto} A_3 \not\stackrel{\tau}{\Longmapsto}$ ($A_3$ is quiescent by hypothesis), we have $A_2^+ \in \mathbf{A}_2$ such that $\llbracket A_2^+ \rrbracket = A_3$ and $\mathcal{S}^+(A_1^+) = \mathcal{S}^+(A_2^+)$. Therefore, we have that $\theta \in \mathsf{Sol}(\llbracket A_2^+ \rrbracket)$, and the fact that $\mathcal{S}^+(A_1^+) = \mathcal{S}^+(A_2^+)$ allows us to say that $\theta \in \mathsf{Sol}^+(A_2^+)$. We can finally apply Axiom 3 to obtain some $A_3^+$ such that $\llbracket A_3^+ \rrbracket = \llbracket A_2^+ \rrbracket = A_3$ and $\theta \in \mathsf{Sol}^+(A_3^+)$. □

Lemmas 16 to 18 are the only ingredients needed to show that Apte computes the trace equivalence as shown next.

**Proposition 22.** *For any quiescent extended simple processes, we have that:*

$$A \approx^s B \text{ if, and only if, } A \approx^{\mathsf{A}} B.$$

*Proof.* Let $A_0 = (\mathcal{P}; \Phi)$, $B_0 = (\mathcal{P}'; \Phi')$, $\mathbf{A}_0 = \{(\mathcal{P}; \Phi; \emptyset; \emptyset)\}$ and $\mathbf{B}_0 = \{(\mathcal{P}'; \Phi'; \emptyset; \emptyset)\}$. We prove the two directions separately.

($\Rightarrow$) Assume $A_0 \approx^s B_0$ and consider some exploration $(\mathbf{A}_0; \mathbf{B}_0) \overset{\mathsf{tr}_o}{\mapsto}{}^{\mathsf{A}} (\mathbf{A}; \mathbf{B})$. We shall establish that $\mathbf{A} \prec^+ \mathbf{B}$. Let $A^+ = (\mathcal{P}_A; \Phi_A; \mathcal{S}_A; \mathcal{S}_A^+)$ be in $\mathbf{A}$ and $\theta \in \mathsf{Sol}^+(A^+)$. By Lemma 16, we have $(\mathcal{P}; \Phi; \emptyset) \overset{\mathsf{tr}}{\Longmapsto} \lfloor\!\lfloor A^+ \rfloor\!\rfloor$ such that $\mathsf{obs}(\mathsf{tr}) = \mathsf{tr}_o$. By hypothesis, there exists $B = (\mathcal{P}_B; \Phi_B; \mathcal{S}_B)$ such that $(\mathcal{P}'; \Phi'; \emptyset) \overset{\mathsf{tr}'}{\Longmapsto} B$, $\mathsf{obs}(\mathsf{tr}') = \mathsf{obs}(\mathsf{tr}) = \mathsf{tr}_o$, $\theta \in \mathsf{Sol}(B)$ and $\Phi_B \lambda_\theta^B \sim \Phi_A \lambda_\theta^A$. We can finally apply Lemma 18, which tells us that there must be some $B^+ \in \mathbf{B}$ such that $\lfloor\!\lfloor B^+ \rfloor\!\rfloor = B$ and $\theta \in \mathsf{Sol}^+(B^+)$.

($\Leftarrow$) We now establish $A_0 \sqsubseteq^s B_0$ assuming $A_0 \approx^{\mathsf{A}} B_0$. Consider $(\mathcal{P}; \Phi; \emptyset) \overset{\mathsf{tr}}{\Longmapsto} A$ and $\theta \in \mathsf{Sol}(A)$. If $A$ is not quiescent, it is easy to complete the latter execution into $(\mathcal{P}; \Phi; \emptyset) \overset{\mathsf{tr}.\tau^k}{\Longmapsto} A' = (\mathcal{P}_A; \Phi_A; \mathcal{S}_A)$ and $\theta \in \mathsf{Sol}(A')$ such that $A'$ is quiescent. By Lemma 17 we know that $(\mathbf{A}_0; \mathbf{B}_0) \overset{\mathsf{tr}_o}{\mapsto}{}^{\mathsf{A}} (\mathbf{A}; \mathbf{B})$ with $\mathsf{obs}(\mathsf{tr}) = \mathsf{tr}_o$, $A^+ \in \mathbf{A}$, $\theta^+ \in \mathsf{Sol}^+(A^+)$ with $A' = \lfloor\!\lfloor A^+ \rfloor\!\rfloor$ and $\lambda_\theta =_{\mathsf{E}} \lambda_{\theta^+}$ where $\lambda_\theta$ (resp. $\lambda_{\theta^+}$) is the substitution associated to $\theta$ (resp. $\theta^+$) w.r.t. $(\Phi_A; \mathcal{S}_A)$. By assumption we have $\mathbf{A} \prec^+ \mathbf{B}$ and thus there exists some $B = (\mathcal{P}_B; \Phi_B; \mathcal{S}_B; \mathcal{S}_B^+) \in \mathbf{B}$ with $\theta^+ \in \mathsf{Sol}^+(B)$, and $\Phi_B \lambda_{\theta^+}^B \sim \Phi_A \lambda_{\theta^+}$ where $\lambda_{\theta^+}^B$ is the substitution associated to $\theta^+$ w.r.t. $(\Phi_B; \mathcal{S}_B)$. By Lemma 16 we have $(\mathcal{P}'; \Phi'; \emptyset) \overset{\mathsf{tr}'}{\Longmapsto} \lfloor\!\lfloor B \rfloor\!\rfloor$ with $\mathsf{obs}(\mathsf{tr}') = \mathsf{tr}_o = \mathsf{obs}(\mathsf{tr})$. To conclude the proof, it remains to show that $\theta \in \mathsf{Sol}(\lfloor\!\lfloor B \rfloor\!\rfloor)$ and that $\Phi_A \lambda_\theta \sim \Phi_B \lambda_\theta^B$ where $\lambda_\theta^B$ is the substitution associated to $\theta$ w.r.t. $(\Phi_B; \mathcal{S}_B)$.

For any $X \in \mathrm{vars}^2(\mathcal{S}_B) = \mathrm{vars}^2(\mathcal{S}_A)$, we have $\mathsf{valid}((X\theta)(\Phi_A \lambda_{\theta^+}))$, $\mathsf{valid}((X\theta^+)(\Phi_A \lambda_{\theta^+}))$, and

$$(X\theta)(\Phi_A \lambda_{\theta^+}) \equiv_{\mathsf{E}_d} (X\theta)(\Phi_A \lambda_\theta) \equiv_{\mathsf{E}_d} x_A \lambda_\theta \equiv_{\mathsf{E}_d} x_A \lambda_{\theta^+} \equiv_{\mathsf{E}_d} (X\theta^+)(\Phi_A \lambda_{\theta^+})$$

where $x_A$ is the first-order variable associated to $X$ in $\mathcal{S}_A$. Since $\Phi_A \lambda_{\theta^+} \sim \Phi_B \lambda_{\theta^+}^B$, we deduce that $(X\theta)(\Phi_B \lambda_{\theta^+}^B) \equiv_{\mathsf{E}_d} (X\theta^+)(\Phi_B \lambda_{\theta^+}^B)$, $\mathsf{valid}((X\theta)(\Phi_B \lambda_{\theta^+}^B))$ and therefore $\theta \in \mathsf{Sol}(\lfloor\!\lfloor B \rfloor\!\rfloor)$, and its associated substitution $\lambda_\theta^B$ w.r.t. $(\Phi_B; \mathcal{S}_B)$ coincides with $\lambda_{\theta^+}^B$, and therefore $\Phi_A \lambda_\theta \sim \Phi_B \lambda_\theta^B$ is a direct consequence of $\Phi_B \lambda_{\theta^+}^B \sim \Phi_A \lambda_{\theta^+}$ and $\lambda_\theta =_{\mathsf{E}} \lambda_{\theta^+}$. □

### 6.3.4 Integrating Compression

We now discuss the integration of the symbolic compressed semantics of Section 6.2.2 as a replacement for the regular symbolic semantics in Apte. Since the symbolic compressed semantics does not interact with exchanged data, we are able to simply adapt $\overset{\bullet}{\mapsto}{}_c^{\mathsf{A1}}$ to reflect the compressed strategy.

**Definition 46.** *Given two sets of extended symbolic processes* $\mathbf{A}, \mathbf{B}$*, and an observable action* $\alpha$*, we write* $(\mathbf{A};\mathbf{B}) \overset{\alpha}{\mapsto}{}_c^{\mathsf{A1}} (\mathbf{A}';\mathbf{B}')$ *when*

$$\mathbf{A}' = \bigcup_{(\mathcal{P};\Phi;\mathcal{S};\mathcal{S}^+)\in\mathbf{A}} \left\{ (\mathcal{P}';\Phi';\mathcal{S}';\mathcal{S}^+) \mid (\mathcal{P};\Phi;\mathcal{S}) \overset{\alpha}{\mapsto}_c (\mathcal{P}';\Phi';\mathcal{S}') \overset{\tau}{\not\Rightarrow} \right\},$$

*and similarly for* $\mathbf{B}'$*. We say that* $(\mathbf{A};\mathbf{B}) \overset{\alpha}{\mapsto}{}_c^{\mathsf{A}} (\mathbf{A}'';\mathbf{B}'')$ *when* $(\mathbf{A};\mathbf{B}) \overset{\alpha}{\mapsto}{}_c^{\mathsf{A1}} (\mathbf{A}';\mathbf{B}')$ *and* $(\mathbf{A}';\mathbf{B}') \mapsto^{\mathsf{A2}} (\mathbf{A}'';\mathbf{B}'')$*.*

*Finally, given two simple extended processes* $A = (\mathcal{P}_A;\Phi_A)$ *and* $B = (\mathcal{P}_B;\Phi_B)$*, we say that* $A \approx_c^{\mathsf{A}} B$ *when* $\mathbf{A} \approx^{+} \mathbf{B}$ *for any* $(\{\!|(\mathcal{P}_A;\Phi_A;\emptyset)|\!\}; \{\!|(\mathcal{P}_B;\Phi_B;\emptyset)|\!\}) \overset{\mathsf{tr}}{\mapsto}{}_c^{\mathsf{A}} (\mathbf{A};\mathbf{B})$*.*

As expected, $\mapsto_c^{\mathsf{A1}}$ allows to consider much fewer explorations than with the original $\mapsto^{\mathsf{A1}}$. It inherits the features of compression, prioritising outputs, not considering interleavings of outputs, executing inputs only under focus, and preventing executions beyond improper blocks. These constraints apply to individual processes in $\mathbf{A} \cup \mathbf{B}$, but we remark that they also have a global effect in $\mapsto_c^{\mathsf{A1}}$, *e.g.* all processes of $\mathbf{A} \cup \mathbf{B}$ must start a new block simultaneously: recall that the beginning of a block corresponds to some outputs after some inputs, and after the outputs, no more outputs are available.

**Example 44.** *Continuing Example 43, there is only one non-trivial*[3] *compressed exploration of one action from the valid pair* $(\{Q_1^t\};\{Q_1^t\})$*. It corresponds to the output on channel* $c_2$*:* $(\{Q_1^t\};\{Q_1^t\}) \overset{\mathtt{out}(c_2,w_2)}{\mapsto}{}_c^{\mathsf{A}} (\{Q_2\},\{Q_2\})$ *for* $Q_2 = (\{R_1\};\{w_2 \mapsto n_2\};\{\emptyset \vdash^?_{X_2} x_2, x_2 =^? \mathsf{ok}\};\mathcal{S}_2^+)$*. In particular, for any* $i \in \{1,2\}$*, we have* $(\{Q_1^t\};\{Q_1^t\}) \overset{\mathtt{in}(c_i,X_i)}{\mapsto}{}_c^{\mathsf{A}} (\emptyset;\emptyset)$*.*

Observe that, because $\mapsto_c^{\mathsf{A}}$ is obtained from $\mapsto^{\mathsf{A}}$ by a subset restriction in $\mapsto^{\mathsf{A1}}$ up to some disequality constraints, we have that $(\mathbf{A}';\mathbf{B}')$ is a valid pair when $(\{\!|A|\!\};\{\!|B|\!\}) \overset{\mathsf{tr}}{\mapsto}{}_c^{\mathsf{A}} (\mathbf{A}';\mathbf{B}')$ for some quiescent, symbolic processes $A, B$ having empty sets of constraints. Following the same reasoning as the one performed in Section 6.3.3, we can establish that $\approx_c^s$ coincides with $\approx_c^{\mathsf{A}}$. The main difference is that $\mapsto_c$ already ignores $\tau$-actions, and therefore we do not need to apply the $\mathsf{obs}(\cdot)$ operator.

**Lemma 19.** *Let* $(\mathbf{A};\mathbf{B})$ *be a valid pair such that* $(\mathbf{A};\mathbf{B}) \overset{\mathsf{tr}}{\mapsto}{}_c^{\mathsf{A}} (\mathbf{A}';\mathbf{B}')$*. Then, for all* $A' \in \mathbf{A}'$ *there is some* $A \in \mathbf{A}$ *such that* $\lfloor\!\lfloor A \rfloor\!\rfloor \overset{\mathsf{tr}}{\mapsto}_c \lfloor\!\lfloor A' \rfloor\!\rfloor$*. Symmetrically for* $B' \in \mathbf{B}'$*.*

**Lemma 20.** *Let* $A = (\mathcal{P};\Phi;\emptyset)$*,* $B = (\mathcal{Q};\Psi;\emptyset)$*, and* $A' = (\mathcal{P}';\Phi';\mathcal{S}')$ *be three quiescent, symbolic processes such that* $(\mathcal{P};\Phi) \approx_c^{\mathsf{A}} (\mathcal{Q};\Psi)$*,* $A \overset{\mathsf{tr}}{\mapsto}_c A'$ *and* $\theta \in \mathsf{Sol}(A')$*. Then there exists an exploration* $(\{\!|A|\!\};\{\!|B|\!\}) \overset{\mathsf{tr}}{\mapsto}{}_c^{\mathsf{A}} (\mathbf{A}';\mathbf{B}')$ *and some* $A^+ \in \mathbf{A}'$*,* $\theta^+ \in \mathsf{Sol}^+(A^+)$ *such that* $\lfloor\!\lfloor A^+ \rfloor\!\rfloor = A'$ *and* $\lambda_\theta =_{\mathsf{E}} \lambda_{\theta^+}$*, where* $\lambda_\theta$ *(resp.* $\lambda_{\theta^+}$*) is the substitution associated to* $\theta$ *(resp. to* $\theta^+$*) with respect to* $(\Phi';\mathcal{S}')$*. Symmetrically for* $B \overset{\mathsf{tr}}{\mapsto}_c B'$*.*

**Lemma 21.** *Let* $A, B$ *and* $A'$ *be quiescent, simple symbolic processes such that* $A \overset{\mathsf{tr}}{\mapsto}_c A' = (\mathcal{P}';\Phi';\mathcal{S}')$*,* $\theta \in \mathsf{Sol}(A')$*, and* $(\{\!|A|\!\};\{\!|B|\!\}) \overset{\mathsf{tr}}{\mapsto}{}_c^{\mathsf{A}} (\mathbf{A}';\mathbf{B}')$ *with* $\theta \in \mathsf{Sol}^+(C)$ *for some* $C \in \mathbf{A}' \cup \mathbf{B}'$*. Then there exists some* $A^+ \in \mathbf{A}'$ *such that* $\lfloor\!\lfloor A^+ \rfloor\!\rfloor = A'$ *and* $\theta \in \mathsf{Sol}^+(A^+)$*. Symmetrically for* $B \overset{\mathsf{tr}}{\mapsto}_c B'$*.*

[3] We dismiss here the (infinitely many) transitions obtained for infeasible actions, which yield $(\emptyset;\emptyset)$.

**Theorem 6.** *For any quiescent extended simple processes, we have that:*

$$\lceil A \rceil \approx_c^s \lceil B \rceil \text{ if, and only if, } A \approx_c^{\mathsf{A}} B.$$

### 6.3.5 Integrating Dependency Constraints

We now define a final variant of Apte explorations, which integrates the reduction strategy to further reduce redundant explorations. We can obviously generate dependency constraints in Apte, just like we did in Section 6.2.3, but the real difficulty is to exploit them in constraint resolution to prune some branches of the exploration performed by Apte. Roughly, we shall simply stop the exploration when reaching a state for which we know that all of its solutions violate dependency constraints. To do that, we rely on the notion of partial solution introduced in Section 6.3.1. In other words, we do not modify Apte's constraint resolution, but simply rely on information that it already provides to know when dependency constraints become unsatisfiable. As we shall see in Section 6.4, this simple strategy is very satisfying in practice.

**Definition 47.** *We define $\mapsto_r^{\mathsf{A}}$ as the greatest relation contained in $\mapsto_c^{\mathsf{A}}$ and such that, for any symbolic processes $A$ and $B$ with empty constraint sets, $(\{\lceil A \rceil\};\{\lceil B \rceil\}) \overset{\mathsf{tr}}{\mapsto}_r^{\mathsf{A}} (\mathbf{A}';\mathbf{B}')$ implies that there is no $\overline{X} \ltimes \overline{w} \in \mathrm{Deps}(\mathsf{tr})$ such that for all $X_i \in \overline{X}$ we have $X_i \in \mathrm{dom}(\mathsf{ps}(\mathbf{A}';\mathbf{B}'))$, and $\overline{w} \cap \mathrm{vars}^1(X_i \mathsf{ps}(\mathbf{A}';\mathbf{B}')) = \emptyset$.*

*Finally, given two simple extended process $A = (\mathcal{P}_A;\Phi_A)$ and $B = (\mathcal{P}_B;\Phi_B)$, we say that $A \approx_r^{\mathsf{A}} B$ when $\mathbf{A} \sim^+ \mathbf{B}$ for any pair $(\mathbf{A};\mathbf{B})$ such that $((\mathcal{P}_A;\Phi_A;\emptyset;\emptyset);(\mathcal{P}_B;\Phi_B;\emptyset;\emptyset)) \overset{\mathsf{tr}}{\mapsto}_r^{\mathsf{A}} (\mathbf{A};\mathbf{B})$.*

**Example 45.** *Continuing Example 44, consider the following compressed exploration, where $Q_3$ contains the constraints $\emptyset \vdash^?_{X_2} x_2$, $\{w_2\} \vdash^?_{X_1} x_1$, $x_2 =^? \mathsf{ok}$ and $x_1 =^? \mathsf{ok}$:*

$$\begin{aligned}(\{Q_0\};\{Q_0\}) \quad &\overset{\mathtt{foc}(\mathtt{in}(c_2,X_2))}{\mapsto}{}^{\mathsf{A}}_c \overset{\mathtt{out}(c_2,w_2).\mathtt{rel}}{\mapsto}{}^{\mathsf{A}}_c (\{Q_2\};\{Q_2\})\\ &\overset{\mathtt{foc}(\mathtt{in}(c_1,X_1))}{\mapsto}{}^{\mathsf{A}}_c \overset{\mathtt{out}(c_1,w_1).\mathtt{rel}}{\mapsto}{}^{\mathsf{A}}_c (\{Q_3\};\{Q_3\}).\end{aligned}$$

*Assuming that $\mathsf{ps}(Q_3) = \{X_2 \mapsto \mathsf{ok}, X_1 \mapsto \mathsf{ok}\}$ (which is the case in the actual Apte procedure) this compressed exploration is not explored by $\mapsto_r^{\mathsf{A}}$ because*

$$X_1 \ltimes w_2 \in \mathrm{Deps}(\mathtt{io}_{c_2}(X_2,w_2).\mathtt{io}_{c_1}(X_1,w_1)),\ X_1\mathsf{ps}(Q_3) = \mathsf{ok} \ \text{ and } \ \{w_2\} \cap \mathrm{vars}^1(\mathsf{ok}) = \emptyset.$$

Below, we show that all reduced executions are captured by $\dot{\mapsto}_r^{\mathsf{A}}$.

**Lemma 22.** *Let $A = (\mathcal{P};\Phi;\emptyset)$, $B = (\mathcal{Q};\Psi;\emptyset)$ and $A' = (\mathcal{P}';\Phi';\mathcal{S}')$ be quiescent, simple symbolic processes such that $(\mathcal{P};\Phi) \approx_r^{\mathsf{A}} (\mathcal{Q};\Psi)$, $A \overset{\mathsf{tr}}{\mapsto}_c A'$, $\theta \in \mathsf{Sol}(A')$ and $\theta \models_{(\Phi';\mathcal{S}')} \mathrm{Deps}(\mathsf{tr})$. Then there exists an exploration $(\{\lceil A \rceil\};\{\lceil B \rceil\}) \overset{\mathsf{tr}}{\mapsto}_r^{\mathsf{A}} (\mathbf{A}';\mathbf{B}')$ and some $A^+ \in \mathbf{A}'$, $\theta^+ \in \mathsf{Sol}^+(A^+)$ such that $\lfloor\!\lfloor A^+ \rfloor\!\rfloor = A'$ and $\lambda_\theta =_{\mathsf{E}} \lambda_{\theta^+}$, where $\lambda_\theta$ (resp. $\lambda_{\theta^+}$) is the substitution associated to $\theta$ (resp. to $\theta^+$) with respect to $(\Phi';\mathcal{S}')$. Symmetrically for $B \overset{\mathsf{tr}}{\mapsto}_c B'$.*

*Proof.* We proceed by induction on $\mathsf{tr}$. The empty case is easy. Otherwise, consider $A \overset{\mathsf{tr}}{\mapsto}_c A_1 \overset{\alpha}{\mapsto}_c A_3 = (\mathcal{P}_3;\Phi_3;\mathcal{S}_3)$ with $\theta \in \mathsf{Sol}(A_3)$, $A_1, A_3$ quiescent, and $\theta \models_{(\Phi_3;\mathcal{S}_3)} \mathrm{Deps}(\mathsf{tr}.\alpha)$. Let $A_1 = (\mathcal{P}_1;\Phi_1;\mathcal{S}_1)$ and $V_1 = \mathrm{vars}^2(\mathcal{S}_1)$. We also have $\theta|_{V_1} \in \mathsf{Sol}(A_1)$ and $\theta|_{V_1} \models_{(\Phi_1;\mathcal{S}_1)} \mathrm{Deps}(\mathsf{tr})$,

so the induction hypothesis applies and we obtain $(\{\!|\!\lceil A\rceil\!|\!\}; \{\!|\!\lceil B\rceil\!|\!\}) \overset{\mathsf{tr}}{\mapsto}{}^{\mathsf{A}}_{r} (\mathbf{A}_1; \mathbf{B}_1)$ with $A_1^+ \in \mathbf{A}_1$, $\lfloor\!|A_1^+|\!\rfloor = A_1$ and $\theta_1^+ \in \mathsf{Sol}^+(A_1^+)$ such that the first-order substitutions associated to $\theta|_{V_1}$ and $\theta_1^+$ w.r.t. $(\Phi_1; \mathcal{S}_1)$ coincide.

By hypothesis we have $A \approx^{\mathsf{A}}_r B$, thus $\mathbf{A}_1 \sim^+ \mathbf{B}_1$. Hence a $\mapsto^{\mathsf{A1}}_c$ transition can take place on that pair. By definition of $\mapsto^{\mathsf{A1}}_c$ and since $\lfloor\!|A_1^+|\!\rfloor = A_1 \overset{\alpha}{\mapsto}_c A_3$, there must be some $(\mathbf{A}_1; \mathbf{B}_1) \overset{\alpha}{\mapsto}{}^{\mathsf{A1}}_{c} (\mathbf{A}_2; \mathbf{B}_2)$ with $A_2^+ \in \mathbf{A}_2$, $\lfloor\!|A_2^+|\!\rfloor = A_3$. Thus $\theta \in \mathsf{Sol}(A_2^+)$ and we can apply Axiom 2 to obtain $(\mathbf{A}_2; \mathbf{B}_2) \mapsto^{\mathsf{A2}} (\mathbf{A}_3; \mathbf{B}_3)$ with $A_3^+ \in \mathbf{A}_3$, $\lfloor\!|A_3^+|\!\rfloor = \lfloor\!|A_2^+|\!\rfloor$ and $\theta_3^+ \in \mathsf{Sol}^+(A_3^+)$ such that the substitutions associated to $\theta$ and $\theta_3^+$ w.r.t. $(\Phi_3; \mathcal{S}_3)$ coincide.

It only remains to show that this extra execution step in $\mapsto^{\mathsf{A}}_c$ is also present in $\mapsto^{\mathsf{A}}_r$, *i.e.* that $\mathsf{ps}(\mathbf{A}_3; \mathbf{B}_3)$ does not violate $\mathrm{Deps}(\mathsf{tr}.\alpha)$ in the sense of Definition 47. This is because, by definition of the partial solution, we have that $\theta_3^+ = \mathsf{ps}(\mathbf{A}_3; \mathbf{B}_3) \sqcup \tau$ for some $\tau$, so that if $\mathsf{ps}(\mathbf{A}_3; \mathbf{B}_3)$ violated $\mathrm{Deps}(\mathsf{tr}.\alpha)$ then we would have $\theta_3^+ \not\models_{(\Phi_3;\mathcal{S}_3)} \mathrm{Deps}(\mathsf{tr}.\alpha)$. Since $\theta_3^+$ and $\theta$ induce the same first-order substitutions with respect to $(\Phi_3; \mathcal{S}_3)$, we would finally have $\theta \not\models_{(\Phi_3;\mathcal{S}_3)} \mathrm{Deps}(\mathsf{tr}.\alpha)$, contradicting the hypothesis on $\theta$. □

Finally, we show the main theorem of this chapter: Apte implementing the reduced strategy as explained above computes the trace equivalence for quiescent simple processes.

**Theorem 7.** *For any quiescent initial simple processes $A$ and $B$ such that $\mathsf{skl}(A) = \mathsf{skl}(B)$, we have that:*

$$A \approx B \text{ if, and only if, } A \approx^{\mathsf{A}}_r B.$$

*Proof.* Let $A = (\mathcal{P}; \Phi)$ and $B = (\mathcal{Q}; \Psi)$ be two quiescent, initial simple processes such that $\mathsf{skl}(A) = \mathsf{skl}(B)$. We prove the two directions separately.

($\Rightarrow$) Applying Corollary 4 and theorem 6, we have that $A \approx B$ implies $A \approx^{\mathsf{A}}_c B$. Then, we easily deduce $A \approx^{\mathsf{A}}_r B$. Indeed, for any $(\{\!|\!\lceil(\mathcal{P}; \Phi; \emptyset)\rceil\!|\!\}; \{\!|\!\lceil(\mathcal{Q}; \Psi; \emptyset)\rceil\!|\!\}) \overset{\mathsf{tr}}{\mapsto}{}^{\mathsf{A}}_{r} (\mathbf{A}'; \mathbf{B}')$ we have $(\{\!|\!\lceil(\mathcal{P}; \Phi; \emptyset)\rceil\!|\!\}; \{\!|\!\lceil(\mathcal{Q}; \Psi; \emptyset)\rceil\!|\!\}) \overset{\mathsf{tr}}{\mapsto}{}^{\mathsf{A}}_{c} (\mathbf{A}'; \mathbf{B}')$ by definition of $\mapsto^{\mathsf{A}}_r$, and thus $\mathbf{A}' \sim^+ \mathbf{B}'$ by hypothesis.

($\Leftarrow$) For the other direction, it suffices to show that $A \approx^{\mathsf{A}}_r B$ implies $A \sqsubseteq^s_r B$ (by Corollary 5). Let $(\mathcal{P}; \Phi; \emptyset) \overset{\mathsf{tr}}{\mapsto}_c A' = (\mathcal{P}'; \Phi'; \mathcal{S}')$ with $\theta \in \mathsf{Sol}(A')$ and $\theta \models_{(\Phi';\mathcal{S}')} \mathrm{Deps}(\mathsf{tr})$. By Lemma 22, we have $(\{\!|\!\lceil(\mathcal{P}; \Phi; \emptyset)\rceil\!|\!\}; \{\!|\!\lceil(\mathcal{Q}; \Psi; \emptyset)\rceil\!|\!\}) \overset{\mathsf{tr}}{\mapsto}{}^{\mathsf{A}}_{r} (\mathbf{A}'; \mathbf{B}')$ with $A^+ \in \mathbf{A}'$, $\theta^+ \in \mathsf{Sol}^+(A^+)$ such that $\lfloor\!|A^+|\!\rfloor = A'$ and $\lambda_\theta^{A'} =_{\mathsf{E}} \lambda_{\theta^+}^{A'}$ where $\lambda_\theta^{A'}$ (resp. $\lambda_{\theta^+}^{A'}$) is the substitution associated to $\theta$ (resp. $\theta^+$) w.r.t. $(\Phi'; \mathcal{S}')$.

Since $A \approx^{\mathsf{A}}_r B$, we have $\mathbf{A}' \sim^+ \mathbf{B}'$: there must be some $B^+ = (\mathcal{P}_{B'}; \Phi_{B'}; \mathcal{S}_{B'}; \mathcal{S}_B^+) \in \mathbf{B}'$ such that $\theta^+ \in \mathsf{Sol}^+(B^+)$ and $\Phi'\lambda_{\theta^+}^{A'} \sim \Phi_{B'}\lambda_{\theta^+}^{B'}$ where $\lambda_{\theta^+}^{B'}$ is the substitution associated to $\theta^+$ w.r.t. $(\Phi_{B'}; \mathcal{S}_{B'})$. By Lemma 19, we have $(\mathcal{Q}; \Psi; \emptyset) \overset{\mathsf{tr}}{\mapsto}_c \lfloor\!|B^+|\!\rfloor$. Furthermore, we can show as before (see the end of the proof of Proposition 22) that $\theta \in \mathsf{Sol}(B^+)$ and $\Phi'\lambda_\theta^{A'} \sim \Phi_{B'}\lambda_\theta^{B'}$, where $\lambda_\theta^{B'}$ is the substitution associated to $\theta$ w.r.t. $(\Phi_{B'}; \mathcal{S}_{B'})$. Finally, by $\theta \models_{(\Phi';\mathcal{S}')} \mathrm{Deps}(\mathsf{tr})$, $D_{(\Phi';\mathcal{S}')} = D_{(\Phi_{B'};\mathcal{S}_{B'})}$ (*i.e.* sets of handles that second-order variables may use coincide), and $\Phi'\lambda_\theta^{A'} \sim \Phi_{B'}\lambda_\theta^{B'}$, we obtain that $\theta \models_{(\Phi_{B'};\mathcal{S}_{B'})} \mathrm{Deps}(\mathsf{tr})$. □

## 6.4 Implementation and Benchmarks

The POR techniques we presented in this part have been implemented, following the above approach, in the official version of Apte [aptb]. We discuss the implementation in Subsection 6.4.1 and provide and comment on benchmarks in Subsection 6.4.2.

### 6.4.1 Implementation

In practice, many processes enjoy a nice property that allows one to ensure that blocking outputs will never occur: it is often the case that enough tests are performed before outputting a term to ensure its validity.

**Example 46.** *Consider the following process, where $k'$ is assumed to be valid (*e.g. *because it is a pure constructor term):*

$$\mathtt{in}(c, x).\mathsf{if}\ \mathsf{sdec}(x, k) = \mathsf{hash}(u)\ \mathsf{then}\ \mathtt{out}(c, \mathsf{senc}(\mathsf{sdec}(x, k), k'))$$

*The term outputted during an execution is necessarily valid thanks to the test that is performed just before this output.*

We exploit this property in order to avoid adding additional disequalities when integrating compression in Apte. Therefore, in this section, we will restrict ourselves to simple processes that are *non-blocking* as defined below.

**Definition 48.** *Let $(\mathcal{P}; \Phi)$ be a simple process. We say that $(\mathcal{P}; \Phi)$ is* non-blocking *if $u$ is valid for any $\mathsf{tr}$, $c$, $u$, $Q'$, $\mathcal{Q}$, $\Psi$ such that $(\mathcal{P}; \Phi) \xrightarrow{\mathsf{tr}} (\{\mathtt{out}(c, u).Q'\} \cup \mathcal{Q}; \Psi)$.*

This condition may be hard to check in general, but it is actually quite easy to see that it is satisfied on all of our examples. Roughly, enough tests are performed before any output action, and this ensures the validity of the term when the output action becomes reachable as in Example 46.

We implemented $\overset{\bullet}{\mapsto}{}^{\mathsf{A}}_{c}$ and $\overset{\bullet}{\mapsto}{}^{\mathsf{A}}_{r}$ for non-blocking processes in the main development line of the tool Apte distributed at [aptb]. Those optimisations can be enabled using an option

```
−with_por [compr|red] [improper].
```

It is thus possible to use the compressed or the reduced semantics with or without the "semi" variant that does not stop exploring after an improper block. The modifications of the code ($\approx$ 2kloc of OCaml) are summarised at `https://github.com/LCBH/APTE/compare/ref...APTE:POR`. Sources and instructions for reproduction of the following benchmarks are openly available [Hira].

### 6.4.2 Benchmarks

We now report on experimental results. We ran the tool (compiled with OCaml 3.12.1) on a single 2.67GHz Xeon core with 48GO of RAM and compared three different versions:

- *reference:* the reference version without our optimisations (*i.e.* $\approx^{\mathsf{A}}$);
- *compression:* using only the compression optimisation (*i.e.* $\approx^{\mathsf{A}}_{c}$ );
- *reduction:* using both compression and reduction (*i.e.* $\approx^{\mathsf{A}}_{r}$ ).

For reference, the version of Apte that we are using in the benchmarks below is available at `https://github.com/APTE/APTE/releases/tag/bench-POR-LMCS` together with all benchmark files, in subdirectory `bench/protocols`. More details, including instructions for reproducing our benchmarks are available at `http://www.lsv.fr/~hirschi/apte_por`.

We first show examples in which equivalence holds. They are the most significant, because the time spent on inequivalent processes is too sensitive to the order in which the (depth-first) exploration is performed.

**Toy example.** We consider a parallel composition of $n$ roles $R_i$ as defined in Example 42: $P_n := \Pi_{i=1}^{n} R_i$. When executed in the regular symbolic semantics $\Rightarrow$, the $2n$ actions of $P_n$ may be interleaved in $(2n)!/2^n$ ways in a trace containing all actions. In the compressed symbolic semantics $\mapsto_c$, the actions of individual $R_i$ processes must be bundled in blocks, so there are only $n!$ interleavings containing all actions. In the reduced symbolic semantics $\mapsto_r$, only one interleaving of that length remains: the trace cannot deviate from the priority order, since the only way to satisfy a dependency constraint would be to feed an input with a message that cannot be derived without some previously output nonce $n_i$, but in that case the message will not be ok and the trace won't be explored further. Note that there is still an exponential number of symbolic traces in the reduced semantics when one takes into account traces with less than $2n$ actions.

We show in Figure 6.5 the time needed to verify $P_n \approx P_n$ for $n = 1$ to 22 in the three versions of Apte described above: *reference*, *compression* and *reduction*. The results, in logarithmic scale, show that each of our optimisations brings an exponential speedup, as predicted by our theoretical analysis. Similar improvements are observed if one compares the numbers of explored pairs rather than execution times.

**Denning-Sacco protocol.** We ran a similar benchmark, checking that Denning-Sacco [DS81] ensures strong secrecy in various scenarios. The protocol has three roles and we added processes playing those roles in turn, starting with three processes in parallel. Strong secrecy is expressed by considering, after one of the roles B, the output of a message encrypted with the established key on one side of the equivalence, and with a fresh key on the other side. The results are plotted in Figure 6.6. The fact that we add one role out of three at each step explains the irregular growth in verification time. We still observe an exponential speedup for each optimisation.

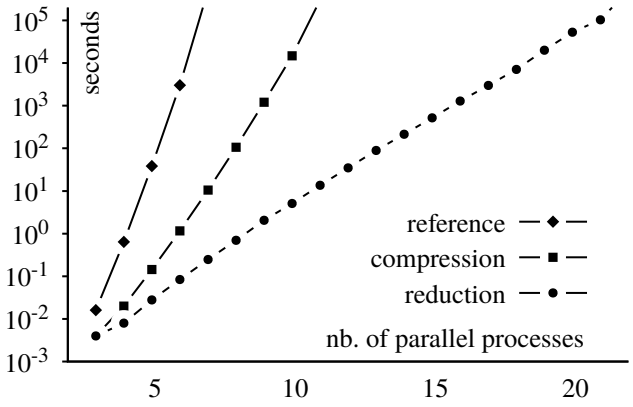


**Figure 6.5** Impact of optimisations on verification time on toy example.

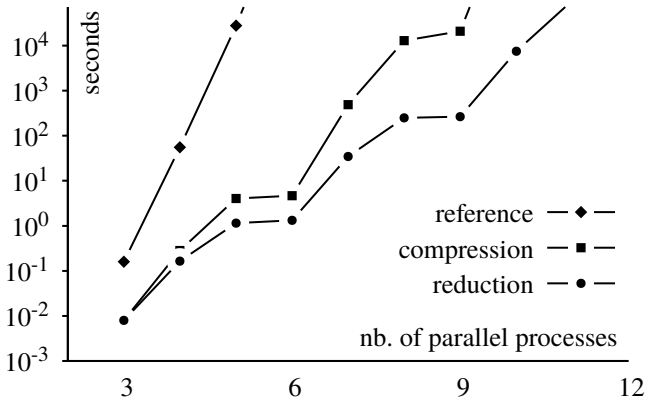


**Figure 6.6** Impact of optimisations on verification time on Denning-Sacco.

**Practical impact.** Finally, we illustrate how our optimisations make Apte much more useful in practice for investigating interesting scenarios. Verifying a single session of a protocol brings little assurance into its security. In order to detect replay attacks and to allow the attacker to compare messages that are exchanged, at least two sessions should be considered. This means having at least four parallel processes for two-party protocols, and six when a trusted third party is involved. This is actually beyond what the unoptimised Apte can handle in a reasonable amount of time. We show in Figure 6.7 how many parallel processes could be handled in 20 hours by Apte on various use cases of protocols, for the same three variants of Apte as before, *i.e. reference*, *compression* and *reduction*. We verify an anonymity property for the Passive Authentication protocol of e-passports. For other protocols, we analyse strong secrecy of established keys: for one of the roles we add, on one side of the equivalence, an output encrypted by the established

key and, on the other side, an output encrypted by a fresh key.

| Protocol | reference | compression | reduction |
|---|---|---|---|
| Needham Schroeder (3-party) | 4 | 6 | 7 |
| Private Authentication (2-party) | 4 | 7 | 7 |
| Yahalom (3-party) | 4 | 5 | 5 |
| E-Passport PA (2-party) | 4 | 7 | 9 |
| Denning-Sacco (3-party) | 5 | 9 | 10 |
| Wide Mouth Frog (3-party) | 6 | 12 | 13 |

**Figure 6.7** Maximum number of parallel processes verifiable in 20 hours.

We finally present the benefits of our optimisations for discovering attacks. We performed some experiments on flawed variants of protocols, shown in Figure 6.8, corresponding to example files in subdirectory `bench/protocols/attacks/` of the above mentioned release. The scenario *Denning-Sacco A* expresses strong secrecy of the (3-party) Denning-Sacco protocol, but this time on two instances of roles at the same time (instead of one as in Figure 6.7). In *Denning-Sacco B*, we consider again a form of strong secrecy expressed by outputting encrypted messages but this time at the end of role B. The *Needham-Schroeder pub* scenario corresponds to strong secrecy of the public-key Needham-Schroder protocol. The *E-Passport PA exposed* experiments show that anonymity is (obviously) lost with the Passive Authentication protocol when the secret key is made public. Similarly, the *Yahalom exposed* experiment shows that strong secrecy of Yahalom is lost when secrets keys are revealed. Since Apte stops its exploration as soon as an attack is found, the time needed for Apte to find the attack highly depends on the order in which the depth-first exploration is performed. However, as shown in Figure 6.8, we always observe in practice dramatic improvements brought by our optimisations compared to the reference version of Apte. In some cases, our optimisations are even mandatory for Apte to find the attack using reasonable resources.

| Protocol | reference | compression | reduction |
|---|---|---|---|
| Denning-Sacco A (6 proc.) | OoM | 0.07s | 0.02s |
| Denning-Sacco B (6 proc.) | 5.83s | 0.04s | 0.04s |
| Needham-Shroeder pub (7 proc.) | TO | 0.77s | 0.67s |
| Needham-Shroeder pub (5 proc.) | 0.79s | 0.21s | 0.13s |
| E-Passport PA exposed (8 proc.) | TO | 0.02s | 0.02s |
| E-Passport PA exposed (6 proc.) | 4.37s | 0.03s | 0.02s |
| Yahalom exposed (4 proc.) | 7.24s | 0.02s | 0.02s |

**Figure 6.8** Impact of optimisations for finding attacks ("X proc." stands for X parallel processes, OoM denotes a consumption of >32Go of RAM and TO denotes a running time of >20 hours).

## 6.5 Conclusion

We have successfully put our two POR techniques into practice. We have shown that compression can be easily lifted to the symbolic setting and that reduction can be embedded into the symbolic

setting by means of a new simple kind of contraints. We have fully described how we integrated those techniques in the state-of-the-art tool Apte and we have proved its correction. Finally, we have provided experimental results showing that the resulting optimised tool Apte has now much better performance.

We believe that the interest of our POR techniques goes beyond the (important) speedups we were able to bring to the tool Apte. Indeed, we believe that other tools could benefit from the same important improvements by reusing the methodology developed in this chapter. We also think that our POR techniques could be adapted for other settings than the constraint solving one. We support those claims in related future work below.

**Future Work.** In addition to the future work we already discussed in Section 5.5, we now mention some avenues for future developments specific to the practical aspects of our POR techniques.

We first note that our compression technique should be applicable and useful in other verification tools, not necessarily based on symbolic execution or constraint solving. First, we have conducted [Hir13, Hirb] an experimental implementation in the tool Spec [TD10] of a preliminary version of our compression and reduction techniques. But this project has been abandoned because of the drawbacks of Spec over Apte (see Subsection 1.4.1 for a comprehensive comparison with Apte) and the complexity of the correction argument.

More interestingly, it is noteworthy that our compression technique has already been independently implemented[4] in the tool Akiss [CCK12] (it even is its default behavior) whereas this tool is not based on constraint solving but is rather based on a dedicated Horn clauses abstraction and a resolution procedure (see Subsection 1.4.1 for a more comprehensive description and comparison with Apte). This has been made possible because our compression technique boils down to a constrained exploration strategy taking only nature of available actions into account.

Similarly, one could seek for adapting our techniques for verification methods based on *backward search* where explorations start with attack states (*e.g.* Tamarin, Maude−NPA) instead of *forward search* where explorations start with the initial state as is the case in our framework.

Finally, one could easily extend the class of simple processes towards action-deterministic processes; notably allowing (nested) parallel compositions and non-action-deterministic prefixes at the beginning of processes. We already have implemented those extensions in the tool Apte but we have not yet proved their correction.

[4] `https://github.com/akiss/akiss`

Chapter 7

# Related Work

The techniques we have presented in this Part borrow from standard ideas from concurrency theory, trace theory, and, perhaps more surprisingly, proof theory. Blending all these ingredients, and adapting them to the demanding framework of security protocols, we have come up with partial order reduction techniques that can effectively be used in symbolic verification algorithms for equivalence properties of security protocols. We now discuss related work, and there is a lot of it given the huge success of POR techniques in various application areas. We shall focus on the novel aspects of our approach, and explain why such techniques have not been needed outside of security protocol analysis. These observations are not new: as pointed out by Baier and Katoen [BK08], "[POR] is mainly appropriate to control-intensive applications and less suited for data-intensive applications"; Clarke *et al.* [CJM00] also remark that "In the domain of model checking of reactive systems, there are numerous techniques for reducing the state space of the system. One such technique is partial-order reduction. This technique does not directly apply to [security protocol analysis] because we explicitly keep track of knowledge of various agents, and our logic can refer to this knowledge in a meaningful way." In Section 7.1, we first compare our work with classical POR techniques. We then comment on previous work in the domain of security protocol analysis in Section 7.2. We conclude the chapter with some remarks on the relationship between our optimised semantics and focused proof systems in Section 7.3.

## 7.1 Classical POR

Partial order reduction techniques have proved very useful in the domain of model checking concurrent programs. Given a Labelled Transition System (LTS) and some property to check (*e.g.* a Linear Temporal Logic formula), the basic idea of POR [Pel98, GvLH$^+$96, BK08] is to only consider a reduced version of the given LTS whose enabled transitions of some states might be not exhaustive but are such that this transformation does not affect the property. POR techniques can be categorised in two groups [GvLH$^+$96]. First, the *persistent set* techniques (*e.g. stubborn sets, ample sets*) where only a sufficiently representative subset of available transitions is explored. Second, *sleep set* techniques memoize past exploration and use this information

along with available transitions to disable some provably redundant transitions. Note that these two kinds of techniques are compatible, and are indeed often combined to obtain better reductions. Theoretical POR techniques [GvLH+96] apply to transition systems which may not be explicitly available in practice, or whose explicit computation may be too costly. In such cases, POR is often applied to an approximation of the LTS that is obtained through static analysis. Another, more recent approach is to use *dynamic POR* [FG05, TKL+12, AAJS14] where the POR arguments are applied based on information that is obtained during the execution of the system.

Clearly, classical POR techniques would apply to our concrete LTS, but that would not be practically useful since this LTS is wildly infinite, taking into account all recipes that the attacker could build. Applying most classical POR techniques to the LTS from which data would have been abstracted away (as it is the case with symbolic semantics; see *e.g.* Section 6.2) would be ineffective: any input would be dependent on any output (since the attacker's knowledge, increased by the output, may enable new input messages). Our compression technique lies between these two extremes. It exploits a semi-commutation property: outputs can be permuted before inputs, but not the converse in general. Further, it exploits the fact that inputs do not increase the attacker's knowledge, and can thus be executed in a chained fashion, under focus. The semi-commutation is reminiscent of the asymmetrical dependency analysis enabled by the *conditional* stubborn set technique [GvLH+96], and the execution of inputs under focus may be explained by means of sleep sets. While it may be possible to formally derive our compressed semantics by instantiating abstract POR techniques to our setting, we have not explored this possibility in detail[1]. Concerning our reduced semantics, it may be seen as an application of the sleep set technique [GvLH+96] (or even as a reformulation of Anisimov's and Knuth's characterisation of lexicographic normal forms [AK79]) but the real contribution with this technique is to have formulated it in such a way (see definitions 28 and 38) that it can be implemented without requiring an *a priori* knowledge of data dependencies: it allows us to eliminate redundant traces on-the-fly as data (in)dependency is discovered by the constraint resolution procedure (as explained in Section 6.3.5) — in this sense, it may be viewed as a case of dynamic POR.

Narrowing the discussion a bit more, we now focus on the fact that our techniques are designed for the verification of equivalence properties. This requirement turns several seemingly trivial observations into subtle technical problems. For instance, ideas akin to compression are often applied without justification (*e.g.* in [SA06, TKL+12, MVB10]) because they may be obvious when one does reachability rather than equivalence checking. To understand this, it is important to distinguish between two very different ways of applying POR to equivalence checking (independently of the precise equivalence under consideration). The first approach is to reduce a system such that the reduced system and the original systems are equivalent. In

[1] Although this would be an interesting question, we do not expect that any improvement of compression would come out of it. Indeed, compression can be argued to be maximal in terms of eliminating redundant traces without analysing data: for any compressed trace there is a way to choose messages and modify tests to obtain a concrete execution which does not belong to the equivalence class of any other compressed trace.

the second approach, one only requires that two reduced systems are equivalent iff the original systems are equivalent. The first approach seems to be more common in the POR literature (where one finds, *e.g.* reductions that preserve LTL-satisfiability [BK08] or bisimilarity [HNW98]) though there are instances of the second approach (*e.g.* for Petri nets [God91]). In the present work, we follow the second approach: neither of our two reduction techniques preserves trace equivalence. This allows stronger reductions but requires extra care: one has to ensure that the independencies used in the reduction of one process are also meaningful for the other processes; in other words, reduction has to be symmetrical. We come back to these two different approaches later, when discussing specific POR techniques for security.

## 7.2 Security Applications

The idea of applying POR to the verification of security protocols dates back, at least, to the work of Clarke *et al.* [CJM00, CJM03]. In this work, the authors remark that traditional POR techniques cannot be directly applied to security mainly because "[they] must keep track of knowledge of various agents" and "[their] logic can refer to this knowledge in a meaningful way". This led them to define a notion of *semi-invisible actions* (output actions, that cannot be swapped after inputs but only before them) and design a reduction that prioritises outputs and performs them in a fixed order. Compared to our work, this reduction is much weaker (even weaker than compression only), only handles a finite set of messages, and only focuses on reachability properties checking.

In [EMMS14], the authors develop "state space reduction" techniques for the Maude-NRL Protocol Analyzer (Maude-NPA). This tool proceeds by backwards reachability analysis and treats at the same level the exploration of protocol executions and attacker's deductions. Several reductions techniques are specific to this setting, and most are unrelated to partial order reduction in general, and to our work in particular. We note that the lazy intruder techniques from [EMMS14] should be compared to what is done in constraint resolution procedures (*e.g.* the one used in Apte) rather than to our work. A simple POR technique used in Maude-NPA is based on the observation that inputs can be executed in priority in the backwards exploration, which corresponds to the fact that we can execute outputs first in forward explorations. We note again that this is only one aspect of the focused strategy, and that it is not trivial to lift this observation from reachability to trace equivalence. Finally, a "transition subsumption" technique is described for Maude-NPA. While highly non-trivial due to the technicalities of the model, this is essentially a tabling technique rather than a partial order reduction. Though it does yield a significant state space reduction (as shown in the experiments [EMMS14]) it falls short of exploiting independencies fully, and has a potentially high computational cost (which is not evaluated in the benchmarks of [EMMS14]).

In [FDW10], Fokkink *e*t al. model security protocols as labelled transition systems whose states contain the control points of different agents as well as previously outputted messages. They devise some POR technique for these transition systems, where output actions are prioritised and performed in a fixed order. In their work, the original and reduced systems are

trace equivalent *modulo* outputs (the same traces can be found after removing output actions). The justification for their reduction would fail in our setting, where we consider standard trace equivalence with observable outputs. More importantly, their requirement that a reduced system should be equivalent to the original one makes it impossible to swap input actions, and thus reductions such as the execution under focus of our compressed semantics cannot be used. The authors leave as future work the problem of combining their algorithm with symbolic executions, in order to be able to lift the restriction to a finite number of messages.

Cremers and Mauw proposed [CM05] a reduction technique for checking secrecy in security protocols. Their method allows to perform outputs eagerly, as in our compressed semantics. It also uses a form of *sleep set* technique to avoid redundant interleavings of input actions. In addition to being applicable only for reachability properties, the algorithm of [CM05] works under the assumption that for each input only finitely many input messages need to be considered. The authors identify as important future work the need to lift their method to the symbolic setting.

Earlier work by Mödersheim *et* al. has shown how to combine POR technique with symbolic semantics [MVB10] in the context of reachability properties for security protocols, which has led to high efficiency gains in the OFMC tool of the AVISPA platform [A$^+$05]. While their reduction is very limited, it brings some key insight on how POR may be combined with symbolic execution. For instance, their reduction imposes a dependency constraint on the interleavings of $\{\mathtt{in}(c,x).\mathtt{out}(c,m),\ \mathtt{in}(d,y).\mathtt{out}(d,m')\}$. Assuming that priority is given to the process working on channel $c$, this constraint enforces that any symbolic interleaving of the form $\mathtt{in}(d,M').\mathtt{out}(d,w').\mathtt{in}(c,M).\mathtt{out}(c,w)$ would only be explored for instances of $M$ that depend on $w'$. Our reduced semantics constrains patterns of arbitrary size (instead of just size 2 diamond patterns as above) by means of dependency constraints. Going back to Example 31 (in Subsection 5.4.3), their technique will only be able (at most) to exploit the dependencies depicted in blue plain arrows, and they will not consider the one represented by the dashed 2-arrow. Lastly, they will even do not detect all the patterns of this kind. Whereas we generate dependency constraints on the fly, they implement their technique by looking for such a pattern afterwards. Moreover, our POR technique has been designed to be sound and complete for trace equivalence checking as well.

## 7.3 Proof Theory

The reader familiar with focused proof systems [And92] will have recognised the strong similarities with our compressed semantics. The strategies are structured in the same way, around positive and negative phases. More deeply, the compressed semantics can actually be derived systematically from the focused proof system of linear logic, through an encoding of our processes into linear logic formulas (such that proof search corresponds to process executions). There are several such encodings in the literature, see for instance [Mil03, GP05, DCS12]. Intuitively, we would map input and output respectively to existential and universal quantifiers, which are respectively positive and negative in (linear) logic — the universal quantifier introduces a fresh handle (eigenvariable) that may be used for future instantiations of existentially quantified variables.

Then, parallel composition and its unit are mapped to the multiplicative disjunction and its unit, and replication to the "why not" modality, all of which are negative. Finally, the behaviour of each construct in the compressed semantics corresponds to the behaviour of its translation in focused proof systems. For instance, our treatment of replication where one must initiate focus when creating a new session is a direct translation of how the "why not" modality is treated in Andreoli's system [And92]. We do not provide here a fully worked-out encoding appropriate for our protocols. It is not trivial, notably due to the need to encode the attacker's knowledge, and internal reductions of protocols — both features require slight extensions of the usual linear logic framework. We have thus chosen to only take the correspondence with linear logic as an intuitive guide, and give a self-contained (and simple) proof of completeness for our compressed semantics (*i.e.* Lemma 4 in Section 5.3) by adapting the positive trunk argument of [MS07]. Note that the strong analogies with proof theory only hold for reachability results concerning compression, *i.e.* lemmas 3 and 4. It is a contribution in itself to observe that focusing (compression) makes sense beyond reachability, at the level of trace equivalence: Theorem 2 (stating that trace equivalence coincides with compressed trace equivalence for action-deterministic processes) has no analogue in the proof theoretical setting, where trace equivalence itself is meaningless.

We motivated reduction by observing that (in)dependencies between blocks of the compressed semantics should be exploited to eliminate redundant interleavings. This same observation has been done in the context of linear logic focusing, and lead to the idea of multi-focusing [CMS] where independent synthetic connectives (the analogue of our blocks) are executed simultaneously as much as possible. That work on multi-focusing is purely theoretical, and it is unclear how multi-focusing could be applied effectively in proof search. It would be interesting to consider whether the gradual construction of unique representatives in our reduced semantics could be extended to the richer setting of linear logic (where proof search branches, unlike process executions).

# Part C

# Verifying Privacy via Sufficient Conditions

**Significantly Improving Precision of Privacy Verification for the Unbounded Case**

## Table of Contents of the Part

# Introduction

In this part, we turn to methods and tools for verifying equivalence properties for an unbounded number of sessions. Remind that such methods and tools all rely on diff-equivalence rather than trace equivalence. As already mentioned in the introduction (Subsection 1.6.2) and discussed further with the example of the Feldhofer protocol (Subsection 3.3.1; Part A), diff-equivalence is not precise enough to be used for the verification of some privacy goals such as unlinkability.

We now narrow down the discussion on unlinkability and anonymity. In this part, we consider the well-established definitions of strong unlinkability and anonymity as defined in [ACRR10]. They have notably been used to establish privacy for various protocols either by hand or using ad hoc encodings (*e.g.* eHealth protocol [DJP12], mobile telephony [AMR$^+$12, AMRR14]). Note that we will provide a brief comparison with alternative definitions in Chapter 8 (Section 8.3) and notably conclude that other definitions (*e.g.* game-based definitions in the symbolic model) do not subsume our chosen definition. Intuitively, strong unlinkability and anonymity are expressed as the trace equivalence between an ideal scenario (where the privacy goal under consideration holds by construction) and a real scenario corresponding to a situation for which we are willing the prove that the privacy goal holds. Hence, for the case of unlinkability, the ideal scenario considers an unbounded number of users that can play at most one session each while the real scenario considers the same users but assume that each user can play an unbounded number of sessions. Unfortunately, as explained below, diff-equivalence is not suitable to establish such modelling of unlinkability.

**Closer look at the limitation of diff-equivalence.** Formally, if $(T(k, n_T) \mid R(k, n_R))$ is a process modelling one session of a 2-party protocol (*e.g.* think of $T$ as an RFID tag and $R$ as a reader) where $k$ is a long-term key (the same for all sessions but distinct for different users) and $n_T, n_R$ are fresh nonces (distinct for each session and each user), then the real scenario would be $P_{\text{real}} = !\nu k.\, !\nu n_T.\nu n_R.(T(k, n_T) \mid R(k, n_R))$ while the ideal scenario would be $P_{\text{ideal}} = !\nu k.\nu n_T.\nu n_R.(T(k, n_T) \mid R(k, n_R))$. We end up with the following trace equivalence to prove: $P_{\text{real}} \approx P_{\text{ideal}}$. If one is willing to prove this property automatically, he is left with tools such as ProVerif, Tamarin or Maude−NPA that can only deal with diff-equivalence. Thus, the first step would be to form a bi-process encoding the above equivalence. Based on the fact that two

successive replications ! collapse, the most appropriate bi-process one can come up with is the following:

$$!\nu k_l.\ !\nu k_r.\nu n_T.\nu n_R.(T(\mathsf{choice}[k_l, k_r], n_T) \mid R(\mathsf{choice}[k_l, k_r], n_R)).$$

It is now possible to understand why diff-equivalence is not precise enough. Since exactly the same rules will be applied on the left and on the right, two instances of the role $T$ having the same key on the left cannot have the same key on the right: *e.g.* $t_1 = T(\mathsf{choice}[k_l^1, k_r^1], n_T)$ and $t_2 = T(\mathsf{choice}[k_l^1, k_r^2], n'_T)$. The same for $R$: *e.g.* $r_1 = R(\mathsf{choice}[k_l^1, k_r^1], n_R)$ and $r_2 = R(\mathsf{choice}[k_l^1, k_r^2], n'_R)$. Note that, on the left, the first instance $t_1$ of $T$ can successfully execute a session of the protocol with the second instance $r_2$ of $R$. However, when looking at the right side, those two processes $t_1$ and $r_2$ are most likely unable to have such an "honest" interaction because their long-term keys (*i.e.* respectively $k_r^1$ and $k_r^2$) do not match. Therefore, the above bi-process is most likely not diff-equivalent and we did not even examine the processes $T$ and $R$ to conclude so. Obviously, the above trace equivalence does not systematically fail to hold since one can come up with the pair $(t_1, r_1)$ on the right to mimic a successful interaction of $(t_1, r_2)$ on the left. To sum up, the crux of the problem is that diff-equivalence considers an over-approximation of the Dolev-Yao attacker by giving him the internal structure of the two processes to be verified, the attacker is thus able to observe from which replication a given instance of a role originates. The latter almost systematically leads to *false attacks* and makes diff-equivalence of no use when it comes to verify unlinkability. Remark that similar issues arise with other privacy goals such as vote-privacy (only partially addressed in [DRS08, BS16]).

In practice, this means that many security protocols cannot be verified. We already discussed the case of the Feldhofer protocol in Subsection 3.3.1 (in Part A). But as explained above, this is not an isolated problem: unlinkability of the BAC protocol (used in e-passport) cannot be automatically established either (despite recent improvements on diff-equivalence checking [CB13]) and similarly for most of the numerous real-world case studies we will present in Chapter 10.

**Contributions.** We believe that looking at trace equivalence of any protocol is a too general problem and that much progress can be expected when one focuses on a few privacy goals and a class of protocols only (yet large and generic enough). We follow this different approach. We aim at proposing sufficient conditions that can be automatically checked, and that imply unlinkability and anonymity for a large class of security protocols. The success of our solution will be measured by confronting it to many real-world case studies.

More precisely, we identify a large class of 2-party protocols (simple else branches, arbitrary cryptographic primitives) and we devise two conditions called *frame opacity* and *well-authentication* that imply unlinkability and anonymity for an unbounded number of sessions. We show how these two conditions can be automatically checked using *e.g.* the ProVerif tool, and we provide tool support for that. Using our tool UKano (built on top of ProVerif), we have automatically analysed several protocols, among them the Basic Access Control (BAC) protocol as well as the Password Authenticated Connection Establishment (PACE) protocol that are both used in e-passports. We notably establish the first proof of unlinkability for ABCDH [AH13] and

for the BAC protocol followed by the Passive Authentication (PA) and Active Authentication (AA) protocols. We also report on an attack that we found on the PACE protocol, and another one that we found on the LAK [LAK06] protocol whereas it is claimed untraceable in [VDR08]. It happens that our conditions are rather tight: we provide an attack every time one of them is not satisfied.

We believe that the overall methodology and proof method could be used for other classes of protocols and other privacy goals. For instance, reusing the core ideas of the present methodology, we devised a new method [CH17] to automatically verify vote-privacy for a large class of e-voting protocols.

**Our sufficient conditions.** We now give an intuitive overview of our two sufficient conditions. In order to do this, assume that we want to design a mutual authentication protocol between a tag $T$ and a reader $R$ based on symmetric encryption, and we want this protocol to be unlinkable. We assume that $k$ is a symmetric key shared between $T$ and $R$.

*Frame opacity.* A first attempt to design such a protocol is presented using Alice & Bob notation as follows ($n_R$ is a fresh nonce):

$$\begin{array}{ll} 1. & R \to T : \quad n_R \\ 2. & T \to R : \quad \{n_R\}_k \end{array}$$

This first attempt based on a challenge-response scheme is actually linkable. Indeed, an active attacker who systematically intercepts the nonce $n_R$ and replaces it by a constant will be able to infer whether the same tag has been used in different sessions or not by comparing the answers he receives. Here, the tag is linkable because, for a certain behaviour (possibly malicious) of the attacker, some relations between messages leak information about the agents that are involved in the execution. Our first condition, namely *frame opacity*, actually checks that all outputted messages have only relations that only depend on what is already observable. Such relations can therefore not be exploited by the attacker to learn anything new about the involved agents.

*Well-authentication.* Our second attempt takes the previous attack into account and randomises the tag's response and should achieve mutual authentication by requiring that the reader must answer to the challenge $n_T$. This protocol can be as follows:

$$\begin{array}{ll} 1. & R \to T : \quad n_R \\ 2. & T \to R : \quad \{n_R, n_T\}_k \\ 3. & R \to T : \quad \{n_T\}_k \end{array}$$

Here, Alice & Bob notation shows its limit. It does not specify how the reader and the tag are supposed to check that the messages they received are of the expected form, and how they should react when the messages are not well formed. This has to be precisely defined, since unlinkability depends on it. For instance, assume the tag does not check that the message he receives at step 3 contains $n_T$, and aborts the session if the received message in not encrypted with its own $k$. In such an implementation, an active attacker can eavesdrop a message $\{n_T\}_k$ sent by $R$ to a tag $T$, and try to inject this message at the third step of another session played

by $T'$. The tag $T'$ will react by either aborting or by continuing the execution of this protocol. Depending on the reaction of the tag, the attacker will be able to infer if $T$ and $T'$ are the same tag or not.

In this example, the attacker adopts a malicious behaviour that is not detected immediately by the tag who keeps executing the protocol. The fact that the tag passes successfully a conditional reveals crucial information about the agents that are involved in the execution. Our second condition, namely *well-authentication*, basically requires that when an execution deviates from the honest one, the agents that are involved cannot successfully pass a conditional, thus avoiding the leak of the binary information success/failure.

*Main theorem.* In a nutshell, well-authentication avoids attacks based on control-flow leaks that may be observed through data or nature of actions while frame opacity avoids attacks based on data leaks taking the form of relations between outputs. Our main theorem states that these two conditions, frame opacity and well-authentication, are actually sufficient to ensure both unlinkability and anonymity. Interestingly, at the core of its proof, we must show that any execution can be transformed into another one indistinguishable from the former in which all sessions are played by distinct agents. Well-authentication is used to prove that the latter has the same control-flow as the first one. Frame opacity then ensures that the latter is indistinguishable from the former. This theorem is of interest as our two conditions are fundamentally simpler than the targeted properties: frame opacity can be expressed and established relying on diff-equivalence (without the aforementioned precision issue) and well-authentication is only a conjunction of reachability properties. In fact, they are both in the scope of existing automatic verification tools like ProVerif.

**Related work.** The precision issue of diff-equivalence is well-known (acknowledged *e.g.* in [DRS08, CB13, BS16]). So far, the main approach that has been developed to solve this issue consists in modifying the notion of diff-equivalence to get closer to trace equivalence. For instance, targeting process algebra with phases (often used for modelling e-voting protocols), the swapping technique introduced in [DRS08] allows to relax constraints imposed by diff-equivalence in specific situations (this technique has then been given formal foundations in [BS16]). Besides, the limitation of the diff-equivalence w.r.t. conditional evaluations has been partially addressed in [CB13] by pushing away the evaluation of some conditionals into terms. Nevertheless, the problem remains in general and the limitation described above is not addressed by those works (incidentally, it is specifically acknowledged for the case of the BAC protocol in [CB13]).

We have chosen to follow a novel approach in the same spirit as the one presented in [BCDH10]. Nevertheless, [BCDH10] only considers a very restricted class of protocols (single-step protocols that only use hash functions), while we target more complex protocols.

**Outline** In Chapter 8, we present the class of protocols and the formal definitions of unlinkability and anonymity we woud like to verify. Our two conditions (frame opacity and well-authentication) and our main theorem are presented in Chapter 9. We also prove their soundness;

*i.e.* the combination of our two conditions always imply unlinkability and anonymity. In Chapter 10, we discuss how to mechanise the verification of our conditions via systematic encodings, we describe our tool UKano mechanising those encodings and use it to analyse an extensive list of real-world case studies. We conclude with future work in Chapter 11.

Chapter 8

# Model & Problem

In this chapter, we first define the instantiation of the semantics (Section 8.1) we shall work with in the present part. We then define (Section 8.2) the class of protocols we are dealing with. Finally, we define (Section 8.3) the security goals *unlinkability* and *anonymity* we want to verify and compare our definitions to others in the literature.

## 8.1 Instantiation of the Model

In this part of the thesis, we eventually leverage the tool ProVerif in Chapter 10 and shall use the corresponding instantiation of the semantics. However, the theoretical part of our approach (*i.e.* the theorem stating that our conditions imply privacy) can be formulated in a more generic framework that we define now.

**Term Algebra.** We assume any signature $\Sigma$, any equational theory $=_{\mathsf{E}}$, and any computation relation $\Downarrow$. In other words, our approach is completely generic in the term algebra.

**Example 47.** *We may for instance consider the signature*

$$\Sigma = \{\mathsf{senc},\ \mathsf{sdec},\ \langle\,\rangle,\ \mathsf{proj}_1,\ \mathsf{proj}_2,\ \oplus,\ 0,\ \mathsf{eq},\ \mathsf{neq},\ \mathsf{ok}\}.$$

*of Example 1 (Chapter 2) equipped with the equational theory of Example 2 (Chapter 2) and the computation relation induced by the rewriting system given in Example 10 (Chapter 2).*

**Syntax & Semantics.** We fix in this part $\mathscr{R} = \mathscr{R}_{\text{par}}$ and thus obtain an internal reduction $\rightsquigarrow_{\mathscr{R}}$ breaking parallel compositions greedily. We already proved in Section 4.1 that this choice does not impact the induced notion of trace equivalence.

**Example 48.** *One of our running examples in this part will be the Feldhofer protocol (see examples 6 and 9 in Chapter 2). We recall here how we modelled it and describe one of its executions. We consider the term algebra introduced in Example 47. The protocol is modelled*

*by the parallel composition of the processes $P_I$ and $P_R$, corresponding respectively to the roles $I$ and $R$.*

$$P_{\mathsf{Fh}} \stackrel{\text{def}}{=} \nu k.\ (\nu n_I.P_I \mid \nu n_R.P_R)$$

*where $P_I$ and $P_R$ are defined as follows, with $u = \mathsf{sdec}(x_1, k)$:*

$$\begin{aligned} P_I \stackrel{\text{def}}{=}\ & \mathtt{out}(c_I, n_I).\\ & \mathtt{in}(c_I, x_1).\\ & \mathsf{let}\ x_2, x_3 = \mathsf{eq}(n_I, \mathsf{proj}_1(u)), \mathsf{proj}_2(u)\ \mathtt{in}\\ & \mathtt{out}(c_I, \mathsf{senc}(\langle x_3, n_I\rangle, k)) \end{aligned}$$

$$\begin{aligned} P_R \stackrel{\text{def}}{=}\ & \mathtt{in}(c_R, y_1).\\ & \mathtt{out}(c_R, \mathsf{senc}(\langle y_1, n_R\rangle, k)).\\ & \mathtt{in}(c_R, y_2).\\ & \mathsf{let}\ y_3 = \mathsf{eq}(y_2, \mathsf{senc}(\langle n_R, y_1\rangle, k))\ \mathtt{in}\ 0 \end{aligned}$$

*We have that $(\{P_{\mathsf{Fh}}\}; \emptyset) \xrightarrow{\mathsf{tr}} (\emptyset; \Phi_0)$ where $\mathsf{tr}$ and $\Phi_0$ are as follows, for fresh names $k', n'_I, n'_R \in \mathcal{N}$:*

$$\begin{aligned} \mathsf{tr} &= \tau_\nu.\tau_\nu.\tau_\nu.\mathtt{out}(c_I, w_1).\mathtt{in}(c_R, w_1).\mathtt{out}(c_R, w_2).\mathtt{in}(c_I, w_2).\tau_{\mathsf{then}}.\mathtt{out}(c_I, w_3).\mathtt{in}(c_R, w_3).\tau_{\mathsf{then}}\\ \Phi_0 &= \{w_1 \mapsto n'_I,\ w_2 \mapsto \mathsf{senc}(\langle n'_I, n'_R\rangle, k'),\ w_3 \mapsto \mathsf{senc}(\langle n'_R, n'_I\rangle, k')\}. \end{aligned}$$

*This execution corresponds to a normal execution of one session of the protocol.*

*As already discussed in Subsection 3.3.1 (in Chapter 3), unlinkability could be expressed as the trace equivalence between $K = (!P_{\mathsf{Fh}}; \emptyset)$ (the ideal version) and $K' = (!\nu k.(!\nu n_I.P_I \mid !\nu n_R.P_R); \emptyset)$ (the real version). As said in the introduction, this equivalence is non-trivial, and cannot be established using existing verification tools such as* ProVerif *or* Tamarin*. The technique developed in this part will notably allow one to establish it automatically.*

In the previous example, the process $!\nu n_I.P_I$ models an initiator having a specific long-term key $k$ ready to execute an unbounded number of sessions of the protocol *concurrently*. Depending on the practical scenario we shall model, sessions of such an agent may not be run *concurrently* but only *sequentially* (*i.e.* one after the other). This will be the case if the initiator is played by a device like RFID tags which can execute at most one session at a time. In such cases, modellings based on replication ! might introduce unwanted behaviours possibly leading to false attacks. We thus define next the constructs $P; Q$ and $¡P$ as syntactic sugar allowing to model *sequential executions* and "sequential replication" that we rather call *repetition*.

**Notation 3.** *We introduce two new constructs as syntactic sugar:*

- *For two processes $P$ and $Q$, we note $P; Q$ the process $P\{0 \mapsto Q\}$ (*i.e. *the process one can obtain from $P$ by replacing all null processes $0$ by the process $Q$). We call $P; Q$ the* sequence of $P$ and $Q$.
- *For a process $P$, we note $¡P$ the process* $\mathrm{rec}X.(P; X)$. *We call such a process the* repetition *of $P$.*

According to the semantics defined in Chapter 2, we may infer the following intuitions. The process $(P;Q)$ behaves like $P$ at first, and after the complete execution of $P$ it behaves like $Q$. The process $¡P$ executes $P$ an arbitrary number of times in sequence, intuitively corresponding to $(P;P;P;\ldots)$. That is the reason why we call *repetition* the construct $¡P$. Such constructions are known to be problematic in process calculi. Our goal here is however quite modest: as it is visible in our operational semantics, our sequential composition is only meaningful for restricted processes (*e.g.* $(!P);Q$ is never able to reach $Q$).

**Example 49.** *For instance,* $¡\nu n_I.P_I$ *models an initiator having a specific long-term key* $k$ *ready to execute an unbounded number of sessions of the protocol* sequentially*.*

## 8.2 A Generic Class of Two-party Protocols

We aim to propose sufficient conditions to ensure unlinkability and anonymity for a generic class of two-party protocols. In this section, we define formally the class of protocols we are interested in. We consider two-party protocols that are therefore made of two roles called the initiator and responder role respectively. We assume a set $\mathcal{L}$ of labels that will be used to name output actions in these roles, allowing us to identify outputs that are performed by a same syntactic output action. These labels have no effect on the semantics.

**Definition 49.** *An* initiator role *is a ground process obtained using the following grammar:*

$$P_I := 0 \;\mid\; \ell : \mathtt{out}(c,u).P_R$$

*where* $c \in \mathcal{C}$*,* $u \in \mathcal{T}(\Sigma_c, \mathcal{N} \cup \mathcal{X})$*,* $\ell \in \mathcal{L}$*, and* $P_R$ *is obtained from the grammar of* responder roles*:*

$$\begin{array}{rcl} P_R & := & 0 \\ & \mid & \mathtt{in}(c,y).\mathsf{let}\ \overline{x} = \overline{t}\ \mathtt{in}\ P_I\ \mathsf{else}\ 0 \\ & \mid & \mathtt{in}(c,y).\mathsf{let}\ \overline{x} = \overline{t}\ \mathtt{in}\ P_I\ \mathsf{else}\ \ell : \mathtt{out}(c',u') \end{array}$$

*where* $c, c' \in \mathcal{C}$*,* $y \in \mathcal{X}$*,* $\overline{x}$ *(resp.* $\overline{t}$*) is a (possibly empty) sequence of variables in* $\mathcal{X}$ *(resp. terms in* $\mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X})$*),* $u' \in \mathcal{T}(\Sigma_c, \mathcal{N} \cup \mathcal{X})$*, and* $\ell \in \mathcal{L}$*.*

Intuitively, a role describes the actions performed by an agent. A responder role consists of waiting for an input and, depending on the outcome of a number of tests, the process will continue by sending a message and possibly waiting for another input, or stop possibly outputting an error message. An initiator behaves similarly but begins with an output. The grammar forces to add a conditional after each input. This is not a real restriction as it is always possible to add trivial conditionals with empty $\overline{x}$ and $\overline{t}$. Finally, note that terms in outputs are necessarily constructor terms (*i.e.* in $\mathcal{T}(\Sigma_c, \mathcal{N} \cup \mathcal{X})$). Therefore, outputs are never blocked.

**Example 50.** *Continuing our running example,* $P_I$ *(resp.* $P_R$*) as defined in Example 48 is an initiator (resp. responder) role, up to the addition of trivial conditionals and distinct labels* $\ell_1$*,* $\ell_2$*, and* $\ell_3$ *to decorate output actions.*

Then, a protocol notably consists of an initiator role and a responder role that can interact together producing an *honest trace* as defined next.

**Definition 50.** *A trace* $\mathsf{tr}$ *is* honest *for a frame* $\Phi$ *if* $\tau_{\mathsf{else}} \notin \mathsf{tr}$ *and* $\mathsf{obs}(\mathsf{tr})$ *is of the form* $\mathtt{out}(\_, w_0).\mathtt{in}(\_, R_0).\mathtt{out}(\_, w_1).\mathtt{in}(\_, R_1)\ldots$ *for arbitrary channel names, and such that* $R_i\Phi \Downarrow{=_{\mathsf{E}}} w_i\Phi \Downarrow$ *for any action* $\mathtt{in}(\_, R_i)$ *occurring in* $\mathsf{tr}$.

An honest trace is a trace in which the attacker does not really interfere, and that allows the execution to progress without going into an else branch, which would intuitively correspond to a way to abort the protocol.

In addition to the pair of initiator and responder roles, more information is needed in order to meaningfully define a protocol. Among the names that occur in these two roles, we need to distinguish those that correspond to long-term data (called *identity names*) and those which shall be freshly generated at each session (called *session names*). We will require that any free name of roles (free names of a process $P$ are denoted by $fn(P)$) must be either a session or an identity name.

We also need to know whether sessions (with the same identity parameters) can be executed *concurrently* or only *sequentially*. For instance, let us assume that the Feldhofer protocol is used in an access control scenario where all tags that are distributed to users have pairwise distinct identities. Assuming that tags cannot be cloned, it is probably more realistic to consider that a tag can be involved in at most one session at a particular time, *i.e.* a tag may run different sessions but only in sequence. Another concrete example where sessions are executed sequentially is the case of the e-passport application where a same passport cannot be involved in two different sessions concurrently.

**Definition 51.** *A protocol* $\Pi$ *is a tuple* $(\overline{k}, \overline{n}_I, \overline{n}_R, \dagger_I, \dagger_R, \mathcal{I}, \mathcal{R})$ *where* $\overline{k}$, $\overline{n}_I$, $\overline{n}_R$ *are three disjoint sets of names,* $\mathcal{I}$ *(resp.* $\mathcal{R}$*) is an initiator (resp. responder) role such that* $fn(\mathcal{I}) \subseteq \overline{k} \sqcup \overline{n}_I$, $fn(\mathcal{R}) \subseteq \overline{k} \sqcup \overline{n}_R$, *and* $\dagger_I, \dagger_R \in \{!, \mathsf{i}\}$. *Labels of* $\mathcal{I}$ *and* $\mathcal{R}$ *must be pairwise distinct. Names* $\overline{k}$ *(resp.* $\overline{n}_I \sqcup \overline{n}_R$*) are called* identity names *(resp.* session names*).*

*Given a protocol* $\Pi$*, we define* $P_\Pi \stackrel{\text{def}}{=} \nu\overline{k}.(\nu\overline{n}_I.\mathcal{I} \mid \nu\overline{n}_R.\mathcal{R})$ *and assume that* $P_\Pi \xrightarrow{\mathsf{tr}_h} (\emptyset; \Phi_h)$ *for some frame* $\Phi_h$ *and some trace* $\mathsf{tr}_h$ *that is honest for* $\Phi_h$.

The component $\dagger_I$ (resp. $\dagger_R$) describes whether the sessions of the role $\mathcal{I}$ (resp. $\mathcal{R}$) can be executed concurrently or only sequentially. The process $P_\Pi$ models a single session of the protocol involving the two roles. Moreover, we require that this process can produce an honest trace.

Given a protocol $\Pi$, we also associate another process $\mathcal{M}_\Pi$ that represents the situation where the protocol can be executed by an arbitrary number of identities, with the possibility of executing an arbitrary number of sessions for a given identity. The formal definition differs slightly depending on whether identity names occur in both roles or only in the role $\mathcal{I}$ (resp. $\mathcal{R}$).

**Definition 52.** *Given a protocol* $\Pi = (\overline{k}, \overline{n}_I, \overline{n}_R, \dagger_I, \dagger_R, \mathcal{I}, \mathcal{R})$*, the process* $\mathcal{M}_\Pi$ *is defined as follows:*

- *If* $\overline{k} \cap fn(\mathcal{I}) \neq \emptyset$ *and* $\overline{k} \cap fn(\mathcal{R}) \neq \emptyset$, *then* $\mathcal{M}_\Pi \stackrel{\text{def}}{=} !\, \nu\overline{k}.(\dagger_I\, \nu\overline{n}_I.\mathcal{I} \mid \dagger_R\, \nu\overline{n}_R.\mathcal{R})$;

- *If* $\overline{k} \cap fn(\mathcal{I}) = \emptyset$ *and* $\overline{k} \cap fn(\mathcal{R}) \neq \emptyset$, *then* $\mathcal{M}_\Pi \stackrel{\text{def}}{=} \dagger_I\, \nu\overline{n}_I.\mathcal{I} \;\mid !\, \nu\overline{k}.\, \dagger_R\, \nu\overline{n}_R.\mathcal{R})$

For the sake of simplicity, in case identity names only occur in one role, we assume that this role is the responder role. The omitted case where identity names occur only in the initiator role is very much similar. In fact, swapping the initiator and responder roles can also be formally achieved by adding an exchange of a fresh nonce at the beginning of the protocol under consideration. Note that the case where both $\overline{k} \cap fn(\mathcal{I})$ and $\overline{k} \cap fn(\mathcal{R})$ are empty means that no identity names are involved and therefore there is no issue regarding privacy.

**Example 51.** *Let* $\Pi_{\mathsf{Fh}} = (k, n_I, n_R, !, !, P_I, P_R)$ *with* $P_I$ *and* $P_R$ *as defined in Example 48. We have seen that* $P_I$ *is an initiator role whereas* $P_R$ *is a responder role. Let* $P_{\mathsf{Fh}} = \nu k.(\nu n_I.P_I \mid \nu n_R.P_R)$. *Let* $\mathsf{tr}_h = \mathsf{tr}$, *and* $\Phi_h = \Phi_0$ *as defined in Example 48. They satisfy the requirements stated in Definition 51, and therefore* $\Pi_{\mathsf{Fh}}$ *is a protocol according to our definition. For this protocol, the identity name* $k$ *occurs both in the role* $P_I$ *and* $P_R$, *and therefore we have that* $\mathcal{M}_{\Pi_{\mathsf{Fh}}} = !\, \nu k.(!\, \nu n_I.P_I \;\mid !\, \nu n_R.P_R)$.

For the purpose of illustrating our method and the use of the repetition operator, we will consider a variant of the Feldhofer protocol, described next.

**Example 52.** *We consider a variant of the RFID protocol presented in Example 48. This protocol can be presented using Alice & Bob notation as follows:*

$$\begin{array}{lll} 1. & I \to R: & n_I \\ 2. & R \to I: & n_R, \{n_I\}_k^{r_R} \\ 3. & I \to R: & \{n_R\}_k^{r_I} \end{array}$$

*The protocol is between an initiator* $I$ *(the reader) and a responder* $R$ *(the tag) that share a symmetric key* $k$. *We consider here a randomised symmetric encryption scheme, and the main difference with the original Feldhofer protocol as presented in Example 48 is the fact that encryption is performed on messages that are reduced to a single nonce. We will see that this may lead to a linkability attack in case concurrent sessions are possible, but this variant is actually safe (w.r.t. unlinkability) otherwise.*

*We consider the computation relation induced by the empty set of equations, and the following rewriting system:*

$$\mathsf{rdec}(\mathsf{renc}(x, y, z), y) \to x \qquad \mathsf{eq}(x, x) \to \mathsf{ok} \qquad \mathsf{proj}_i(\langle x_1, x_2 \rangle) \to x_i \quad \textit{for } i \in \{1, 2\}.$$

*The processes* $P'_I$ *and* $P'_R$ *modelling respectively the initiator and the responder roles are*

*defined as follows:*

$$\begin{aligned}
P'_I \quad &\stackrel{\text{def}}{=} \quad \mathtt{out}(c_I, n_I).\\
&\qquad\quad \mathtt{in}(c_I, x_1).\\
&\qquad\quad \mathsf{let}\ x_2, x_3 = \mathsf{eq}(n_I, \mathsf{rdec}(\mathsf{proj}_2(x_1), k)), \mathsf{proj}_1(x_1)\ \mathtt{in}\\
&\qquad\quad \mathtt{out}(c_I, \mathsf{renc}(x_3, k, r_I))\\
P'_R \quad &\stackrel{\text{def}}{=} \quad \mathtt{in}(c_R, y_1).\\
&\qquad\quad \mathtt{out}(c_R, \langle n_R, \mathsf{renc}(y_1, k, r_R)\rangle).\\
&\qquad\quad \mathtt{in}(c_R, y_2).\\
&\qquad\quad \mathsf{let}\ y_3 = \mathsf{eq}(n_R, \mathsf{rdec}(y_2, k))\ \mathtt{in}\ 0
\end{aligned}$$

*The tuple* $\Pi^{\mathsf{i}}_{\mathsf{Fh}'} \stackrel{\text{def}}{=} (k, \{n_I, r_I\}, \{n_R, r_R\}, \mathsf{i}, \mathsf{i}, P'_I, P'_R)$ *is a protocol according to Definition 51. For this protocol, we have again that* $k$ *occurs both in the roles* $P'_R$ *and* $P'_I$*, and therefore we have that:*

$$\mathcal{M}_{\Pi^{\mathsf{i}}_{\mathsf{Fh}'}} = !\,\nu k.(\mathsf{i}\ \nu(n_I, r_I).P'_I \mid \mathsf{i}\ \nu(n_R, r_R).P'_R).$$

As a last example, we will consider one for which identity names only occur in one role. This example can be seen as a simplified version of the DAA sign protocol that will be detailed in Chapter 10.

**Example 53.** *We consider a simplified version of the protocol DAA sign (adapted from [SRC15]). Note that a comprehensive analysis of the protocol DAA sign (as well as the protocol DAA join) will be conducted in Chapter 10. Before describing the protocol itself, we introduce the term algebra that will allow us to model signature, and zero knowledge proofs used in that protocol. For this, we consider:*

- $\Sigma_c = \{\mathsf{sign}, \mathsf{zk}, \mathsf{pk}, \langle\ \rangle, \mathsf{tuple}, \mathsf{ok}, \mathsf{sk}_{\mathsf{I}}, \mathsf{error}\}$*, and*
- $\Sigma_d = \{\mathsf{checksign}, \mathsf{check}_{\mathsf{zk}}, \mathsf{public}_{\mathsf{zk}}, \mathsf{proj}_1, \mathsf{proj}_2, \mathsf{proj}^4_1, \mathsf{proj}^4_2, \mathsf{proj}^4_3, \mathsf{proj}^4_4\}$.

*We consider the computation relation induced by the empty set of equations, and the following rewriting system:*

$$\begin{aligned}
\mathsf{checksign}(\mathsf{sign}(x, y), \mathsf{pk}(y)) &\to x\\
\mathsf{check}_{\mathsf{zk}}(\mathsf{zk}(\mathsf{sign}(\langle x_k, x_{id}\rangle, z_{\mathsf{sk}}), x_k, \mathsf{tuple}(y_1, y_2, y_3, \mathsf{pk}(z_{\mathsf{sk}})))) &\to \mathsf{ok}\\
\mathsf{public}_{\mathsf{zk}}(\mathsf{zk}(x, y, z)) &\to z\\
\mathsf{proj}_i(\langle y_1, y_2\rangle) &\to y_i \quad i \in \{1, 2\}\\
\mathsf{proj}^4_i(\mathsf{tuple}(y_1, y_2, y_3, y_4)) &\to y_i \quad i \in \{1, 2, 3, 4\}
\end{aligned}$$

*The protocol is between a client* $C$ *(the responder) and a verifier* $V$ *(the initiator of the protocol). The client is willing to sign a message* $m$ *using a credential issued by some issuer and then he has to convince* $V$ *that the latter signature is genuine. The client* $C$ *has a long-term secret key* $k_C$*, an identity* $id_C$*, and some credential* $cred_C = \mathsf{sign}(\langle k_C, id_C\rangle, \mathsf{sk}_{\mathsf{I}})$ *issued by some issuer* $I$ *having* $\mathsf{sk}_{\mathsf{I}}$ *as a long-term signature key. Such a credential would be typically obtained*

*once for all through a protocol similar to DAA join. We give below an Alice & Bob description of the protocol:*

1. $V \to C: \quad n_V$
2. $C \to V: \quad \mathsf{zk}(cred_C, k_C, \mathsf{tuple}(n_V, n_C, m, \mathsf{pk}(\mathsf{sk_I})))$

*The verifier starts by challenging the client with a fresh nonce, the latter then sends a complex zero-knowledge proof bound to this challenge proving that he knows a credential from the expected issuer bound to the secret $k_C$ he knows. Before accepting this zero-knowledge proof, the verifier $V$ (i) checks the validity of the zero-knowledge proof using the* $\mathsf{check_{zk}}$ *operator, and (ii) verifies that this proof is bound to the challenge $n_V$ and to the public key of the issuer $I$ using the* $\mathsf{public_{zk}}$ *operator.*

*The processes $P_C$ and $P_V$ modelling respectively the client and the verifier role are defined as follows:*

$$\begin{aligned} P_V \stackrel{\text{def}}{=}\ & \mathtt{out}(c_V, n_V).\\ & \mathtt{in}(c_V, x_1).\\ & \mathsf{let}\ x_2, x_3, x_4 =\\ & \mathsf{eq}(\mathsf{check_{zk}}(x_1), \mathsf{ok}), \mathsf{eq}(\mathsf{proj}_1^4(\mathsf{public_{zk}}(x_1)), n_V), \mathsf{eq}(\mathsf{proj}_4^4(\mathsf{public_{zk}}(x_1)), \mathsf{pk}(\mathsf{sk_I}))\ \mathtt{in}\ 0\\ & \mathsf{else}\ \mathtt{out}(c_V, \mathsf{error}) \end{aligned}$$

$$\begin{aligned} P_C \stackrel{\text{def}}{=}\ & \mathtt{in}(c_R, y_1).\\ & \mathtt{out}(c_R, \mathsf{zk}(\mathsf{sign}(\langle k_C, id_C\rangle, \mathsf{sk_I}), k_C, \mathsf{tuple}(y_1, n_C, m, \mathsf{pk}(\mathsf{sk_I})))). \end{aligned}$$

*This protocol falls in our class: the two parties being the verifier $V$ and the client $C$. The tuple $\Pi_{\mathsf{DAA}} \stackrel{\text{def}}{=} (\{k_C, id_C\}, \{n_V\}, \{n_C, m\}, !, !, P_V, P_C)$ is a protocol according to our Definition 51. We have that $k_C$ or $id_C$ only occur in $P_C$, and therefore following Definition 52, we have that:*

$$\mathcal{M}_{\Pi_{\mathsf{DAA}}} = (!\ \nu n_V.P_V)\ \mid\ (!\ \nu(k_C, id_C).\ !\ \nu(n_C, m).P_C)$$

*This models infinitely many different clients that may take part to infinitely many sessions of the protocol with any verifier that executes always the same role (he has no proper identity). We consider here a scenario where sessions can be executed concurrently.*

**Discussion: shared and non-shared protocols.** As shown in Definition 52, we distinguish two cases whether (i) both roles use identity names (*i.e.* $fn(\mathcal{I}) \cap \overline{k} \neq \emptyset$ and $fn(\mathcal{R}) \cap \overline{k} \neq \emptyset$) or (ii) only one role uses identity names (*i.e.* by symmetry this is when $fn(\mathcal{I}) \cap \overline{k} = \emptyset$ and $fn(\mathcal{R}) \cap \overline{k} \neq \emptyset$). This corresponds to the cases where (i) we should consider arbitrary number of users for each role or (ii) we should consider arbitrary number of users having the role $\mathcal{R}$ but it suffices to consider the single identity of $\mathcal{I}$. In addition to this distinction, note that two very different kinds of protocols lie in the class (i):

(i-a) First case is when $fn(\mathcal{I}) \cap fn(\mathcal{R}) \neq \emptyset$ and we will refer to this case as the *shared case*. Indeed, roles $\mathcal{I}$ and $\mathcal{R}$ from protocols satisfying the latter initially share some knowledge before the beginning of the protocol (*i.e.* they share the knowledge of the names in $fn(\mathcal{I}) \cap fn(\mathcal{R})$). In practice, this shared knowledge may have been established in various ways such as by using

prior protocols, using another communication channel (*e.g.* optical scan of a password as it is done with e-passports, use of PIN codes) or by retrieving the identity from a database that matches the first received message (as it is often done with RFID protocols). For such protocols, it is expected that an initiator user and a responder user can communicate successfully producing an honest execution *only if* they have the same identity (*i.e.* they share the same names $\overline{k}$).

(i-b) Second case is when $fn(\mathcal{I}) \cap fn(\mathcal{R}) = \emptyset$ and we will refer to this case as the *non-shared case* as roles do not share any prior knowledge. For such cases, it is expected that an initiator user and a responder user can communicate successfully producing an honest execution whatever their identities.

Unlinkability and anonymity will be uniformly expressed for the cases (i-a) and (i-b) but our sufficient conditions will slightly differ depending on the considered case.

## 8.3 Security Goals

This section is dedicated to the definition of the security properties we are willing to verify on protocols: unlinkability and anonymity.

### 8.3.1 Unlinkability

Remind that according to the ISO/IEC standard 15408 [ISO09], unlinkability aims at ensuring that a user may make multiple uses of a service or a resource without others being able to link these uses together. In terms of our modelling, a protocol preserves unlinkability if any two sessions of a same role look to an outsider as if they have been executed with different identity names. In other words, an ideal version of the protocol with respect to unlinkability, allows the roles $\mathcal{I}$ and $\mathcal{R}$ to be executed at most once for each identity names. An outside observer should then not be able to tell the difference between the original protocol and the ideal version of this protocol.

In order to precisely define this notion, we have to formally define this ideal version of a protocol $\Pi$. This ideal version, denoted $\mathcal{S}_\Pi$, represents an arbitrary number of agents that can at most execute one session each. Such a process is obtained from $\mathcal{M}_\Pi$ by simply removing the symbols ! and ¡ that are in the scope of identity names. Indeed, those constructs enable each identity to execute an arbitrary number of sessions (respectively concurrently and sequentially). Formally, depending on whether identity names occur in both roles, or only in the initiator role, this leads to slighly different definitions.

**Definition 53.** *Given a protocol* $\Pi = (\overline{k}, \overline{n}_I, \overline{n}_R, \dagger_I, \dagger_R, \mathcal{I}, \mathcal{R})$*, the process* $\mathcal{S}_\Pi$ *is defined as follows:*

- *If* $\overline{k} \cap fn(\mathcal{I}) \neq \emptyset$ *and* $\overline{k} \cap fn(\mathcal{R}) \neq \emptyset$*, then* $\mathcal{S}_\Pi \stackrel{\text{def}}{=} ! \, \nu\overline{k}.(\nu\overline{n}_I.\mathcal{I} \mid \nu\overline{n}_R.\mathcal{R})$*;*
- *If* $\overline{k} \cap fn(\mathcal{I}) = \emptyset$ *and* $\overline{k} \cap fn(\mathcal{R}) \neq \emptyset$*, then* $\mathcal{S}_\Pi \stackrel{\text{def}}{=} \dagger_I \, \nu\overline{n}_I.\mathcal{I} \mid ! \, \nu\overline{k}.\nu\overline{n}_R.\mathcal{R}$*.*

Then, unlinkability is defined as a trace equivalence between $\mathcal{S}_\Pi$ (where each identity can execute at most one session) and $\mathcal{M}_\Pi$ (where each identity can execute an arbitrary number of sessions).

**Definition 54.** *A protocol* $\Pi = (\overline{k}, \overline{n}_I, \overline{n}_R, \dagger_I, \dagger_R, \mathcal{I}, \mathcal{R})$ *ensures* unlinkability *if* $\mathcal{M}_\Pi \approx \mathcal{S}_\Pi$.

Even if we consider the notion of trace equivalence instead of the stronger notion of labeled bisimilarity, our definition is in line with the strong notion of unlinkability as proposed in [ACRR10].

**Example 54.** *Going back to our running example (Example 51), unlinkability is expressed through the following trace equivalence:*

$$!\,\nu k.(!\,\nu n_I.P_I \mid !\,\nu n_R.P_R) \approx !\,\nu k.(\nu n_I.P_I \mid \nu n_R.P_R).$$

Although unlinkability of only one role (*e.g.* the tag for RFID protocols) is often considered in the literature (including [ACRR10]), we consider a stronger notion here since both roles are treated symmetrically. We believe this is needed to not miss some practical attacks (see Chapter 10 where such attacks are shown on real-world case studies).

**Example 55.** *We consider the variant of the Feldhofer protocol as described in Example 52. However, for illustrative purposes, we consider* $\Pi^{!}_{\mathsf{Fh}'} \stackrel{\text{def}}{=} (k, \{n_I, r_I\}, \{n_R, r_R\}, !, !, P'_I, P'_R)$, i.e. *a situation where concurrent sessions are authorised. We may be interested in checking unlinkability as in Example 54, i.e. whether the following equivalence holds or not:*

$$!\,\nu k.(\nu(n_I, r_I).P'_I \mid \nu(n_R, r_R).P'_R) \approx !\,\nu k.(!\,\nu(n_I, r_I).P'_I \mid !\,\nu(n_R, r_R).P'_R)$$

*Actually, this equivalence does not hold. When concurrent sessions are authorised, an attacker can interfere and swap messages coming from two different parallel sessions. Performing these swaps, the two sessions will end correctly if, and only if, the keys that are involved in both sessions are the same. Such a scenario, depicted in Figure 8.1 (which is possible on the right-hand side) cannot be mimicked on the left-hand side (each tag can execute only once). Therefore the attacker observes that such an interaction is possible whereas this scenario has no counterpart in the single session scenario. In practice, this can be very harmful when* e.g. *tags are distributed among distinct groups (*e.g. *for access control policies) sharing each the same key* $k$*. By interacting with two tags, the attacker would then be able to know if they belong to the same group. The attacker would thus have the ability to trace a group (rather than individual tags); we still consider the former as unlinkability attack.*

*Now, coming back to the variant of the Feldhofer protocol as decribed in Example 52 (with sessions running sequentially only), it can be shown relying on the technique developed in this part that unlinkability holds. We have that:*

$$!\,\nu k.(\nu(n_I, r_I).P'_I \mid \nu(n_R, r_R).P'_R) \approx !\,\nu k.(¡\,\nu(n_I, r_I).P'_I \mid ¡\,\nu(n_R, r_R).P'_R)$$

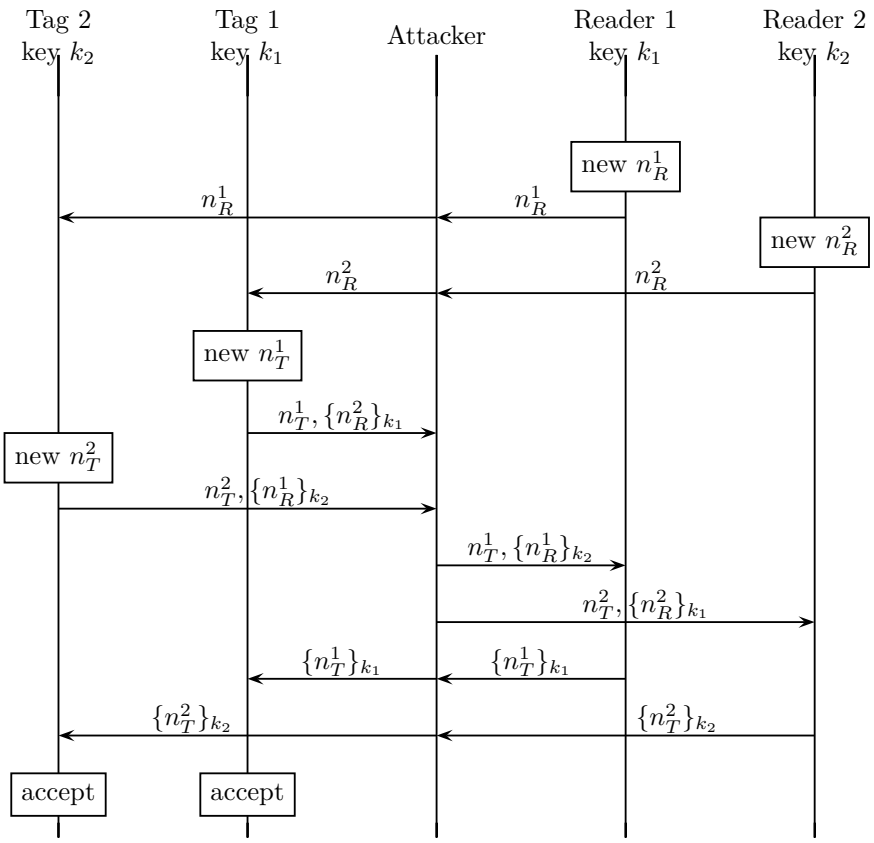


**Figure 8.1** A scenario involving two concurrent sessions (possible only if $k_1 = k_2$).

### 8.3.2 Anonymity

According to the ISO/IEC standard 15408 [ISO09], anonymity aims at ensuring that a user may use a service or a resource without disclosing its identity. In terms of our modelling, a protocol preserves anonymity of some identities $\overline{id} \subseteq \overline{k}$, if a session executed with some particular (public) identities $\overline{\mathsf{id}_0}$ looks to an outsider as if it has been executed with different identity names. In other words, an outside observer should not be able to tell the difference between the original protocol and a version of the protocol where the attacker knows that specific roles $\mathcal{I}$ and $\mathcal{R}$ with identities $\mathsf{id}_0$ (known by the attacker) are present.

**Definition 55.** *Given a protocol* $\Pi = (\overline{k}, \overline{n}_I, \overline{n}_R, \dagger_I, \dagger_R, \mathcal{I}, \mathcal{R})$*, and* $\overline{id} \subseteq \overline{k}$*, the process* $\mathcal{M}_{\Pi,\overline{id}}$ *is defined as follows:*

- *If* $\overline{k} \cap fn(\mathcal{I}) \neq \emptyset$ *and* $\overline{k} \cap fn(\mathcal{R}) \neq \emptyset$*, then* $\mathcal{M}_{\Pi,\overline{id}} \stackrel{\text{def}}{=} \mathcal{M}_\Pi \mid \nu\,\overline{k}.(\dagger_I \nu\,\overline{n}_I.\mathcal{I}_0 \mid \dagger_R \nu\,\overline{n}_R.\mathcal{R}_0)$.
- *If* $\overline{k} \cap fn(\mathcal{I}) = \emptyset$ *and* $\overline{k} \cap fn(\mathcal{R}) \neq \emptyset$*, then* $\mathcal{M}_{\Pi,\overline{id}} \stackrel{\text{def}}{=} \mathcal{M}_\Pi \mid \nu\,\overline{k}.\dagger_R \nu\,\overline{n}_R.\mathcal{R}_0$.

*where* $\mathcal{I}_0 = \mathcal{I}\{\overline{id} \mapsto \overline{\mathsf{id}_0}\}$ *and* $\mathcal{R}_0 = \mathcal{R}\{\overline{id} \mapsto \overline{\mathsf{id}_0}\}$ *for some fresh public constants* $\overline{\mathsf{id}_0}$.

**Definition 56.** *Let* $\Pi = (\overline{k}, \overline{n}_I, \overline{n}_R, \dagger_I, \dagger_R, \mathcal{I}, \mathcal{R})$*, and* $\overline{id} \subseteq \overline{k}$*. We say that* $\Pi$ *ensures* anonymity w.r.t. $\overline{id}$ *if* $\mathcal{M}_{\Pi,\overline{id}} \approx \mathcal{M}_{\Pi}$.

**Example 56.** *Going back to Example 53, anonymity w.r.t. identity of the client (*i.e. $id_C$*) is expressed through the following equivalence (where* $\overline{m_C}$ *represents the sequence* $n_C$*,* $m$*):*

$$\begin{array}{rl} & ! \, \nu n_V.P_V \;\mid\; (! \, \nu(k_C, id_C).\, ! \, \nu\overline{m_C}.P_C) \;\mid\; (\nu(k_C, id_C).! \, \nu\overline{m_C}.P_C\{id_C \mapsto \mathsf{id}_0\}) \\ \approx & ! \, \nu n_V.P_V \;\mid\; (! \, \nu(k_C, id_C).\, ! \, \nu\overline{m_C}.P_C) \end{array}$$

*This intuitively represents the fact that the situation in which a specific client with some known identity* $\mathsf{id}_0$ *may execute some sessions is indistinguishable from a situation in which this client is not present at all. Therefore, if these two situations are indeed indistinguishable from the point of view of the attacker, it would mean that there is no way for the attacker to deduce whether a client with a specific identity is present or not. This particular equivalence that only involves the replication operator (and not* $\mathsf{i}$*) can actually be established relying directly on the* ProVerif *tool. Our method also applies in presence of the* $\mathsf{i}$ *operator whereas* ProVerif *is not able to analyse such a scenario.*

### 8.3.3 Discussion

The definitions we proposed are variations of the ones proposed in [ACRR10]. In particular, they all share the same pattern: we compare some process modelling real usage of the protocol with another process modelling an idealised version of the system. However, a flurry of alternative definitions of unlinkability have been proposed in the literature (see, *e.g.* [BCEDH13, Bru14] for a comparison). Among the strongest ones, various game-based formulations have been considered, both in the computational and symbolic models. Some of these definitions, unlike our definition of unlinkability, can be verified directly in ProVerif using the notion of diff-equivalence [BMU08].

**Game-based definition.** We will not give any formal definition but instead briefly give its general idea. In such a definition, the following game is considered:

1. *Learning phase:* During this phase, the attacker can trigger an arbitrary number of sessions of the two roles (namely tag and reader) with the identity of his choice. This allows him to gain some knowledge. Eventually, the attacker chooses to end the learning phase and enter the second phase.

2. *Guessing phase:* The challenger chooses an identity $x$ among two distinguished identities $id_1$ and $id_2$. The attacker is allowed to interact again (an arbitrary number of times) with roles of $x$, or of identities other than $id_1$ and $id_2$.

The attacker wins the game if he can infer whether $x$ is $id_1$ or $id_2$, *i.e.* if he is able to distinguish between these two scenarios. This is typically the kind of scenario that can be checked relying on the diff-equivalence notion implemented in several automatic tools (*e.g.* ProVerif, Tamarin). However, as illustrated by the following example, we would like to stress that such game-based definitions do not imply unlinkability as defined in this part.

**Example 57.** *We consider the following protocol between a tag $T$ and a reader $R$ that share a symmetric key $k$. We consider that sessions can be executed in parallel, and we assume that $T$ aborts in case the nonce $n_R$ he receives is equal to the nonce $n_T$ he sent previously (in the same session).*

$$\begin{array}{lll} 1. & T \to R: & \{n_T\}_k \\ 2. & R \to T: & \{n_R\}_k \\ 3. & T \to R: & \{n_R \oplus n_T\}_k \end{array}$$

*We consider the term algebra introduced in Example 1, and the equational theory introduced in Example 2 (see Chapter 2) with in addition the equation* $\mathsf{sdec}(\mathsf{senc}(x,y),y) = x$. *To show that the property formally stated in Definition 54 does not hold, consider the following scenario.*

$$\begin{array}{ll} 1.\ T \to R:\ \{n_T\}_k & \\ & 1'.\ T' \to R:\ \{n'_T\}_k \\ 2.\ I(R) \to T:\ \{n'_T\}_k & \\ & 2'.\ I(R) \to T':\ \{n_T\}_k \\ 3.\ T \to R:\ \{n'_T \oplus n_T\}_k & \\ & 3'.\ T' \to R:\ \{n_T \oplus n'_T\}_k \end{array}$$

*A same tag starts two sessions*[1] *and therefore generates two nonces $n_T$ and $n'_T$. The attacker answers to these requests by sending back the two encrypted messages to the tag who will accept both of them, and sends on the network two messages that are actually equal (the exclusive or operator is commutative). Therefore the attacker observes a test, namely the equality between the last two messages, which has no counterpart in the single session scenario. Therefore, this protocol does not ensure unlinkability. In practice, this can be very harmful. Suppose, for example, that tags are distributed among distinct groups (*e.g. *for access control policies) sharing each the same key $k$. By interacting with two tags, the attacker would then be able to know if they belong to the same group and thus be able to trace groups.*

To the best of our knowledge, all the existing game-based variants of unlinkability that are amenable to automation relying on the diff-equivalence notion suffer from this problem. We may also note that the attack described in Example 55 is not captured when considering the game-based version of unlinkability. Actually, this protocol can even be proved secure using ProVerif when considering the game-based version of unlinkability while it suffers from the attack depicted in Figure 8.1.

[1] This is possible if different *physical* tags share the same identity (*e.g.* as it may be the case in access control scenarios). In such a case, it may happen that two different *physical* tags are running two sessions concurrently.

Chapter 9

# Sufficient Conditions for Privacy

We now define our two conditions, namely *frame opacity* (Section 9.2) and *well-authentication* (Section 9.3), and our Main Theorem (Section 9.4) which states that these conditions are sufficient to ensure unlinkability and anonymity as defined in Chapter 8. We conclude the chapter with the proof of our Main Theorem (Section 9.5). Before doing that, we shall introduce annotations in the semantics of processes (Section 9.1) in order to ease their analysis.

## 9.1 Annotations

We shall now define an annotated semantics whose transitions are equipped with more informative actions. The annotated actions will feature labels identifying which concurrent process has performed the action. This will allow us to identify which specific agent (with some specific identity and session names) performed some action.

Formally, an *annotation* is of the form $A(\overline{k}, \overline{n})$ where $A \in \{I, R\}$ and $\overline{k}$, $\overline{n}$ are sequences of names, or constants from $\overline{\mathsf{id}_0}$. Annotations are noted with the letter $a$, and the set of annotations is noted $\mathcal{A}$. In the rest of the part, we shall call *agent* any initiator or responder process (*i.e.* $\mathcal{I}$ or $\mathcal{R}$) or a continuation of those. It corresponds to a certain user (described by its identity names) playing a certain session (described by its session names) at a certain step of its role. Therefore, annotations and agents are closely related (there is a one-to-one correspondence if one forgets about continuations). However, we rely on the notion of agents when we want to point out the process, and we use the notion of annotations to refer to the information that are used to decorate actions and processes.

Given a protocol $\Pi = (\overline{k}, \overline{n}_I, \overline{n}_R, \dagger_I, \dagger_R, \mathcal{I}, \mathcal{R})$ and $\overline{id} \subseteq \overline{k}$, consider any execution of $\mathcal{M}_{\Pi,\overline{id}}$, $\mathcal{M}_\Pi$ or $\mathcal{S}_\Pi$. In such an execution, all $\tau$ actions except $\tau_{\mathsf{then}}$ and $\tau_{\mathsf{else}}$ actions (*i.e.* $\tau_\nu, \tau_!, \tau_r$) are solely used to instantiate new *agents*, by unfolding replications ! (using the semantical rule Repl) or repetitions i (using the semantical rule Unfold), or choosing fresh session and identity names (using the semantical rule New). Performing these actions results in the creation of *agents*, that is, instances of $\mathcal{I}$ and $\mathcal{R}$ with fresh names or constants from $\mathsf{id}_0$. Actions other than $\tau_\nu, \tau_!, \tau_r$ (that is, input, output and conditionals) are then only performed by those *agents*.

The previous remark allows us to define an *annotated semantics* for our processes of interest. In that semantics, agents in the multiset of processes are annotated by their identity (*i.e.* identity and session names that have been created for them), and actions other than $\tau_\nu, \tau_!, \tau_r$ are annotated with the identity of the agent responsible for that action. Note that actions $\tau_{\mathsf{then}}$ and $\tau_{\mathsf{else}}$ will be annotated as well. Traces of the annotated semantics will be denoted by $\mathsf{ta}$. We stress that agents having in their identity constants $\overline{\mathsf{id}_0}$ shall be annotated with some $A(\overline{k}, \overline{n})$ where $\overline{\mathsf{id}_0} \subseteq \overline{k}$. Intuitively, in such a case, we keep the information that the identity $\overline{\mathsf{id}_0}$ of that agent has been disclosed to the attacker in the annotation.

We also assume that labels used to decorate output actions (*i.e.* elements of $\mathcal{L}$) are added to the produced output actions so that we can refer to them when needed (output actions are thus of the form $\ell : \mathtt{out}(c, w)[a]$).

In annotated traces, $\tau_\nu, \tau_!, \tau_r$ actions that are solely used to instantiate new agents are not really important. We sometimes need to reason up to these $\tau$ actions. Given two annotated trace $\mathsf{ta}$ and $\mathsf{ta}'$, we write $\mathsf{ta} \stackrel{\tau}{=} \mathsf{ta}'$ when both traces together with their annotations are equal up to some $\tau_\nu, \tau_!, \tau_r$ actions. We also write $K \stackrel{\mathsf{ta}}{\Longrightarrow} K'$ when $K \xrightarrow{\mathsf{ta}'} K'$ for some $\mathsf{ta}'$ such that $\mathsf{ta} \stackrel{\tau}{=} \mathsf{ta}'$. Finally, an *annotated process* is of the form $P[a]$ where $P$ is a process and $a \in \mathcal{A}$.

**Example 58.** *Considering the protocol $\Pi_{\mathsf{Fh}}$ defined in Example 51, process $\mathcal{S}_{\Pi_{\mathsf{Fh}}}$ can notably perform the execution seen in Example 48. The annotated execution has the trace $\mathsf{ta}$ given below, where $k'$, $n'_I$ and $n'_R$ are fresh names, $a_I = I(k', n'_I)$ and $a_R = R(k', n'_R)$:*

$$\mathsf{ta} = \tau_!.\tau_\nu.\tau_\nu.\tau_\nu.\, \ell_1 : \mathtt{out}(c_I, w_1)[a_I].\mathtt{in}(c_R, w_1)[a_R].\quad \ell_2 : \mathtt{out}(c_R, w_2)[a_R].\mathtt{in}(c_I, w_2)[a_I].\tau_{\mathsf{then}}[a_I].$$
$$\ell_3 : \mathtt{out}(c_I, w_3)[a_I].\mathtt{in}(c_R, w_3)[a_R].\tau_{\mathsf{then}}[a_R]$$

*After the initial $\tau$ actions, the annotated configuration is $(\{\mathcal{I}\sigma_I[a_I], \mathcal{R}\sigma_R[a_R], \mathcal{S}_\Pi\}; \emptyset)$ where $\sigma_I = \{k \mapsto k', n_I \mapsto n'_I\}$, and $\sigma_R = \{k \mapsto k', n_R \mapsto n'_R\}$. The structure is preserved for the rest of the execution of $\mathsf{ta}$, with three processes in the multiset (until they become null), two of which remaining annotated with $a_I$ and $a_R$. After $\mathsf{ta}$, the annotated configuration is $(\{\mathcal{S}_\Pi\}; \Phi_0)$ where $\Phi_0$ has been defined in Example 48.*

**Example 59.** *Going back to Example 56 and starting with $\mathcal{M}_{\Pi, \overline{id}}$, a possible annotated configuration obtained after some $\tau_\nu, \tau_!, \tau_r$ actions can be $K = (\mathcal{P}, \emptyset)$ where $\mathcal{P}$ is a multiset containing the following processes:*

- $P_C\{k_C \mapsto k_C^0, id_C \mapsto \mathsf{id}_0, \overline{m_C} \mapsto \overline{m_C^0}\}[a_C^0]$;
- $!\nu\, \overline{m_C}.P_C\{k_C \mapsto k_C^0, id_C \mapsto \mathsf{id}_0\}$;
- $P_V\{n_V \mapsto n_V^1\}[a_V^1]$; *and*
- $\mathcal{M}_{\Pi_{\mathsf{DAA}}}$.

*where $a_V^1 = \mathcal{I}(\epsilon, n_V^1)$ and $a_C^0 = \mathcal{R}((k_C^0, \mathsf{id}_0), \overline{m_C^0})$. We may note that the annotation $a_V^1$ contains the empty sequence $\epsilon$ since the initiator role does not rely on identity names; and the annotation $a_C^0$ contains $\mathsf{id}_0$.*

## 9.2 Frame Opacity

In light of attacks based on leakage from messages where non-trivial relations between outputted messages are exploited by the attacker to trace an agent, our first condition will express that all relations the attacker can establish on output messages only depend on what is already observable by the attacker and never depend on a priori hidden information such as identity names of specific agents. Therefore, such relations cannot be exploited by the attacker to learn anything new about the agents involved in the execution. We achieve the latter by requiring that any reachable frame must be indistinguishable from an *idealised frame* that only depends on data already observed in the execution, and not on the specific agents (and their names) of that execution.

As a first approximation, one might take the idealisation of a frame $\{w_i \mapsto u_i\}_{i\in I}$ to be $\{w_i \mapsto n_i\}_{i\in I}$ where the $(n_i)_{i\in I}$ are distinct fresh nonces. It would then be very strong to require that frames obtained in arbitrary protocol executions are statically equivalent to their idealisation defined in this way. Although this would allow us to carry out our theoretical development, it would not be realistic since any protocol using, *e.g.* a pair, would fail to satisfy this condition. We thus need a notion of idealisation that retains part of the shape of messages, which a priori does not reveal anything sensitive to the attacker. We also want to allow outputs to depend on session names or previous inputs in ways that are observable, *e.g.* to cover the output of the signature of a previously input message.

Our idealised frames will be obtained by replacing each message, produced by an output of label $\ell$, by a context that only depends on $\ell$, whose holes are filled with fresh session names and (idealisations of) previously inputted messages. Intuitively, this is still enough to ensure that the attacker does not learn anything that is identity-specific. In order to formalise this notion, we assume two disjoint and countable subsets of variables: *input variables* $\mathcal{X}^{\mathsf{i}} = \{x_1^{\mathsf{i}}, x_2^{\mathsf{i}}, \ldots\} \subseteq \mathcal{X}$, and session *name variables* $\mathcal{X}^{\mathsf{n}} = \{x_1^{\mathsf{n}}, x_2^{\mathsf{n}}, \ldots\} \subseteq \mathcal{X}$, and a fixed, arbitrary *idealisation operator* $\mathrm{ideal}(\cdot) : \mathcal{L} \mapsto \mathcal{T}(\Sigma_c, \mathcal{X}^{\mathsf{i}} \cup \mathcal{X}^{\mathsf{n}})$. Variables $x_j^{\mathsf{i}}$ intuitively refers to the $j$-nth variable received by the agent of interest. Therefore, we assume that idealisation operator satisfies the following: $\forall \ell \in \mathcal{L}, \mathrm{ideal}(\ell) \cap \mathcal{X}^{\mathsf{i}} \subseteq \{x_1^{\mathsf{i}}, \ldots, x_k^{\mathsf{i}}\}$ where $k$ is the number of inputs preceding the output labelled $\ell$.

**Definition 57.** *Let* $\mathrm{fr} : \mathcal{A} \times \mathcal{X}^{\mathsf{n}} \mapsto \mathcal{N}$ *be an injective function assigning names to each agent and name variable. We define the idealised frame associated to* $\mathsf{ta}$*, denoted* $\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta})$*, inductively on the annotated trace* $\mathsf{ta}$*:*

- $\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\epsilon) = \emptyset$ *and* $\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta}.\alpha) = \Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta})$ *if* $\alpha$ *is not an output;*
- $\Phi^{\mathrm{fr}}_{\mathrm{ideal}}\big(\mathsf{ta}.(\ell : \mathtt{out}(c, w)[a])\big) = \Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta}) \cup \{w \mapsto \mathrm{ideal}(\ell)\sigma^{\mathsf{i}}\sigma^{\mathsf{n}}\}$ *where*
  - $\sigma^{\mathsf{n}}(x_j^{\mathsf{n}}) = \mathrm{fr}(a, x_j^{\mathsf{n}})$ *when* $x_j^{\mathsf{n}} \in \mathcal{X}^{\mathsf{n}}$*, and*
  - $\sigma^{\mathsf{i}}(x_j^{\mathsf{i}}) = u_j$ *where* $\big(R_j \Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta})\big) \Downarrow u_j$ *when* $x_j^{\mathsf{i}} \in \mathcal{X}^{\mathsf{i}}$ *and* $R_j$ *is the recipe corresponding to the* $j$*-th input of agent* $a$ *in* $\mathsf{ta}$*.*

*Note that this is not necessarily well-defined, as the term* $t_j = R_j\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta})$ *may not compute to a message* $m_j$ *such that* $t_j \Downarrow m_j$*. However, if* $\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta})$ *is well-defined, and* $\mathsf{ta}.\alpha$ *results from an execution together with the frame* $\Phi$ *such that* $\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta}) \sim \Phi$*, then* $\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta}.\alpha)$ *is also well-defined.*

Remark that, by definition, $\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta})$ never depends on the specific identity names occurring in $\mathsf{ta}$ but only on the set of involved agents (without any knowledge about their identity names) and how they interact with each other. In particular, idealised frames do not depend on whether agents rely on the specific constants $\overline{\mathsf{id}_0}$ or not.

**Example 60.** *Continuing Example 58, we may consider the idealisation operator defined as follows:*

$$\ell_1 \mapsto x_1^{\mathsf{n}},\ \ell_2 \mapsto x_2^{\mathsf{n}},\ \ell_3 \mapsto x_3^{\mathsf{n}}.$$

*Let* fr *be an injective function such that* $\mathrm{fr}(a_I, x_j^{\mathsf{n}}) = n_j^I$ *and* $\mathrm{fr}(a_R, x_j^{\mathsf{n}}) = n_j^R$*. We have that* $\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta})$ *is well-defined and*

$$\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta}) = \{w_1 \mapsto n_1^I, w_2 \mapsto n_2^R, w_3 \mapsto n_3^I\}.$$

On the latter simple example, such an idealisation will be sufficient to establish that any reachable frame obtained through an execution of $\mathcal{M}_{\Pi_{\mathsf{Fh}}}$ is indistinguishable from its idealisation. This simple idealisation operator is however not always sufficient for our purposes, and as illustrated by the following two examples, we sometimes need to consider more complex idealisation operators.

**Example 61.** *Continuing Example 52, and to have some hope to establish our indistinguishability property, namely* frame opacity *defined below, we need to define an idealisation operator that retains part of the shape of outputted messages. Typically, assuming that the three outputs are labelled with* $\ell_1$*,* $\ell_2$ *and* $\ell_3$ *respectively, we will consider:*

$$\ell_1 \mapsto x_1^{\mathsf{n}},\ \ell_2 \mapsto \langle x_2^{\mathsf{n}}, x_3^{\mathsf{n}}\rangle,\ \ell_3 \mapsto x_4^{\mathsf{n}}$$

**Example 62.** *Regarding Example 53, we need also to define an idealisation that retains the shape of the second outputted message. Moreover, the idealisation of the second outputted message will depend on the nonce previously received. Typically, assuming that the two outputs are labelled with* $\ell_1$*, and* $\ell_2$ *respectively, we will consider:*

$$\ell_1 \mapsto x_1^{\mathsf{n}},\ \ell_2 \mapsto \mathsf{zk}(\mathsf{sign}(\langle x_2^{\mathsf{n}}, x_3^{\mathsf{n}}\rangle, \mathsf{sk_I}), x_2^{\mathsf{n}}, \mathsf{tuple}(x_1^{\mathsf{i}}, x_4^{\mathsf{n}}, x_5^{\mathsf{n}}, \mathsf{pk}(\mathsf{sk_I})))$$

The following proposition establishes that the particular choice of fr in $\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta})$ is irrelevant with respect to static equivalence. We can thus note $\Phi_{\mathrm{ideal}}(\mathsf{ta}) \sim \Phi$ instead of $\exists \mathrm{fr},\ \Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta}) \sim \Phi$.

**Proposition 23.** *We have* $\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta}) \sim \Phi^{\mathrm{fr}'}_{\mathrm{ideal}}(\mathsf{ta})$ *for any* fr *and* fr′*.*

*Proof.* It suffices to observe that there exists a bijection $\sigma : \mathcal{N} \mapsto \mathcal{N}$ such that $\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta}) = \Phi^{\mathrm{fr}'}_{\mathrm{ideal}}(\mathsf{ta})\sigma$. □

We can now formalise the notion of frame opacity as announced: it requires that all reachable frames must be statically equivalent to idealised frames.

**Definition 58.** *The protocol* $\Pi$ *ensures* frame opacity *if:*

$$\textit{for any execution } (\mathcal{M}_{\Pi,\overline{id}};\emptyset) \xrightarrow{\mathsf{ta}} (Q;\Phi),\ \textit{it holds that } \Phi_{\mathrm{ideal}}(\mathsf{ta}) \sim \Phi.$$

Note that $\Phi_{\mathrm{ideal}}(\mathsf{ta})$ is well-defined above. Formally, it can be established by induction on the length of traces: assuming that $\Pi$ ensures frame opacity for traces of length $n$ then it is easy to establish that $\Phi_{\mathrm{ideal}}(\mathsf{ta})$ is defined for traces $\mathsf{ta}$ executable by $\mathcal{M}_{\Pi,\overline{id}}$ of length at most $n+1$ and the frame opacity condition is thus defined for such traces. In the rest of the part, we will only consider idealisations for protocols that satisfy frame opacity, which ensures well-definedness.

There are many ways to choose the idealisation operator ideal$(\cdot)$. We present below a *canonical syntactical construction* that is sufficient to deal with almost all our case studies. This construction has been implemented as a heuristic to automatically build idealisation operators in the tool UKano. The tool UKano also provides other heuristics that generally lead to better performance but are less tight (*i.e.* they cannot always be used to establish frame opacity). We explain how UKano verifies frame opacity and compare the different heuristics it can leverage in Chapter 10.

At first reading, it is possible to skip the rest of the section and directly go to Section 9.3 since proposed canonical constructions are just instantiations of our generic notion of idealisation.

### 9.2.1 Canonical Syntactical Idealisation

Intuitively, this construction builds the idealisation operator by examining the initiator and responder roles as syntactically given in the protocol definition. The main idea is to consider (syntactical) outputted terms one by one, and to replace identity names and variables bound by a let construct by pairwise distinct session names variables.

**Definition 59.** *Let* $\Pi = (\overline{k}, \overline{n}_I, \overline{n}_R, \dagger_I, \dagger_R, \mathcal{I}, \mathcal{R})$ *be a protocol that uses input variables* $\{x^{\mathsf{i}}_1, x^{\mathsf{i}}_2, \ldots\} \subseteq \mathcal{X}^{\mathsf{i}}$ *(in this order) for its two roles, and distinct variables from* $\mathcal{X}_{\mathsf{let}}$ *in let constructions. Let* $\sigma$ *be an injective renaming from* $\overline{k} \cup \overline{n_R} \cup \overline{n_I} \cup \mathcal{X}_{\mathsf{let}} \to \mathcal{X}^{\mathsf{n}}$*. The canonical syntactical idealisation operator associated to* $\Pi$ *maps any* $\ell \in \mathcal{L}$ *occurring in an output action of the form* $\ell : \mathtt{out}(c,u)$ *(for some* $c$ *and some* $u$*) in* $\mathcal{I}$ *or* $\mathcal{R}$ *to* $u\sigma$*.*

**Example 63** (Continuing Example 51)**.** *First, we perform some renaming to satisfy the conditions imposed by the previous definition. We therefore replace* $x_1$ *by* $x^{\mathsf{i}}_1$ *in role* $\mathcal{I}$*, and* $y_1, y_2$ *by* $x^{\mathsf{i}}_1, x^{\mathsf{i}}_2$ *in role* $\mathcal{R}$*. We assume that* $x_2, x_3$*, and* $y_3$ *are elements of* $\mathcal{X}_{\mathsf{let}}$*. We consider a renaming* $\sigma$ *that maps* $k, n_I, n_R, x_2, x_3, y_3$ *to* $x^{\mathsf{n}}_1, \ldots, x^{\mathsf{n}}_6$*. We obtain the following idealisation operator:*

$$\ell_1 \mapsto x^{\mathsf{n}}_2; \quad \ell_2 \mapsto \mathsf{senc}(\langle x^{\mathsf{i}}_1, x^{\mathsf{n}}_3\rangle, x^{\mathsf{n}}_1); \quad \ell_3 \mapsto \mathsf{senc}(\langle x^{\mathsf{n}}_5, x^{\mathsf{n}}_2\rangle, x^{\mathsf{n}}_1)$$

*Considering* fr *as defined in Example 60,* i.e. *such that* $\mathrm{fr}(a_I, x^{\mathsf{n}}_j) = n^I_j$ *and* $\mathrm{fr}(a_R, x^{\mathsf{n}}_j) = n^R_j$*, and relying on the idealisation operator defined above, we have that:*

$$\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta}) = \{w_1 \mapsto n^I_2,\ w_2 \mapsto \mathsf{senc}(\langle n^I_2, n^R_3\rangle, n^R_1),\ w_3 \mapsto \mathsf{senc}(\langle n^I_5, n^I_2\rangle, n^I_1)\}$$

*where* $\mathsf{ta}$ *is the trace given in Example 58.*

*Although this canonical syntactical idealisation is quite different from the one described in Example 60, it also allows us to establish that any reachable frame obtained through an execution of* $\mathcal{M}_{\Pi_{\mathsf{Fh}}}$ *is indistinguishable from its idealisation.*

**Example 64** (Continuing Example 52)**.** *Consider a renaming* $\sigma$ *that maps* $k$, $n_I$, $n_R$, $x_2$, $x_3$, $y_3$, $r_I$, $r_R$ *to* $x_1^{\mathsf{n}}, \ldots, x_8^{\mathsf{n}}$*. We obtain the following idealisation operator:*

$$\ell_1 \mapsto x_2^{\mathsf{n}}; \quad \ell_2 \mapsto \langle x_3^{\mathsf{n}}, \mathsf{renc}(x_1^{\mathsf{i}}, x_1^{\mathsf{n}}, x_8^{\mathsf{n}})\rangle; \quad \ell_3 \mapsto \mathsf{renc}(x_5^{\mathsf{n}}, x_1^{\mathsf{n}}, x_7^{\mathsf{n}})$$

*Such an idealisation operator is suitable to establish frame opacity.*

**Example 65** (Continuing Example 53)**.** *Consider a renaming* $\sigma$ *that maps:* $k_C$, $id_C$, $n_V$, $n_C$, $m$, $x_2$, $x_3$, $x_4$, $y_3$ *to* $x_1^{\mathsf{n}}, x_2^{\mathsf{n}}, \ldots, x_9^{\mathsf{n}}$*. We obtain the following idealisation operator:*

$$\ell_1 \mapsto x_3^{\mathsf{n}}; \quad \ell_2 \mapsto \mathsf{zk}(\mathsf{sign}(\langle x_1^{\mathsf{n}}, x_2^{\mathsf{n}}\rangle, \mathsf{sk_I}), x_1^{\mathsf{n}}, \mathsf{tuple}(x_1^{\mathsf{i}}, x_4^{\mathsf{n}}, x_5^{\mathsf{n}}, \mathsf{pk}(\mathsf{sk_I}))); \quad \ell_3 \mapsto \mathsf{error}$$

*Such an idealisation operator is also suitable to establish frame opacity.*

As illustrated by the previous examples, this canonical syntactical idealisation is sufficient to conclude on most examples. Actually, using this canonical construction, we automatically build the idealisation operator and check frame opacity for all the examples we have introduced in the previous sections and for most of the case studies presented in Chapter 10.

### 9.2.2 Semantical Idealisation

However, the previous construction is clearly purely syntactic and therefore closely connected to the way we write the roles of the protocol. Its main weakness is the way it deals with variables bound by let constructions. Since there is no way to statically guess the shape of messages that will be instantiated for those variables, the previous technique replaces them by fresh session names. False negatives may result from such over-approximations.

We may therefore prefer to build an idealisation operator looking at the messages outputted during a concrete execution. In such a case, we may simply retain part of the shape of messages, which a priori does not reveal anything sensitive to the attacker (*e.g.* pairs, lists). This can be formalised as follows:

**Definition 60.** *A symbol* $\mathsf{f}$ *(of arity* $n$*) in* $\Sigma$ *is* transparent *if it is a public constructor symbol that does not occur in* $\mathsf{E}$ *and such that: for all* $1 \leq i \leq n$*, there exists a recipe* $R_i \in \mathcal{T}(\Sigma_{\mathsf{pub}}, \{w\})$ *such that for any message* $u = \mathsf{f}(u_1, \ldots, u_n)$*, we have that* $R_i\{w \mapsto u\} \Downarrow v_i$ *for some* $v_i$ *such that* $v_i =_{\mathsf{E}} u_i$*.*

**Example 66.** *Considering the signature and the equational theory introduced in Example 1 and Example 2, the symbols* $\langle\ \rangle$ *and* $\mathsf{ok}$ *are the only ones that are transparent. Regarding the pairing operator, the recipes* $R_1 = \mathsf{proj}_1(w)$ *and* $R_2 = \mathsf{proj}_2(w)$ *satisfy the requirements.*

Once such a set $\Sigma_t$ is fixed, the idealisation associated to a label $\ell$ occurring in $\Pi$ will be computed relying on a particular (but arbitrary) message $u$ that has been outputted with this

label $\ell$ during a concrete execution of $\mathcal{M}_\Pi$. The main idea is to go through transparent functions until getting stuck, and then replacing the remaining sub-terms using distinct name variables from $\mathcal{X}^{\mathsf{n}}$.

**Example 67.** *Consider the protocol given in Example 51 and the frame $\Phi_0$ as defined in Example 48, the resulting idealisation associated to $\Pi$ (considering the messages outputted in $\Phi_0$) is: $\ell_1 \mapsto x_1^{\mathsf{n}}$; $\ell_2 \mapsto x_2^{\mathsf{n}}$; $\ell_3 \mapsto x_3^{\mathsf{n}}$. The protocol given in Example 52 will give us: $\ell_1 \mapsto x_1^{\mathsf{n}}$; $\ell_2 \mapsto \langle x_2^{\mathsf{n}}, x_3^{\mathsf{n}}\rangle$; $\ell_3 \mapsto x_4^{\mathsf{n}}$. Even if these idealisation operators are quite different from the ones presented in Example 63 and Example 64, they are also suitable to establish frame opacity.*

In [HBD16], the idealisation operator associated to $\Pi$ was exclusively computed using this method. However, it happens to be insufficient to establish frame opacity in presence of function symbols that are neither transparent nor totally opaque such as signatures. Indeed, a signature function symbol is not transparent according to our definition, but an attacker can make the difference between a signature $\mathsf{sign}(m, sk(A))$ and a random nonce. Therefore, replacing such a term by a fresh session name will never allow one to establish frame opacity.

However, this construction leads to simpler idealisation operators that often allow better performance. We thus also have implemented a heuristic following the above ideas in the tool UKano.

## 9.3 Well-Authentication

Our second condition will prevent the attacker from obtaining some information about agents through the outcome of conditionals. To do so, we will essentially require that conditionals of $\mathcal{I}$ and $\mathcal{R}$ can only be executed successfully in honest, intended interactions. However, it is unnecessary to impose such a condition on conditionals that never leak any information, which are found in several security protocols. We characterise below a simple class of such conditionals, for which the attacker will always know the outcome of the conditional based on the past interaction.

**Definition 61.** *A conditional* $\mathsf{let}\ \overline{z} = \overline{t}\ \mathtt{in}\ P\ \mathsf{else}\ Q$ *occurring in $\mathcal{A} \in \{\mathcal{I}, \mathcal{R}\}$ is* safe *if $\overline{t} \in \mathcal{T}(\Sigma_{\mathsf{pub}}, \{x_1, \ldots, x_n\} \cup \{u_1, \ldots, u_n\})$, where the $x_i$ are the variables bound by the previous inputs of that role, and $u_i$ are the messages used in the previous outputs of that role.*

**Example 68.** *Consider the process given below:*

$$\mathtt{out}(c, u).\mathtt{in}(c, x).\mathsf{let}\ z = \mathsf{neq}(x, u)\ \mathtt{in}\ P\ \mathsf{else}\ Q$$

*The conditional is used to ensure that the agent will not accept as input the message he sent at the previous step. Such a conditional is safe according to our definition.*

Note that trivial conditionals required by the grammar of protocols (Definition 49) are safe and will thus not get in the way of our analysis. We can now formalise the notion of association, which expresses that two agents are having an honest, intended interaction (*i.e.* the attacker essentially did not interfere in their communications). For an annotated trace $\mathsf{ta}$ and annotations

$a$ and $a'$, we denote by $\mathsf{ta}|_{a,a'}$ the subsequence of $\mathsf{ta}$ that consists of actions of the form $\alpha[a]$ or $\alpha[a']$.

**Definition 62.** *Given a protocol $\Pi$, we say that two annotations $a_1 = A_1(\overline{k}_1, \overline{n}_1)$ and $a_2 = A_2(\overline{k}_2, \overline{n}_2)$ are* associated *in $(\mathsf{ta}, \Phi)$ if:*

- *they are* dual, i.e. *$A_1 \neq A_2$, and $\overline{k_1} = \overline{k_2}$ when $fn(\mathcal{R}) \cap fn(\mathcal{I}) \neq \emptyset$ (the shared case);*
- *the interaction $\mathsf{ta}|_{a_1,a_2}$ is honest for $\Phi$ (see Definition 50).*

**Example 69.** *Continuing Example 58, the annotations $I(k', n'_I)$ and $R(k', n'_R)$ are associated in $(\mathsf{ta}, \Phi_0)$.*

Finally, we can state our second condition.

**Definition 63.** *The protocol $\Pi$ is* well-authenticating *if, for any execution $(\mathcal{M}_{\Pi,\overline{id}}; \emptyset) \xrightarrow{\mathsf{ta}.\tau_{\mathsf{then}}[a]} (\mathcal{P}; \Phi)$ either the last action corresponds to a safe conditional, or there exists $a'$ such that:*

*(i) The annotations $a$ and $a'$ are associated in $(\mathsf{ta}, \Phi)$;*

*(ii) Moreover, when $fn(\mathcal{R}) \cap fn(\mathcal{I}) \neq \emptyset$ (the shared case), $a'$ (resp. $a$) is only associated with $a$ (resp. $a'$) in $(\mathsf{ta}, \Phi)$.*

Intuitively, this condition does not require anything for safe conditional as we already know that they cannot leak new information to the attacker (he already knows their outcome). For unsafe conditionals, condition *(i)* requires that whenever an agent $a$ evaluates them positively (*i.e.* he does not abort the protocol), it must be the case that this agent $a$ is so far having an honest interaction with a dual agent $a'$. Indeed, as discussed in introduction, it is crucial to avoid such unsafe conditionals to be evaluated positively when the attacker is interfering because this could leak crucial information.

As illustrated in the following example, Condition *(ii)* is needed to prevent from having executions where an annotation is associated to several annotations, which would break unlinkability in the shared case (*i.e.* when $fn(\mathcal{R}) \cap fn(\mathcal{I}) \neq \emptyset$).

**Example 70.** *We consider a protocol between an initiator and a responder that share a symmetric key $k$. The protocol can be described informally as follows:*

$$1.\quad I \to R: \quad \{n_I\}_k$$
$$2.\quad R \to I: \quad n_R$$

*Assuming that the two outputs are labelled with $\ell_1$ and $\ell_2$ respectively, we have that the idealisation operator: $\ell_1 \mapsto x_1^{\mathsf{n}}$, $\ell_2 \mapsto x_2^{\mathsf{n}}$ is suitable to establish frame opacity.*

*We may note that the only conditional is the one performed by the responder role when receiving the ciphertext. He will check whether it is indeed an encryption with the expected key $k$. When an action $\tau_{\mathsf{then}}[R(k, n_R)]$ occurs, it means that a ciphertext encrypted with $k$ has been received by $R(k, n_R)$ and since the key $k$ is unknown by the attacker, such a ciphertext has been sent by a participant: this is necessarily a participant executing the initiator role with key $k$. Hence condition $(i)$ of well-authentication holds (and can actually be formally proved). However, condition*

| Protocol | Frame opacity | Well-authentication | Unlinkability |
|---|---|---|---|
| Feldhofer (Example 51) | ✓ | ✓ | **safe** |
| Feldhofer variant with ! (Example 55) | ✓ | × | attack |
| Feldhofer variant with ¡ (Example 52) | ✓ | ✓(with Tamarin) | **safe** |
| DAA-like (Example 53) | ✓ | ✓ | **safe** |

**Table 9.1** Summary of our running examples

*(ii) fails to hold since two responder roles may accept a same ciphertext* $\{n_I\}_k$ *and therefore be associated to the same agent acting as an initiator. This corresponds to an attack scenario w.r.t. our formal definition of unlinkability since such a trace will have no counterpart in* $\mathcal{S}_\Pi$*. More formally, the trace* $\mathsf{tr} = \mathtt{out}(c_I, w_0).\mathtt{in}(c_R, w_0).\tau_{\mathsf{then}}.\mathtt{out}(c_R, w_1).\mathtt{in}(c_R, w_0).\tau_{\mathsf{then}}.\mathtt{out}(c_R, w_2)$ *will be executable starting from* $\mathcal{M}_\Pi$ *and will allow one to reach that frame:*

$$\Phi = \{w_0 \mapsto \mathsf{senc}(n_I, k);\ w_1 \mapsto n_R;\ w_2 \mapsto n'_R\}$$

*Starting from* $\mathcal{S}_\Pi$ *the second action* $\tau_{\mathsf{then}}$ *will not be possible, and more importantly this will prevent the observable action* $\mathtt{out}(c_R, w_2)$ *to be triggered.*

## 9.4 Main Theorem: Soundness of Conditions w.r.t. Privacy

Our main theorem establishes that the previous two conditions are always sufficient to ensure unlinkability and anonymity for security protocols in our class.

**Theorem 8.** *Consider a protocol* $\Pi = (\overline{k}, \overline{n}_I, \overline{n}_R, \dagger_I, \dagger_R, \mathcal{I}, \mathcal{R})$ *and some identity names* $\overline{id} \subseteq \overline{k}$*. If the protocol is well-authenticating and ensures frame opacity, then* $\Pi$ *ensures unlinkability and anonymity w.r.t.* $\overline{id}$*.*

Note that, when $\overline{id} = \emptyset$, we have that $\mathcal{M}_{\Pi,\overline{id}} \approx \mathcal{M}_\Pi$ and our two conditions coincide on $\mathcal{M}_{\Pi,\overline{id}}$ and $\mathcal{M}_\Pi$. We thus have as a corollary that if $\mathcal{M}_\Pi$ ensures well-authentication and frame opacity, then $\Pi$ is unlinkable.

Before proving our Main Theorem in the next section, we show its applicability by summarising the different case studies we were able to verify notably using our theorem.

We first summarise in Table 9.1 the result of the confrontation of our method to our various running examples, focusing on unlinkability. We note ✓ for a condition automatically checked using our tool UKano and × when the condition does not hold. For the analysis of Feldhofer variant with ¡, we do not rely on UKano (which is based on ProVerif) since ProVerif does not support the ¡ operator. We establish the well-authentication property using Tamarin and frame opacity using ProVerif by allowing sessions to be run concurrently and thus doing a sound over-approximation of the protocol's behaviours. Remark that being able to verify different conditions using different methods or tools is also an advantage of our method. Except for the latter example, all positive results were automatically established using our tool UKano.

In Table 9.2, we summarise the result of the confrontation of our method to our real-world case studies, focusing on unlinkability. Detailed descriptions of our case studies are in Chapter 10.

| Protocol | Frame opacity | Well-auth. | Unlinkability |
|---|---|---|---|
| Hash-Lock | ✓ | ✓ | **safe** |
| LAK (stateless) | – | × | attack |
| Fixed LAK | ✓ | ✓ | **safe** |
| BAC | ✓ | ✓ | **safe** |
| BAC/PA/AA | ✓ | ✓ | **safe** |
| BAC/AA/PA | ✓ | ✓ | **safe** |
| PACE (faillible dec) | – | × | attack |
| PACE (as in [BFK09]) | – | × | attack |
| PACE | – | × | attack |
| PACE with tags | ✓* | ✓ | **safe** |
| DAA sign | ✓ | ✓ | **safe** |
| DAA join | ✓ | ✓ | **safe** |
| ABCDH (irma) | ✓* | ✓* | **safe** |

**Table 9.2** Summary of our case studies

We remark that our conditions have proven to be tight enough for all our case studies: when a condition fails to hold, we could always discover a real attack on unlinkability. Most of the positive results (when unlinkability holds) and all attacks are new. Note that all positive results were established automatically using our tool UKano (which is based on ProVerif) without any manual effort (except for the cases indicated by ✓* for which little manual efforts were needed).

## 9.5 Proof of our Main Theorem

We provide in this section the proof of Theorem 8. Our main argument consists in showing that, for any execution of $\mathcal{M}_{\Pi,\overline{id}}$, there is an indistinguishable execution of $\mathcal{S}_\Pi$ (the other direction being easy). This indistinguishable execution will be obtained by a modification of the involved agents. We will proceed via a renaming of agents applied to an abstraction of the given execution of $\mathcal{M}_{\Pi,\overline{id}}$. We then prove that the renamed executions can still be executed and produce an indistinguishable frame.

In this section we fix a protocol $\Pi$, some identity names $\overline{id}$ and some fresh constants $\overline{\mathsf{id}_0}$ yielding a process $\mathcal{M}_{\Pi,\overline{id}}$ as defined in Section 8.3. The construction of the proof will slightly differ depending on $\dagger_I, \dagger_R$ (sequential or concurrent sessions) and whether $fn(\mathcal{I}) \cap fn(\mathcal{R}) = \emptyset$ or not (shared case).

### 9.5.1 Abstraction of Configurations

Instead of working with $\mathcal{M}_{\Pi,\overline{id}}$, $\mathcal{M}_\Pi$, and $\mathcal{S}_\Pi$, it will be more convenient to work with *ground configurations*. Intuitively, we will associate to each execution of $\mathcal{M}_{\Pi,\overline{id}}$, $\mathcal{M}_\Pi$, and $\mathcal{S}_\Pi$, a ground configuration that contains all agents involved in that execution, already correctly instantiated. By doing so, we are able to get rid of technical details such as unfolding replications and rep-

etitions a necessary number of times, create necessary identity and session names, etc. These ground configurations are generated from sequences of annotations satisfying some requirements.

### Sequences of Annotations

The sequence of annotations from which we will build ground configurations shall satisfy some requirements that we list below. Essentially, the goal is to make sure that no freshness condition over session and identity names is violated.

**Definition 64.** *A sequence $S$ of annotations is* well-formed *if the following conditions hold.*

- *In all annotations $A(\overline{k'}, \overline{n'})$, the session parameters $\overline{n'}$ are names, and the identity parameters $\overline{k'}$ are made of names or constants $\overline{\mathsf{id}_0}$. We also have that $|\overline{n'}| = |\overline{n_A}|$ and $|\overline{k'}| = |\overline{k}|$ (remind that $\overline{n_I}, \overline{n_R}$ and $\overline{k}$ are session and identity names from $\Pi$). Moreover, when $\overline{\mathsf{id}_0} \cap \overline{k'} \neq \emptyset$, then it holds that $k'_i = \mathsf{id}_{0i}$ if, and only if, $k_i = id_i$ (where $\mathsf{id}_{0i}, id_i, k_i, k'_i$ are respectively the i-nth element of the sequences $\overline{\mathsf{id}_0}, \overline{id}, \overline{k}, \overline{k'}$).*
- *No name appears both as identity and session parameter in any two annotations.*
- *Two different annotations never share a session parameter.*
- *Two annotations either have the same identity parameters, or do not share any identity parameter at all.*

*We say that a sequence of annotations $S$ is* single-session *when two different annotations of the same role never share an identity parameter and no annotation contains a constant in $\overline{\mathsf{id}_0}$.*

We straightforwardly lift those definitions to annotated traces by only keeping the underlying sequence of annotations and dropping actions. Roughly, well-formed (resp. well-formed, single-session) sequence of annotations contains annotations of agents that can be instantiated from $\mathcal{M}_{\Pi,\overline{id}}$ (resp. $\mathcal{S}_\Pi$). Conversely, we may note that given $\mathsf{ta}$ such that $\mathcal{M}_{\Pi,\overline{id}} \xrightarrow{\mathsf{ta}} K'$, we have that $\mathsf{ta}$ is well-formed.

### Ground Configurations

After the introduction of some notations, we explain how *ground configurations* are obtained from well-formed sequences of annotations.

Given a well-formed sequence of annotations $S$, and some role $A \in \{I, R\}$, we introduce following notations:

- we note $\mathrm{id}_A(S)$ the set of identity parameters of agents of role $A$ occurring in $S$ (*i.e.* the set $\{\overline{k} \mid A(\overline{k}, \overline{n}) \in S\}$),
- for $\overline{l} \in \mathrm{id}_A(S)$, we note $\mathrm{sess}_A(S, \overline{l})$ the set of session parameters of agents of role $A$ and identity parameters $\overline{l}$ occurring in $S$ (*i.e.* the set $\{\overline{n} \mid A(\overline{l}, \overline{n}) \in S\}$),

- for $\bar{l} \in \mathrm{id}_A(S)$, we note $\mathrm{sess}_A^{\mathsf{seq}}(S, \bar{l})$ the sequence made of elements from $\mathrm{sess}_A(S, \bar{l})$, without repetition, in order of first occurrence in $S$.

Finally, for some sequence of elements $L = e_1, e_2, e_3, \ldots$ and a family of processes $\{P(e_i)\}_i$, we denote by $\coprod_{e \in L} P(e)$ the process $P(e_1); (P(e_2); (P(e_3); \ldots))$.

**Definition 65.** *Let $S$ be a well-formed sequence of annotations. The* ground configuration associated to $S$ *denoted by $\mathcal{K}(S)$ is the multiset $\mathcal{P}_I \sqcup \mathcal{P}_R$ where $\mathcal{P}_A$ is defined as follows depending on $\dagger_A$ , where $\mathcal{A}$ is the process of $\Pi$ corresponding to the role annotation $A$:*

- *if $\dagger_A = !$ ($A$ can execute sessions concurrently) then*

$$\mathcal{P}_A = \left\{ \left(\mathcal{A}\{\overline{k} \mapsto \bar{l}, \overline{n}_A \mapsto \overline{m}\}\right)[A(\bar{l}, \overline{m})] \;\middle|\; A(\bar{l}, \overline{m}) \in S \right\}.$$

- *if $\dagger_A = \mathsf{i}$ ($A$ cannot execute sessions concurrently) then*

$$\mathcal{P}_A = \left\{ \coprod_{\overline{m} \in S_{\bar{l}}} Q_A^{\bar{l}}(\overline{m}) \;\middle|\; \bar{l} \in \mathrm{id}_A(S) \text{ and } S_{\bar{l}} = \mathrm{sess}_A^{\mathsf{seq}}(S, \bar{l}) \right\}.$$

  *where $Q_A^{\bar{l}}(\overline{m}) = \left(\mathcal{A}\{\overline{k} \mapsto \bar{l}, \overline{n}_A \mapsto \overline{m}\}\right)[A(\bar{l}, \overline{m})]$.*

Note that, as illustrated by the next example, we now consider annotations inside a process.

**Example 71.** *Consider for instance a trivial protocol $\Pi \stackrel{\text{def}}{=} (k, n_I, n_R, \mathsf{i}, \mathsf{i}, \mathcal{I}, \mathcal{R})$ where $\mathcal{I} = \mathtt{out}(c_I, \mathsf{senc}(n_I, k))$ and $\mathcal{R} = \mathtt{in}(c_R, x)$. We have that*

$$\mathcal{M}_\Pi = !\, \nu k.(\mathsf{i}\ \nu n_I.\mathcal{I} \mid \mathsf{i}\ \nu n_R.\mathcal{R}) \xrightarrow{\mathsf{ta}} (\mathcal{Q}; \phi)$$

*for $\mathsf{ta} = \tau_!.\tau_\nu.\tau_r.\tau_\nu.\ \ell : \mathtt{out}(c_I, w_0)[I(k, n_I)].\tau_r.\tau_\nu.\ \ell : \mathtt{out}(c_I, w_1)[I(k, n'_I)]$. The ground configuration associated to $\mathsf{ta}$ is as follows:*

$$\mathcal{K}(\mathsf{ta}) = \coprod_{n \in (n_I), (n'_I)} Q_I^k(n) = (\mathtt{out}(c_I, \mathsf{senc}(n_I, k))[I(k, n_I)]); (\mathtt{out}(c_I, \mathsf{senc}(n'_I, k))[I(k, n'_I)]).$$

*Note that $\mathcal{K}(\mathsf{ta})$ is also able to produce the annotated trace $\mathsf{ta}$ up to $\stackrel{\tau}{=}$.*

We lift those definitions to annotated traces as before. A ground configuration associated to a well-formed sequence of annotations is essentially an "unfolding" of $\mathcal{M}_{\Pi,\overline{id}}$. Therefore, there is a strong relationship between the original process and the one obtained through $\mathcal{K}(\cdot)$ as shown next.

**Proposition 24.** *If $\mathsf{ta}$ is a well-formed annotated trace then the following hold:*

*(1) If $\mathcal{M}_{\Pi,\overline{id}} \xRightarrow{\mathsf{ta}} K$ (resp. $\mathcal{S}_\Pi \xRightarrow{\mathsf{ta}} K$), then $\mathcal{K}(\mathsf{ta}) \xRightarrow{\mathsf{ta}} K'$ for some $K'$ such that $\Phi(K) = \Phi(K')$.*

*(2) If $\mathcal{K}(\mathsf{ta}) \xRightarrow{\mathsf{ta}} K$, then $\mathcal{M}_{\Pi,\overline{id}} \xRightarrow{\mathsf{ta}} K'$ for some $K'$ such that $\Phi(K) = \Phi(K')$.*

*(3) If* $\mathcal{K}(\mathsf{ta}) \stackrel{\mathsf{ta}}{\Longrightarrow} K$ *and* $\mathsf{ta}$ *is single-session, then* $\mathcal{S}_\Pi \stackrel{\mathsf{ta}}{\Longrightarrow} K'$ *for some* $K'$ *such that* $\Phi(K) = \Phi(K')$.

*(4) If* $\mathsf{ta} = \mathsf{ta}_1.\mathsf{ta}_2$ *and* $\mathcal{K}(\mathsf{ta}_1.\mathsf{ta}_2) \xrightarrow{\mathsf{ta}_1} K$ *then* $\mathcal{K}(\mathsf{ta}_1) \xrightarrow{\mathsf{ta}_1} K'$ *with* $\Phi(K) = \Phi(K')$.

*Proof.* Item (1) holds by construction of the operator $\mathcal{K}(\cdot)$, which has been built by closely mimicking how $\mathcal{M}_{\Pi,\overline{id}}$ and $\mathcal{S}_\Pi$ create agents. We thus have that when an agent is at top-level in a configuration in the execution of $\mathcal{M}_{\Pi,\overline{id}}$ (or $\mathcal{S}_\Pi$) then it is also available in the execution of $\mathcal{K}(\mathsf{ta})$. In general, less $\tau_\nu, \tau_!, \tau_r$ actions are necessary for the execution starting with $\mathcal{K}(\mathsf{ta})$ than for the executions starting with $\mathcal{M}_{\Pi,\overline{id}}$ or $\mathcal{S}_\Pi$. Indeed, there is no need, in ground configurations, to spawn agents by unfolding replications or repetitions or creating fresh names. This is because agents are (more) immediately available in $\mathcal{K}(\mathsf{ta})$.

Item (2) heavily relies on the well-formedness of $\mathsf{ta}$. One can thus prove that all agents in $\mathcal{K}(\mathsf{ta})$ can be created along the execution by choosing appropriate names when triggering rules $\mathsf{New}$. For instance, the first item of Definition 64 makes sure that the arity of parameters in agents matches the number of names to be created. The second and third items of Definition 64 ensure that the freshness guard conditions of the rule $\mathsf{New}$ holds for names to be created. Finally, the fourth item of Definition 64 implies that when an agent $a = A(\overline{k}, \overline{n})$ must be created then either (i) names in $\overline{k}$ are completely fresh and this identity $\overline{k}$ can be created from $\mathcal{M}_{\Pi,\overline{id}}$ by unfolding ! and create names $\overline{k}$ or (ii) names in $\overline{k}$ have already been created and thus the agent $a$ can be created from the last replicated process used to create identity $\overline{k}$ in the first place.

Item (3) is similar to (2). The single-session hypothesis provides exactly what is needed to mimic the execution using $\mathcal{S}_\Pi$ rather than $\mathcal{M}_{\Pi,\overline{id}}$.

Finally, item (4) stems from a simple observation. Compared to $\mathcal{K}(\mathsf{ta}_1)$, the multiset of processes $\mathcal{K}(\mathsf{ta}_1.\mathsf{ta}_2)$ adds processes in parallel and in sequence after some processes of $\mathcal{K}(\mathsf{ta}_1)$. However, these extra processes are unused when executing $\mathsf{ta}_1$, thus $\mathcal{K}(\mathsf{ta}_1)$ can perform the same execution. □

#### Renamings of Annotations

As mentioned before, we shall prove that for any execution of $\mathcal{M}_{\Pi,\overline{id}}$, there is an indistinguishable execution of $\mathcal{S}_\Pi$. This indistinguishable execution that $\mathcal{S}_\Pi$ can perform will be obtained by a renaming of annotations. We define next a generic notion of such renamings of annotations. However, the crux of the final proof is to find a *good* renaming that implies: (i) the executability by $\mathcal{S}_\Pi$ of the renamed trace, (ii) the static indistinguishability of the resulting frames (before and after the renaming).

**Definition 66.** *A* renaming of annotations *(denoted by* $\rho$*) is an injective mapping from annotations to annotations such that:*

- *for any well-formed sequence of annotations* $S$*,* $S\rho$ *is well-formed;*
- $\rho$ *is role-preserving:* i.e. *initiator (resp. responder) annotations are mapped to initiator (resp. responder) annotations;*

- *for any* $a_1 = A_1(\overline{k_1}, \overline{n_1})$, $a_2 = A_2(\overline{k_2}, \overline{n_2}) \in \mathcal{A}$, *if* $\rho(a_1)$ *and* $\rho(a_2)$ *have the same identity parameters, then so do* $a_1$ *and* $a_2$ *(*i.e. $\overline{k_1} = \overline{k_2}$*).*

The two first conditions are expected: renaming of annotations shall only modify session and identity parameters whilst preserving well-formedness. The final condition ensures that renamings do not create more "sequential dependencies" between agents (*i.e.* agents sharing the same identity and whose role can execute sessions only sequentially): after renaming, less pairs of agents have same identity.

Next, we define $\mathsf{ta}\rho$ as the annotated trace obtained from $\mathsf{ta}$ by applying $\rho$ to annotations only. Note that, by definition of renamings, the resulting $\mathsf{ta}\rho$ is well-formed as well.

One can also define the effect of renamings on ground configurations. If $\rho(A(\overline{k}, \overline{n})) = A(\overline{k'}, \overline{n'})$, the renaming $\sigma$ induced by $\rho$ on $A(\overline{k}, \overline{n})$ is the (injective) mapping such that $\overline{k}\sigma = \overline{k'}$ and $\overline{n}\sigma = \overline{n'}$. Given a ground configuration $\mathcal{P} = \{\coprod_j P_j^i[a_j^i]\}_i$, we define $\mathcal{P}\rho = \{\coprod_j P_j^i \sigma_j^i[\rho(a_j^i)]\}_i$ where $\sigma_j^i$ is the renaming induced by $\rho$ on $a_j^i$. Note that the renaming on parameters induced by a renaming of annotations may conflict: this happens, for example, when $\rho(A(\overline{k}, \overline{n})) = A(\overline{k_1}, \overline{n})$ and $\rho(A(\overline{k}, \overline{m})) = A(\overline{k_2}, \overline{m})$.

A renaming of annotations can break executability. Even when executability is preserved, it is not obvious to relate processes before and after the renaming, as messages can be affected in complex ways and conditionals may not evaluate to the same outcome. Fortunately, frame opacity and well-authentication will provide us with strong properties to reason over executions, as seen in the next subsections.

**Example 72.** *Consider the annotated trace* $\mathsf{ta}$ *from Example 58 that* $\mathcal{M}_{\Pi,\overline{id}}$ *can execute. The ground configuration* $\mathcal{K}(\mathsf{ta})$ *can execute it as well, using* $\Rightarrow$*. We now define* $\rho$ *as follows:* $\rho(I(k', n'_I)) = I(k_1, n'_I)$ *and* $\rho(R(k', n'_R)) = R(k_2, n'_R)$ *for some fresh names* $k_1, k_2$*. The trace* $\mathsf{ta}\rho$ *can no longer be executed by* $\mathcal{M}_{\Pi,\overline{id}}$ *nor by* $\mathcal{K}(\mathsf{ta}\rho)$ *(even using* $\Rightarrow$*) because the first output sent by* $R(k_2, n'_R)$ *(*i.e. $\mathsf{senc}(\langle n'_I, n'_R\rangle, k_2)$*) will not be accepted by* $I(k_1, n'_I)$ *since* $k_1 \neq k_2$*.*

### 9.5.2 Control is Determined by Associations

We show in that section that the outcome of tests is entirely determined by associations. This will be useful to show that, if we modify an execution (by renaming agents) while preserving enough associations, then the control flow is left unchanged.

**Proposition 25.** *We assume that* $\Pi$ *satisfies item (i) of the well-authentication condition. Let* $\mathsf{ta} = \mathsf{ta}_0.\tau_x[a_1]$ *(for* $\tau_x \in \{\tau_{\mathsf{then}}, \tau_{\mathsf{else}}\}$*) be a well-formed annotated trace such that*

$$\mathcal{K}(\mathsf{ta}) \xrightarrow{\mathsf{ta}_0.\tau_x[a_1]} (\mathcal{P}; \Phi)$$

*and the last action (*i.e. $\tau_x[a_1]$*) is performed by an unsafe conditional. We have that* $\tau_x = \tau_{\mathsf{then}}$ *if, and only if, there exists* $a_2 \in \mathcal{A}$ *such that* $a_1$ *and* $a_2$ *are associated in* $(\mathsf{ta}_0, \Phi)$*.*

*Proof.* ($\Rightarrow$) We start by applying Proposition 24 to obtain an execution

$$\mathcal{M}_{\Pi,\overline{id}} \xrightarrow{\mathsf{ta}_0.\tau_{\mathsf{then}}[a_1]} (\mathcal{P}'; \Phi) \text{ and thus } \mathcal{M}_{\Pi,\overline{id}} \xrightarrow{\mathsf{ta}_0^*.\tau_{\mathsf{then}}[a_1]} (\mathcal{P}''; \Phi).$$

for some $\mathsf{ta}_0^* \stackrel{\tau}{=} \mathsf{ta}_0$. As a consequence of well-authentication, item (i) applied on the above execution, we obtain that for some $a_2 \in \mathcal{A}$, $a_1$ and $a_2$ are associated in $(\mathsf{ta}_0^*, \Phi)$. Since $\mathsf{ta}_0 \stackrel{\tau}{=} \mathsf{ta}_0^*$, they are also associated in $(\mathsf{ta}_0, \Phi)$.

($\Leftarrow$) For this other direction, we observe that (up to changes of recipes that do not affect the resulting messages) if two agents are associated in the above execution (starting with $\mathcal{K}(\mathsf{ta})$), then they are executing *the* honest trace of $\Pi$ modulo a renaming of parameters, thus the considered test must be successful. We thus assume that $a_1 = A(\overline{k_1}, \overline{n_1})$ and $a_2 = A(\overline{k_2}, \overline{n_2})$ are associated in $(\mathsf{ta}_0, \Phi)$ we shall prove that $\tau_x = \tau_{\mathsf{then}}$. By association, $\mathsf{ta}_0|_{a_1,a_2}$ is honest: its observable actions are of the form $\mathtt{out}(c_1, w_1).\mathtt{in}(c_1', M_1) \ldots \mathtt{out}(c_n, w_n).\mathtt{in}(c_n', M_n)$ with possibly an extra output at the end, and are such that $M_i\Phi \Downarrow_{=\mathsf{E}} w_i\Phi$ for all $1 \leq i \leq n$. Consider $\mathsf{ta}'$ obtained from $\mathsf{ta}_0$ by replacing each recipe $M_i$ by $w_i$. Since this change of recipes does not affect the resulting messages, the modified trace can still be executed by $\mathcal{K}(\mathsf{ta})$ and yields the same configuration $(\mathcal{P}; \Phi)$. But now $\mathsf{ta}'|_{a_1,a_2}$ is a self-contained execution, *i.e.* if $P$ and $Q$ are the processes (possibly sub-processes) respectively annotated $a_1$ and $a_2$ in $\mathcal{K}(\mathsf{ta})$, we have:

$$(\{P[a_1], Q[a_2]\}; \emptyset) \xrightarrow{\mathsf{ta}'|_{a_1,a_2}} (\{P'[a_1], Q'[a_2]\}; \Phi') \xrightarrow{\tau_x[a_1]} (\{P''[a_1], Q'[a_2]\}; \Phi').$$

In the shared case (*i.e.* $fn(\mathcal{I}) \cap fn(\mathcal{R}) \neq \emptyset$), by definition of association, the identity parameters of $a_1$ are equal to those of $a_2$. Otherwise, it holds that $fn(\mathcal{I}) \cap fn(\mathcal{R}) = \emptyset$. In both cases, we thus have:

$$\begin{array}{rcl} (\{\nu\, \overline{k}.(\nu\, \overline{n}_I.\mathcal{I} \mid \nu\, \overline{n}_R.\mathcal{R})\}; \emptyset) & \xrightarrow{\tau^*} & (\{P[a_1], Q[a_2]\}; \emptyset) \\ & \xrightarrow{\mathsf{ta}'|_{a_1,a_2}} & (\{P'[a_1], Q'[a_2]\}; \Phi') \\ & \xrightarrow{\tau_x[a_1]} & (\{P''[a_1], Q'[a_2]\}; \Phi'). \end{array}$$

In that execution, everything is deterministic (up to the equational theory) and thus the execution is actually a prefix of *the* honest execution of $\Pi$ (from the process $P_\Pi$ defined in Definition 51), up to a bijective renaming of parameters (note that $P$ and $Q$ do not share session parameters). Remind that all tests must be positive in the honest execution (*i.e.* $\tau_{\mathsf{else}}$ does not occur in the honest execution). Therefore, $\tau_x = \tau_{\mathsf{then}}$ concluding the proof. □

### 9.5.3 Invariance of Frame Idealisations

In general, a renaming of annotations can break executability: as illustrated in Example 72, mapping two dual annotations to annotations of different identities breaks the ability of the two underlying agents to communicate successfully. Moreover, even when executability is preserved, parameters change (so do names) and thus frames are modified. However, as stated next in Proposition 26, such renaming do not change idealised frames. We obtain the latter since we made sure that idealised frames only depend on what is already observable and not on specific identity or session parameters. In combination with frame opacity, this will imply (Proposition 27) that a renaming of annotations has no observable effect on the resulting *real* frames.

**Proposition 26.** *Let* $\mathsf{ta}$ *be an annotated trace and* $\mathrm{fr}_1$ *be a name assignment such that* $\Phi^{\mathrm{fr}_1}_{\mathrm{ideal}}(\mathsf{ta})$ *is well-defined. Let* $\rho$ *be a renaming of annotations, and* $\mathrm{fr}_2$ *be an injective function satisfying* $\mathrm{fr}_2(a\rho, x) = \mathrm{fr}_1(a, x)$. *We have that* $\Phi^{\mathrm{fr}_2}_{\mathrm{ideal}}(\mathsf{ta})$ *is well-defined and* $\Phi^{\mathrm{fr}_1}_{\mathrm{ideal}}(\mathsf{ta}) = \Phi^{\mathrm{fr}_2}_{\mathrm{ideal}}(\mathsf{ta}\rho)$.

*Proof.* We proceed by induction on $\mathsf{ta}$. The interesting case is when $\mathsf{ta} = \mathsf{ta}_0.\ell : \mathtt{out}(c, w)[a]$ and

$$\Phi^{\mathrm{fr}_1}_{\mathrm{ideal}}(\mathsf{ta}_0.\ell : \mathtt{out}(c, w)[a]) = \Phi^{\mathrm{fr}_1}_{\mathrm{ideal}}(\mathsf{ta}_0) \cup \{w \mapsto \mathrm{ideal}(\ell)\sigma^{\mathsf{i}}_1\sigma^{\mathsf{n}}_1\}$$

with $\sigma^{\mathsf{n}}_1(x^{\mathsf{n}}_j) = \mathrm{fr}_1(a, x^{\mathsf{n}}_j)$ and $\sigma^{\mathsf{i}}_1(x^{\mathsf{i}}_j) = R_j\Phi^{\mathrm{fr}_1}_{\mathrm{ideal}}(\mathsf{ta}_0) \Downarrow$ where $R_j$ is the $j$-th input of $a$ in $\mathsf{ta}_0$.

The idealised frame $\Phi^{\mathrm{fr}_2}_{\mathrm{ideal}}(\mathsf{ta})$ is defined similarly, using $\sigma^{\mathsf{n}}_2$ and $\sigma^{\mathsf{i}}_2$ relying on $\mathsf{ta}_0\rho$ and $\mathrm{fr}_2$ instead of $\mathsf{ta}_0$ and $\mathrm{fr}_1$. By induction hypothesis (*i.e.* $\Phi^{\mathrm{fr}_1}_{\mathrm{ideal}}(\mathsf{ta}_0) = \Phi^{\mathrm{fr}_2}_{\mathrm{ideal}}(\mathsf{ta}_0\rho)$), we only have to establish that $\mathrm{ideal}(\ell)\sigma^{\mathsf{i}}_1\sigma^{\mathsf{n}}_1 = \mathrm{ideal}(\ell)\sigma^{\mathsf{i}}_2\sigma^{\mathsf{n}}_2$ with $\sigma^{\mathsf{n}}_2(x^{\mathsf{n}}_j) = \mathrm{fr}_2(a\rho, x^{\mathsf{n}}_j)$ and $\sigma^{\mathsf{i}}_2(x^{\mathsf{i}}_j) = R^{\rho}_j\Phi^{\mathrm{fr}_2}_{\mathrm{ideal}}(\mathsf{ta}_0\rho) \Downarrow$ where $R^{\rho}_j$ is the $j$-th input of $a\rho$ in $\mathsf{ta}_0\rho$. Since $R^{\rho}_j = R_j$, we have $\sigma^{\mathsf{i}}_1(x^{\mathsf{i}}_j) = \sigma^{\mathsf{i}}_2(x^{\mathsf{i}}_j)$. Moreover, we have that $\sigma^{\mathsf{n}}_2(x^{\mathsf{n}}_j) = \mathrm{fr}_2(a\rho, x^{\mathsf{n}}_j) = \mathrm{fr}_1(a, x^{\mathsf{n}}_j) = \sigma^{\mathsf{n}}_1(x^{\mathsf{n}}_j)$, which allows us to conclude. □

**Proposition 27.** *We assume that* $\Pi$ *satisfies the frame opacity condition. Let* $\rho$ *be a renaming of annotations and* $\mathsf{ta}$ *be a well-formed annotated trace. If* $\mathcal{K}(\mathsf{ta}) \xRightarrow{\mathsf{ta}} (\mathcal{P}_1; \Phi_1)$ *and* $\mathcal{K}(\mathsf{ta}\rho) \xRightarrow{\mathsf{ta}\rho} (\mathcal{P}_2; \Phi_2)$*, then we have that* $\Phi_1 \sim \Phi_2$*.*

*Proof.* We start by applying Proposition 24 on the two given executions to obtain two executions starting with $\mathcal{M}_{\Pi,\overline{id}}$:

$$\mathcal{M}_{\Pi,\overline{id}} \xrightarrow{\mathsf{ta}^*} (\mathcal{P}'_1; \Phi_1) \text{ and } \mathcal{M}_{\Pi,\overline{id}} \xrightarrow{\mathsf{ta}^*_\rho} (\mathcal{P}'_2; \Phi_2)$$

with $\mathsf{ta}^* \overset{\tau}{=} \mathsf{ta}$ and $\mathsf{ta}^*_\rho \overset{\tau}{=} \mathsf{ta}\rho$. Note that, if $\mathsf{ta}_1$ and $\mathsf{ta}_2$ are two annotated traces such that $\mathsf{ta}_1 \overset{\tau}{=} \mathsf{ta}_2$ and $\mathrm{fr}_1$ is a name assignment, then $\Phi^{\mathrm{fr}_1}_{\mathrm{ideal}}(\mathsf{ta}_1) = \Phi^{\mathrm{fr}_1}_{\mathrm{ideal}}(\mathsf{ta}_2)$. Therefore, as a direct consequence of Proposition 26, frame opacity and Proposition 23, we obtain the required conclusion: $\Phi_1 \sim \Phi_2$. □

### 9.5.4 A sufficient Condition for Preserving Executability

We can now state a key lemma (Lemma 23), identifying a class of renamings which yield indistinguishable executions. More precisely, this lemma shows that for renamings satisfying some requirements, if $\mathcal{K}(\mathsf{ta})$ can execute an annotated trace $\mathsf{ta}$ then $\mathcal{K}(\mathsf{ta})\rho$ has an indistinguishable execution following the annotated trace $\mathsf{ta}\rho$. Remark that, in the conclusion, the renaming is applied after building the ground configuration ($\mathcal{K}(\mathsf{ta})\rho$) instead of building the ground configuration of the renamed trace ($\mathcal{K}(\mathsf{ta}\rho)$). Both variants are a priori different. However, in the final proof and in order to leverage previous propositions, we will need to relate executions of $\mathcal{K}(\mathsf{ta})$ with executions of $\mathcal{K}(\mathsf{ta}\rho)$. The following easy proposition bridges this gap. We also state and prove a variant of Proposition 24, item (4). when $\rho$ is applied after building the ground configuration.

**Proposition 28.** *Let* $\mathsf{ta}$ *be a well-formed annotated trace and* $\rho$ *a renaming of annotations. If* $\mathcal{K}(\mathsf{ta})\rho \xRightarrow{\mathsf{ta}\rho} (\mathcal{P}'; \Phi)$ *then* $\mathcal{K}(\mathsf{ta}\rho) \xRightarrow{\mathsf{ta}\rho} (\mathcal{P}''; \Phi)$*.*

*Proof.* Essentially, the proposition follows from the fact that there are less agents in sequence in $\mathcal{K}(\mathsf{ta}\rho)$ than in $\mathcal{K}(\mathsf{ta})\rho$, thanks to the third item of Definition 66.

More formally, by considering $\mathcal{K}(\mathsf{ta}\rho)$ and $\mathcal{K}(\mathsf{ta})\rho$ as multiset of processes without sequence (by removing all sequences and taking the union of processes), we obtain the same multisets.

Next, it suffices to prove that no execution is blocked by a sequence in $\mathcal{K}(\mathsf{ta}\rho)$. By definition of the renaming $\rho$ (third requirement in Definition 66), if an agent $P[\rho(a)]$ occurring in $\mathcal{K}(\mathsf{ta}\rho)$ is in sequence with an agent $\rho(a')$ before him, then $P[a\rho]$ occurring in $(\mathcal{K}(\mathsf{ta})\rho)$ must be in sequence with $a$ before him as well. Hence, when a process $P[\rho(a)];Q$ is available (*i.e.* at top-level) at some point in the execution from $\mathcal{K}(\mathsf{ta})\rho$, then a similar process $P[\rho(a)];Q'$ is also available at the same point in the execution from $K(\mathsf{ta}\rho)$. However, a processes may become available in the execution from $K(\mathsf{ta})\rho$ only after having performed rule Unfold, while the same process may be immediately available in the multiset in the execution from $K(\mathsf{ta}\rho)$. This is why we only obtain a weak execution $\mathcal{K}(\mathsf{ta}\rho) \stackrel{\mathsf{ta}\rho}{\Longrightarrow} (\mathcal{P}'';\Phi)$. □

**Proposition 29.** *If* $\mathsf{ta} = \mathsf{ta}_1.\mathsf{ta}_2$ *is a well-formed annotated trace and* $\mathcal{K}(\mathsf{ta}_1.\mathsf{ta}_2)\rho \xrightarrow{\mathsf{ta}_1\rho} K$ *then* $\mathcal{K}(\mathsf{ta}_1)\rho \xrightarrow{\mathsf{ta}_1\rho} K'$ *with* $\Phi(K) = \Phi(K')$.

*Proof.* The argument is the same as for Proposition 24, item (4): the processes that are added to $\mathcal{K}(\mathsf{ta}_1)\rho$ when considering $\mathcal{K}(\mathsf{ta}_1.\mathsf{ta}_2)\rho$ are unused in the execution of $\mathsf{ta}_1\rho$; moreover, the effect of $\rho$ on the processes of $\mathcal{K}(\mathsf{ta}_1)$ is obviously the same as in $\mathcal{K}(\mathsf{ta}_1.\mathsf{ta}_2)$. □

Finally, after having defined the notion of *connection* between agents, we can state our key lemma.

**Definition 67.** *Annotations* $a$ *and* $a'$ *are* connected *in* $(\mathsf{ta},\Phi)$ *if they are associated in* $(\mathsf{ta}_0,\Phi)$ *for some prefix* $\mathsf{ta}_0$ *of* $\mathsf{ta}$ *that contains at least one* $\tau_{\mathsf{then}}$ *action of an unsafe conditional annotated with either* $a$ *or* $a'$.

**Lemma 23.** *We assume that* $\Pi$ *satisfies frame opacity and item (i) of well-authentication. Let* $\mathsf{ta}$ *be a well-formed annotated trace such that* $\mathcal{K}(\mathsf{ta}) \xrightarrow{\mathsf{ta}} (\mathcal{P};\Phi)$. *Let* $\rho$ *be a renaming of annotations. Moreover, when* $fn(\mathcal{I}) \cap fn(\mathcal{R}) \neq \emptyset$ *(the shared case), we assume that for any annotations* $a$, $a'$, *it holds that* $a$ *and* $a'$ *are connected in* $(\mathsf{ta},\Phi)$, *if, and only if,* $\rho(a)$ *and* $\rho(a')$ *are dual.*

*Under those assumptions, one has* $\mathcal{K}(\mathsf{ta})\rho \xrightarrow{\mathsf{ta}\rho} (\mathcal{Q};\Psi)$ *for some* $\Psi$ *such that* $\Phi \sim \Psi$.

*Proof.* We will proceed by induction on $\mathsf{ta}$ but we need to strengthen the invariants notably by relating the multisets $\mathcal{P}$ and $\mathcal{Q}$: when $\mathcal{P} = \{\coprod_{j\in\mathcal{J}} P_j^i[a_j^i]\}_{i\in\mathcal{I}}$, $\mathcal{Q}$ must be of the form $\mathcal{Q} = \{\coprod_{j\in\mathcal{J}} Q_j^i[\rho(a_j^i)]\}_{i\in\mathcal{I}}$, where $P_j^i$ and $Q_j^i$ are equal up to terms (*i.e.* ignoring terms in outputs and conditionals). We note $\mathcal{P} \mp \mathcal{Q}$ when this is the case.

For any prefix $\mathcal{K}(\mathsf{ta}) \xrightarrow{\mathsf{ta}_0} (\mathcal{P}_0;\Phi_0)$ of the given execution, we prove that there exists an execution $\mathcal{K}(\mathsf{ta})\rho \xrightarrow{\mathsf{ta}_0\rho} (\mathcal{Q}_0;\Psi_0)$ with those invariants:

(a) $\mathcal{P}_0 \mp \mathcal{Q}_0$;

(b) $\Phi_0 \sim \Psi_0$;

($c_1$) when $fn(\mathcal{I}) \cap fn(\mathcal{R}) = \emptyset$ (the non-shared case), $\rho(a)$ and $\rho(a')$ are associated in $(\mathsf{ta}_0\rho,\Psi_0)$ if, and only if, $a$ and $a'$ are associated in $(\mathsf{ta}_0,\Phi)$;

($c_2$) when $fn(\mathcal{I}) \cap fn(\mathcal{R}) \neq \emptyset$ (the shared case), $\rho(a)$ and $\rho(a')$ are associated in $(\mathsf{ta}_0\rho,\Psi_0)$ if, and only if, $a$ and $a'$ are associated in $(\mathsf{ta}_0,\Phi_0)$ and connected in $(\mathsf{ta},\Phi)$.

We proceed by induction on the prefix $\mathsf{ta}_0$ of $\mathsf{ta}$.

If $\mathsf{ta}_0$ is empty, then $\mathsf{ta}_0\rho$ can obviously be executed, and condition (a) holds because $\mathcal{K}(\mathsf{ta}) \mp \mathcal{K}(\mathsf{ta})\rho$ by definition. Condition (b) is trivial since frames are both empty. In order to check conditions ($c_1$) and ($c_2$), note that association coincides with duality for empty traces. Let us verify condition ($c_1$) in the non-shared case. Being dual simply means being distinct roles, hence one obviously has that $\rho(a)$ and $\rho(a')$ are dual if, and only if, $a$ and $a'$ are. Let us verify condition ($c_2$) in the shared case. By hypothesis, we have that $a$ and $a'$ are connected in $(\mathsf{ta}, \Phi)$ if, and only if, $\rho(a)$ and $\rho(a')$ are dual. Furthermore, if $a$ and $a'$ are connected in $(\mathsf{ta}, \Phi)$ then they are dual and therefore they are associated in $(\epsilon, \emptyset)$. Hence ($c_2$).

Otherwise, we are considering a prefix of $\mathsf{ta}$ of the form $\mathsf{ta}_0.\alpha$ where $\alpha$ can be any action. The action $\alpha$ may be an input, an output, a conditional (*i.e.* $\tau_{\mathsf{then}}$ or $\tau_{\mathsf{else}}$) or a $\tau$ action produced by the Unfold rule. By (a), we know that there is a process in $\mathcal{Q}_0$ which is able to perform an action of the same nature. If this action is $\tau$ (excluding the cases of $\tau_{\mathsf{then}}, \tau_{\mathsf{else}}$) then triggering the process in $\mathcal{Q}_0$ producing $\tau$ concludes the proof since all invariants are trivially preserved after this action.

We now have to deal with the case where $\alpha$ is an input, an output or a conditional. In those cases $\alpha$ is necessarily annotated, say by $a$, and has been produced by a process annotated $a$ in $\mathcal{P}_0$. By induction hypothesis we have $(\mathcal{P}_0; \Phi_0)$ and $(\mathcal{Q}_0; \Psi_0)$ and executions

$$\mathcal{K}(\mathsf{ta}) \xrightarrow{\mathsf{ta}_0} (\mathcal{P}_0; \Phi_0) \xrightarrow{\alpha[a]} (\mathcal{P}'_0; \Phi'_0) \text{ and } \mathcal{K}(\mathsf{ta})\rho \xrightarrow{\mathsf{ta}_0\rho} (\mathcal{Q}_0; \Psi_0)$$

satisfying all our invariants. Note that one has $(\mathsf{ta}_0.\alpha[a])\rho = \mathsf{ta}_0\rho.\alpha[\rho(a)]$. Now, we have to prove that exactly the same action $\alpha$ can be executed by the corresponding process annotated $\rho(a)$ in $\mathcal{Q}_0$ given by the invariant $\mathcal{P}_0 \mp \mathcal{Q}_0$ (taking into account recipe for input and failure/success for conditional), and that our invariants are preserved after its execution. We now distinguish the three kinds of actions $\alpha$ can be:

**Case where $\alpha$ is an output.** We immediately have that $\mathcal{Q}_0$ can perform $\alpha[\rho(a)]$, on the same channel and with the same handle. We now have to check our invariants for $\mathsf{ta}_0.\alpha[a]$. Condition (a) is obviously preserved. Conditions ($c_1$) and ($c_2$) follow from the fact that association is not affected by the execution of an output: $\rho(a)$ and $\rho(a')$ are associated in $(\mathsf{ta}_0.\alpha[a])\rho$ if, and only if, they are associated in $\mathsf{ta}_0\rho$, and similarly without $\rho$. Finally, we shall prove (b): $\Phi'_0 \sim \Psi'_0$ where $\Phi'_0$ (resp. $\Psi'_0$) is the resulting frame after the action $\alpha[a]$ (resp. $\alpha[a\rho]$). Applying Proposition 24 item (4) on the execution before renaming and Proposition 29 on the execution after renaming, one obtains

$$\mathcal{K}(\mathsf{ta}_0.\alpha[a]) \xrightarrow{\mathsf{ta}_0.\alpha[a]} (\mathcal{P}''_0; \Phi'_0) \text{ and } \mathcal{K}(\mathsf{ta}_0.\alpha[a])\rho \xrightarrow{\mathsf{ta}_0\rho.\alpha[a\rho]} (\mathcal{Q}'_0; \Psi'_0).$$

We finally conclude $\Phi'_0 \sim \Psi'_0$ using Proposition 28 on the execution on the right and then Proposition 27.

**Case where $\alpha$ is a conditional.** We first need to make sure that the outcome of the test is the same for $a$ and $a\rho$. We let $\tau_x$ (resp. $\tau_y$) be the action produced by evaluating the conditional

of $a$ (resp. $a\rho$) and shall prove that $\tau_x = \tau_y$. We distinguish two cases, whether the conditional is safe or not.

- If the conditional is safe, then its outcome only depends on the inputs and outputs of $a$ that are statically equivalent to those of $\rho(a)$ by the invariant (b). Hence, the outcome of that test is the same for $a$ and $\rho(a)$ and $\tau_x = \tau_y$.
- If the conditional is unsafe, we make use of Proposition 25 to show that the outcome of the conditional is the same on both sides. First, we deduce the following executions from Proposition 24 item (4) applied on the execution before renaming and Proposition 29 applied on the execution after renaming:

$$\mathcal{K}(\mathsf{ta}_0.\tau_x[a]) \xrightarrow{\mathsf{ta}_0.\tau_x[a]} (\mathcal{P}_0'; \Phi_0) \text{ and } \mathcal{K}(\mathsf{ta}_0.\tau_y[a])\rho \xrightarrow{\mathsf{ta}_0\rho.\tau_y[a\rho]} (\mathcal{Q}_0'; \Psi_0).$$

  To infer $\tau_x = \tau_y$ from Proposition 25, it remains to prove that $a$ and $a'$ are associated in $(\mathsf{ta}_0, \Phi_0)$ if, and only if, $\rho(a)$ and $\rho(a')$ are associated in $(\mathsf{ta}_0\rho, \Psi_0)$. When not in the shared case, this is given by the invariant ($c_1$). Otherwise, ($c_2$) gives us that $a$ and $a'$ are associated when $\rho(a)$ and $\rho(a')$ are associated. Conversely, if $a$ and $a'$ are associated in $(\mathsf{ta}_0, \Phi)$, then by Proposition 25, the outcome of the test will be positive (*i.e.* $\tau_x = \tau_{\mathsf{then}}$) and $a$ and $a'$ are thus connected in $(\mathsf{ta}_0, \Phi)$. Therefore, $\rho(a)$ and $\rho(a')$ are associated in $(\mathsf{ta}_0\rho, \Psi_0)$ by ($c_2$).

After the execution of this conditional producing $\tau_x = \tau_y$, condition (a) obviously still holds since $\tau_x = \tau_y$ implying that both agents go to the same branch of the conditional. Invariant (b) is trivial since frames have not changed. Conditions ($c_1$) and ($c_2$) are preserved because the association between $a$ and $a'$ is preserved if, and only if, the outcome of the test is positive, which is the same before and after the renaming.

**Case where $\alpha$ is an input.** We immediately have that $\mathcal{Q}_0$ can perform $\alpha[\rho(a)]$ on the same channel and with the same recipe (since $\mathrm{dom}(\Phi_0) = \mathrm{dom}(\Psi_0)$ follows from $\Phi_0 \sim \Psi_0$). Conditions (a) and (b) are obviously preserved. Conditions ($c_1$) and ($c_2$) are preserved because honest interactions are preserved by the renaming, since $\Phi_0 \sim \Psi_0$ by invariant (b). We only detail one direction of ($c_1$), the other cases being similar. Assume that $\rho(a)$ and $\rho(a')$ are associated in $((\mathsf{ta}_0.\alpha[a])\rho, \Psi_0)$. The renamed agents $\rho(a)$ and $\rho(a')$ are also associated in $(\mathsf{ta}_0\rho, \Phi')$, thus $a$ and $a'$ are associated in $(\mathsf{ta}_0, \Phi')$. Now, because $\alpha$ did not break the association of $\rho(a)$ and $\rho(a')$ in $(\mathsf{ta}_0\rho, \Psi_0)$, it must be that the input message in $\alpha = \mathtt{in}(c, M)$ corresponds to the last output of $\rho(a')$ in $\mathsf{ta}_0\rho$. Formally, if that last output corresponds to the handle $w$ in $\Psi_0$, we have $M\Psi_0 \Downarrow_{=\mathsf{E}} w\Psi_0$. But, because $\Phi_0 \sim \Phi_0$ by invariant (b), we then also have $M\Phi_0 \Downarrow_{=\mathsf{E}} w\Phi_0$ and the association of $a$ and $a'$ in $(\mathsf{ta}_0, \Phi_0)$ carries over to $(\mathsf{ta}_0.\alpha[a], \Phi_0)$. □

### 9.5.5 Final Proof

Thanks to Lemma 23, we can transform any execution of $\mathcal{M}_{\Pi,\overline{id}}$ into an indistinguishable execution of $\mathcal{S}_\Pi$, provided that an appropriate renaming of annotations exists. In order to prove that

such a renaming exists in Proposition 31, we show below that in the shared case, there cannot be agents connected multiple times.

**Proposition 30.** *Assume that* $\Pi$ *satisfies item (ii) of the well-authentication condition and that* $fn(\mathcal{I}) \cap fn(\mathcal{R}) \neq \emptyset$ *(shared case). Consider a well-formed annotated trace* $\mathsf{ta}$ *and an execution*

$$\mathcal{K}(\mathsf{ta}) \xrightarrow{\mathsf{ta}} (\mathcal{P}; \Phi).$$

*If there are three annotations* $a, a_1, a_2$ *such that* $a$ *and* $a_1$ *are connected in* $(\mathsf{ta}, \Phi)$ *and* $a$ *and* $a_2$ *are connected in* $(\mathsf{ta}, \Phi)$ *then* $a_1 = a_2$.

*Proof.* Consider the first unsafe conditional performed by one of the three agents $a, a_1, a_2$. We claim that when this action is performed, $a$ and $a_1$ are associated, and so are $a$ and $a_2$, which contradicts condition (ii) of well-authentication. Indeed, both pairs $(a, a_1)$ and $(a, a_2)$ are connected in $(\mathsf{ta}, \Phi)$ and are thus associated until the first unsafe test of the corresponding pairs. □

**Proposition 31.** *We assume that* $\Pi$ *satisfies item (ii) of well-authentication. For any well-formed annotated trace* $\mathsf{ta}$ *such that*

$$\mathcal{K}(\mathsf{ta}) \xrightarrow{\mathsf{ta}} K,$$

*there exists a renaming of annotations* $\rho$ *satisfying the hypothesis of Lemma 23 and such that* $\mathsf{ta}\rho$ *is single-session.*

*Proof.* For $\overline{k} \in \mathrm{id}_I(\mathsf{ta}) \cup \mathrm{id}_R(\mathsf{ta})$, we define $\mathrm{Co}(\overline{k})$ as follows:

- if in the shared case (*i.e.* $fn(\mathcal{I}) \cap fn(\mathcal{R}) \neq \emptyset$), we let $\mathrm{Co}(\overline{k})$ be the set of all $(\overline{n_1}, \overline{n_2})$ such that $I(\overline{k}, \overline{n_1})$ and $R(\overline{k}, \overline{n_2})$ are connected in $(\mathsf{ta}, \Phi(K))$;
- otherwise (*i.e.* $fn(\mathcal{I}) \cap fn(\mathcal{R}) = \emptyset$), we let $\mathrm{Co}(\overline{k})$ be the empty set.

Essentially, $\mathrm{Co}(\overline{k})$ denotes the set of pairs of (dual) sessions that the renaming to be defined should keep on the same identity. Applying Proposition 30, we deduce that for any $\overline{k} \in \mathrm{id}_A(\mathsf{ta})$ and $(\overline{n_1}, \overline{n_2}), (\overline{n_3}, \overline{n_4}) \in \mathrm{Co}(\overline{k})$, then either (i) $\overline{n_1} = \overline{n_3}$ and $\overline{n_2} = \overline{n_4}$ or (ii) $\overline{n_1} \neq \overline{n_3}$ and $\overline{n_2} \neq \overline{n_4}$.

Next, we assume the existence of a function $k^c : \mathcal{N}^* \times \mathcal{N}^* \times^* \mathcal{N}^* \mapsto \mathcal{N}^*$ that associates to any sequence of names $(\overline{k}, \overline{n_1}, \overline{n_2})$ a vector of names of the length of identity parameters of $\Pi$: $\overline{k'} = k^c(\overline{k}, \overline{n_1}, \overline{n_2})$. These name vectors are assumed to be all disjoint and not containing any name already occurring in the annotations of $\mathsf{ta}$. This gives us a mean to pick fresh identity parameters for each combination of $\overline{k}, \overline{n_1}, \overline{n_2}$ taken from the annotations of $\mathsf{ta}$. We also assume a function $k^1$ such that the vectors $k^1(\overline{k}, \overline{n_1})$ are again disjoint and not overlapping with annotations of $\mathsf{ta}$ and any $k^c(\overline{k'}, \overline{n_1'}, \overline{n_2'})$, and similarly for $k^2(\overline{k}, \overline{n_2})$ which should also not overlap with $k^1$ vectors. These last two collections of identity parameters will be used to give fresh identities to initiator and responder agents, independently. We then define $\rho$ as follows:

$$\begin{array}{rcll} I(\overline{k}, \overline{n_1}) & \mapsto & I(k^c(\overline{k}, \overline{n_1}, \overline{n_2}), \overline{n_1}) & \text{if } (\overline{n_1}, \overline{n_2}) \in \mathrm{Co}(\overline{k}) \\ & \mapsto & I(k^1(\overline{k}, \overline{n_1}), \overline{n_1}) & \text{otherwise} \\ R(\overline{k}, \overline{n_2}) & \mapsto & R(k^c(\overline{k}, \overline{n_1}, \overline{n_2}), \overline{n_2}) & \text{if } (\overline{n_1}, \overline{n_2}) \in \mathrm{Co}(\overline{k}) \\ & \mapsto & R(k^2(\overline{k}, \overline{n_2}), \overline{n_2}) & \text{otherwise} \end{array}$$

Let us prove that $\rho$ satisfies all requirements.

*The mapping $\rho$ is a renaming.* First, for any well-formed $\mathsf{ta}'$, the fact that $\mathsf{ta}'\rho$ is well-formed follows from the following: (i) session names are not modified and (ii) identity names are all pairwise distinct and never intersect except for agents $\rho(I(\overline{k}, \overline{n_1}))$ and $\rho(R(\overline{k}, \overline{n_2}))$ such that $(\overline{n_1}, \overline{n_2}) \in \mathrm{Co}(\overline{k})$ but there cannot be a third agent sharing the same identity names according to the result obtained above. The mapping $\rho$ is obviously role-preserving. Finally, if $\rho(a)$ and $\rho(a')$ share the same identity parameters then (a) $fn(\mathcal{I}) \cap fn(\mathcal{R}) \neq \emptyset$ and (b) $a$ and $a'$ are connected in $(\mathsf{ta}, \Phi)$ and are thus dual implying that $a$ and $a'$ share the same identity parameters as well.

*The renaming $\rho$ is single-session* since $\mathsf{id}_0$ never occurs in the image of $\rho$ and all agents are mapped to agents having fresh, distinct identity parameters except for agents $a, a'$ that were connected in $(\mathsf{ta}, \Phi)$. But as already discussed, in such a case, there is no third agent sharing those identity parameters and $a$ and $a'$ are necessarily dual.

*Hypothesis of Lemma 23.* To conclude, we shall prove that, in the shared case, for any annotations $a$, $a'$, it holds that $a$ and $a'$ are connected in $(\mathsf{ta}, \Phi)$, if, and only if, $\rho(a)$ and $\rho(a')$ are dual. The $\Leftarrow$ direction has already been proved above. The $\Rightarrow$ direction follows from the fact that, in such a case, the session parameters of $a$ and $a'$ would be in $\mathrm{Co}(\overline{k})$. □

We conclude the section with the final proof of Theorem 8.

*Proof of Theorem 8.* It is easy to see that $\mathcal{S}_\Pi \sqsubseteq \mathcal{M}_\Pi \sqsubseteq \mathcal{M}_{\Pi,\overline{id}}$, so it only remains to establish that $\mathcal{M}_{\Pi,\overline{id}} \sqsubseteq \mathcal{S}_\Pi$. Consider an execution $\mathcal{M}_{\Pi,\overline{id}} \xrightarrow{\mathsf{ta}} (\mathcal{P}; \Phi)$. Proposition 24 yields an annotated trace $\mathsf{ta}' \stackrel{\tau}{=} \mathsf{ta}$ and an execution $\mathcal{K}(\mathsf{ta}) \xrightarrow{\mathsf{ta}'} (\mathcal{P}', \Phi)$.

Let $\rho$ be the renaming obtained in Proposition 31 for $\mathsf{ta}'$. By Lemma 23, $\mathsf{ta}'\rho$ remains executable and is indistinguishable from $\mathsf{ta}'$:

$$\mathcal{K}(\mathsf{ta})\rho \xrightarrow{\mathsf{ta}'\rho} (\mathcal{Q}; \Phi_\rho) \text{ with } \Phi_\rho \sim \Phi.$$

We then deduce from Proposition 28 an execution

$$\mathcal{K}(\mathsf{ta}\rho) \xRightarrow{\mathsf{ta}'\rho} (\mathcal{Q}'; \Phi_\rho).$$

Since $\mathsf{ta}\rho$ is single-session, Proposition 24 implies:

$$(\mathcal{S}_\Pi; \emptyset) \xrightarrow{\mathsf{ta}^*} (\mathcal{Q}''; \Phi_\rho) \text{ with } \mathsf{ta}^* \stackrel{\tau}{=} \mathsf{ta}'\rho.$$

This execution allows us to conclude: it has the same observable actions as $\mathsf{ta}$ since $\mathsf{obs}(\mathsf{ta}) = \mathsf{obs}(\mathsf{ta}') = \mathsf{obs}(\mathsf{ta}'\rho) = \mathsf{obs}(\mathsf{ta}^*)$, and yields a statically equivalent frame (*i.e.* $\Phi_\rho \sim \Phi$). □

*Chapter 10*

# Mechanisation & Case Studies

In this chapter, we show why our Main Theorem and our conditions are interesting in practice. We first explain how to verify each of our conditions using dedicated encodings in Section 10.1. For that, we rely on the fact that each condition is fundamentally simpler that the original properties of interest (*i.e.* unlinkability and anonymity) and only focuses on one aspect of the underlying complex verification problem. For instance, one can get rid of control-flow issues when verifying frame-opacity and one can verify well-authentication using only reachability properties instead of more complex equivalence properties. Next, we describe our tool UKano that automates all those encodings.

In Section 10.2 with consider several case studies and we show that our method allows to establish new proofs or to find new attacks on real-world security protocols.

## 10.1 Mechanisation

We now discuss how to verify unlinkability and anonymity in practice, through the verification of our two conditions. More specifically, we describe how appropriate encodings allow one to verify frame opacity (Subsection 10.1.1) and well-authentication (Subsection 10.1.2), respectively through diff-equivalence and correspondence properties. These can then be verified by several tools such as ProVerif and Tamarin.

We additionally provide a tool, called UKano [UKA] (10.1.3), which mechanises the encodings described in this section. Our tool takes as input a specification of a protocol in our class and, computes encodings, and calls ProVerif to automatically check our two conditions, and thus unlinkability and anonymity. As we shall see in Section 10.2, our tool concludes on many interesting case studies.

Due to our choice of base tool, our presentation is sometimes biased towards ProVerif. We note, however, that the encodings are quite generic, and may be used to verify our properties through other tools, automatic or not, *e.g.* Tamarin. This may be useful to bypass some of ProVerif's limitations w.r.t. the supported cryptographic primitives or sequential sessions (which cannot be modelled and verified with enough precision).

### 10.1.1 Frame Opacity

We first explain how to check frame opacity using the diff-equivalence feature of ProVerif [BAF08]. Intuitively, diff-equivalence is obtained from trace equivalence by forcing the two processes (or configurations) being compared to follow the same execution. More formally, diff-equivalence is a property of *bi-processes*, which are processes (or configurations) in which some terms are replaced by *bi-terms* of the form $\mathsf{choice}[u_1, u_2]$. Intuitively, a bi-process represents two processes: the first (resp. second) process is obtained by selecting the first (resp. second) component of choice operators. Said otherwise, each execution of a bi-process produces two frames instead of one. More importantly, when bi-processes are executed, exactly the same rule should be applied to the two processes on both sides. A bi-process is *diff-equivalent* if, when executing it, *(i)* both sides can always be executed synchronously (*i.e.* it never happens that a rule can be applied on one side but not on the other side) and *(ii)* the two resulting frames are statically equivalent. We say that two regular multisets of processes are *diff-equivalent* when they can be combined into a *diff-equivalent* bi-process.

Our notion of frame opacity requires that, for any execution $\mathcal{M}_{\Pi,\overline{id}} \xrightarrow{\mathsf{ta}} (P;\Phi)$, one has $\Phi \approx \Phi_{\mathrm{ideal}}(\mathsf{ta})$. It is possible to verify this condition by checking the diff-equivalence between $\mathcal{M}_{\Pi,\overline{id}}$ and a modified version of this process where each output $u$ (identified by a label in $\mathcal{L}$) has been replaced by a *static idealisation* induced by the idealisation operator ideal$(\cdot)$. As a first approximation, we consider the bi-process $\mathrm{biproc}(\mathcal{M}_{\Pi,\overline{id}})$ obtained from $\mathcal{M}_{\Pi,\overline{id}}$ by replacing each $\ell : \mathtt{out}(c, u_\ell)$ by $\ell : \mathtt{out}(c, \mathsf{choice}[u_\ell, \mathrm{ideal}(\ell)])$ and binding $\mathcal{X}^{\mathsf{n}}$ variables used in ideal$(\ell)$ by '$\nu$' constructs at the beginning of each role instance, so that they will be generated fresh for each session. We assume here that variables $x^{\mathsf{i}} \in \mathcal{X}^{\mathsf{i}}$ actually correspond to the appropriate input variables; this can be obtained easily by renaming.

The issue with this first bi-process is that diff-equivalence in ProVerif forces the left-hand and right-hand processes to execute exactly the same kind of actions at the same time, even for unobservable actions such as $\tau_{\mathsf{then}}, \tau_{\mathsf{else}}$. This is too constraining: typically, a normal execution involving a ciphered message may be mapped to an idealised execution where that message is replaced by a nonce, hence it will not pass tests such as the ability to be decrypted. However, recall that our only goal here is to make sure that, if a frame $\Phi$ is reachable from $\mathcal{M}_{\Pi,\overline{id}}$ then it is statically equivalent to its idealisation. We are not really interested in executability of the process on the right as we only use it to produce idealised frame corresponding to the *real* frame produced by the process on the left. We overcome this difficulty in the actual definition of $\mathrm{biproc}(\mathcal{M}_{\Pi,\overline{id}})$ by pushing conditionals into messages and putting else branches in parallel. We do not formally explain how to do so as it heavily depends on specificities of ProVerif, but just give an example to illustrate the pushing of conditionals: we show in Figure 10.1 (part of) the bi-process resulting from the application of our transformation to our running example; note that the computation of `merge` never fails (neither on the left, nor on the right) because `catchFail` always returns a message. More examples can be found in the ProVerif files produced by UKano for our case studies, available online [UKe].

Assuming that diff-equivalence holds for $\mathrm{biproc}(\mathcal{M}_{\Pi,\overline{id}})$, we have that frame opacity holds

```
! new k;
! new nI; new nR; new n1; new n2; new n3;
 (    (* Initiator role: *)
  out(cI,choice[nI, n1]);
  in(cI,x);
  let merge = choice[
    let catchFail =
      let yt,ynR = eq(proj1(dec(x,k)),nI),
                   proj2(dec(x,k)) in
        enc((ynR,nI),k)
    else n2,
    n2] in
  out(cI,merge))
|(    (*  Responder role: *)
  in(cR,z);
  out(cR,choice[enc((nI,nR),k), n3]);  ...)
```

**Figure 10.1** Example of ProVerif file checking frame opacity (Feldhofer)

too. Indeed, for any execution $\mathcal{M}_{\Pi,\overline{id}} \xrightarrow{\mathsf{ta}} (P;\Phi)$, there exists some execution $\mathrm{biproc}(\mathcal{M}_{\Pi,\overline{id}}) \xrightarrow{\mathsf{ta}} (Q, \mathsf{choice}[\Phi,\Phi_r])$ with $\Phi \approx \Phi_r$. By construction of the bi-process we have that $\Phi_r \approx \Phi_{\mathrm{ideal}}(\mathsf{ta})$, thus frame opacity holds.

**Practical application.** Our tool UKano does not require the user to input the idealisation function. Instead, a default idealisation is extracted from the protocol's outputs. The user is informed about this idealisation, and if he wants to, he can bypass it using annotations. In practice, this is rarely necessary. Moreover, UKano lets the user choose among different heuristics to build idealisation operators. We briefly explain and compare them in Subsection 10.1.3.

In case of protocols with sequential sessions, we could check frame opacity with concurrent sessions, which implies the property for the concurrent case.

We note, however, that frame opacity may be checked using more lightweight verification techniques than diff-equivalence. At least in case of standard cryptographic primitives (pairs, symmetric and asymmetric encryption, hash, MACs and signatures), we conjecture that frame opacity can be reduced to secrecy and freshness conditions; such reachability properties can be handled efficiently by several tools. For such cases (*i.e.* uses of standard primitives), we thus claim that our methods allows to completely reduce the verification of unlinkability and anonymity into purely reachability properties verification.

### 10.1.2 Well-authentication

We explain below how to check condition *(i)* of well-authentication. Once that condition is established, together with frame opacity, we shall see that condition *(ii)* is a consequence of a simple assumption on the choice of idealisation, which is always guaranteed when using UKano.

### Condition (i)

Condition *(i)* of well-authentication is basically a conjunction of reachability properties, which can be checked in ProVerif using correspondence properties [AB03]. For each role $\mathtt{A} \in \{\mathcal{I}, \mathcal{R}\}$, we associate to each syntactical output (resp. input) of the role an event which uniquely identifies the action. More formally, we use events of the form $\mathtt{OutA_i}(\overline{k}, \overline{n}, m)$ and $\mathtt{InA_j}(\overline{k}, \overline{n}, m)$, whose arguments contain:

- identity parameters $\overline{k}$ and session parameters $\overline{n}$;
- the message $m$ that is inputted or outputted.

In the same fashion, we also add events of the form $\mathtt{TestA_k}(\overline{k}, \overline{n})$ at the beginning of each `then` branch.

For each conditional of the protocol, we first check if the simple syntactical definition of *safe* conditionals holds (see Definition 61). If it is the case we do nothing for this conditional. Otherwise, we need to check condition (i) of well-authentication. It can be expressed as a correspondence property using events as explained next. Given a role $\mathtt{A} \in \{\mathcal{I}, \mathcal{R}\}$ and a conditional of this role whose event is $\mathtt{TestA_i}(\overline{k}, \overline{n})$, we express as a correspondence property the fact that if the conditional is positively evaluated, then the involved agent must be associated to a dual agent:

1. when the event $\mathtt{TestA_i}(\overline{k}, \overline{n})$ is fired,
2. there must be a previous event $\mathtt{InA_j}(\overline{k}, \overline{n}, m)$ ($\mathtt{InA_j}$ corresponding to the input just before the conditional),
3. and a previous event $\mathtt{OutB_k}(\overline{k}', \overline{n}', m)$ ($\mathtt{OutB_k}$ corresponding to the output that fed the input $\mathtt{InA_j}$ in the honest execution),
4. and a previous event $\mathtt{InB_1}(\overline{k}', \overline{n}', m')$ ($\mathtt{InB_1}$ corresponding to the first input before $\mathtt{OutB_k}$),
5. and a previous event $\mathtt{OutA_m}(\overline{k}', \overline{n}, m')$ ($\mathtt{OutA_m}$ corresponding to the output that fed the input $\mathtt{InB_1}$ in the honest execution), etc.

Moreover, when $fn(\mathcal{I}) \cap fn(\mathcal{R}) \neq \emptyset$ (shared case), to reflect that duality also requires that identity parameters should be the same, we replace $\overline{k}'$ by $\overline{k}$. Note that by using the same messages ($m$, $m'$, etc.) for corresponding inputs and outputs, we express that the messages that are outputted and inputted are equal modulo the equational theory E.

We show in Figure 10.2 the ProVerif query we produce for checking condition (i) on the first conditional of $P_I$ from our running example.

**Practical application.** In our tool, safe conditionals are not (yet) automatically identified. Actually, the tool lists all conditionals and tells which ones satisfy condition (i) of well-authentication. The user can thus easily get rid of the conditionals that he identifies as safe. Furthermore, the

```
query k:key, n1:bitstring, n2:bitstring,
      nt:bitstring, nr:bitstring,
      mP:bitstring, mR:bitstring;
      event(TestI1(k,n1)) ==>
        (event(InI1(k,n1,mR)) ==>
          (event(OutR1(k,n2,mR)) ==>
            (event(InR1(k,n2,mP)) ==>
              (event(OutI1(k,n1,mP)))
        ) ) ).
```

**Figure 10.2** Example of ProVerif query for checking condition (i)

structure of the ProVerif file produced by UKano makes it easy for the user to remove the proof obligations corresponding to safe conditionals.

Note that, for some examples, we also verified condition (i) of well-authentication using Tamarin by encoding the queries described above as simple lemmas (models are available at [UKe]). In our case, one of the most important advantage of Tamarin over ProVerif is its capability to model ! and thus protocols for which a role executes its sessions in sequence. Relying on Tamarin, we were thus able to verify condition (i) for protocols that ensure unlinkability when sessions are running sequentially but not when they are running concurrently (*e.g.* we automatically verified the variant of Feldhofer described in Example 52).

### Condition (ii)

We shall prove that, for the shared case, once condition (i) of well-authentication is known to hold, condition (ii) is a consequence of two simpler conditions. First, the first conditional of the responder role should be safe — remark that if this does not hold, similar attacks as the one discussed in Example 70 may break unlinkability. Second, messages outputted in honest interactions by different agents should always be different.

**Lemma 24.** *Let* $\Pi = (\overline{k}, \overline{n}_I, \overline{n}_R, \dagger_I, \dagger_R, \mathcal{I}, \mathcal{R})$ *be a protocol such that* $fn(\mathcal{I}) \cap fn(\mathcal{R}) \neq \emptyset$ *(shared case) that satisfies condition (i) of well-authentication. Condition (ii) of well-authentication holds provided that:*

*(a) the first conditional that occurs in* $\mathcal{R}$ *is safe;*

*(b) for any execution* $\mathsf{ta}$ *of* $(\mathcal{M}_{\Pi,\overline{id}}; \emptyset)$*, if* $\mathsf{ta}_1 = \mathsf{ta}|_{a_1,b_1}$ *and* $\mathsf{ta}_2 = \mathsf{ta}|_{a_2,b_2}$ *are honest with* $a_1 \neq a_2$*, then any message outputted by* $a_1$ *in* $\mathsf{ta}_1$ *is different (modulo* $\mathsf{E}$*) from any message outputted by* $a_2$ *in* $\mathsf{ta}_2$*.*

*Proof.* Consider an execution $\mathcal{M}_{\Pi,\overline{id}} \xrightarrow{\mathsf{ta}.\tau_{\mathsf{then}}[a']} (\mathcal{P}; \Phi)$ where two agents $a$ and $a'$ are associated and $a'$ has performed the last $\tau_{\mathsf{then}}$. If this test corresponds to a safe conditional, there is nothing to prove. Otherwise, we shall prove that $a$ is only associated to $a'$, and vice versa.

**Agent $a'$ is only associated to $a$.** Consider the last input of $a'$ and the last output of $a$:

$$\mathcal{M}_{\Pi,\overline{id}} \xrightarrow{\mathsf{ta}.\mathtt{out}(c,w_\ell)[a].\mathsf{ta}'.\mathtt{in}(c',R)[a'].\mathsf{ta}''.\tau_{\mathsf{then}}[a']} (\mathcal{P};\Phi)$$

We have $R\Phi \Downarrow \Phi(w_\ell)$. Assume, for the sake of contradiction, that $a'$ is associated to another agent $b \neq a$. Then we have $R\Phi \Downarrow_{=_{\mathsf{E}}} \Phi(w_{\ell'})$, hence $\Phi(w_\ell) =_{\mathsf{E}} \Phi(w_{\ell'})$, for a handle $w_{\ell'}$ corresponding to some output of $b$ in the honest trace $\mathsf{ta}|_{a',b}$. This contradicts assumption (b).

**Agent $a$ is only associated to $a'$.** Agent $a$ must have performed an input in $\mathsf{ta}$: this is obvious if $a$ is a responder, and follows from assumption (a) otherwise. Let $\ell$ be the label of the previous output of $a'$ and $w_\ell$ be the corresponding handle in the frame $\Phi$. Our execution is thus of the following form:

$$\mathcal{M}_{\Pi,\overline{id}} \xrightarrow{\mathsf{ta}.\mathtt{out}(c,w_\ell)[a'].\mathsf{ta}'.\mathtt{in}(c,R)[a].\mathsf{ta}''.\tau_{\mathsf{then}}[a']} (\mathcal{P};\Phi)$$

We know that the message $m$, satisfying $R\Phi \Downarrow m$, which is inputted by $a$ is equal (modulo $\mathsf{E}$) to the previous output of $a'$, that is $\Phi(w_\ell)$. As before, condition (b) implies that it cannot be equal to the output of another agent having an honest interaction in $\mathsf{ta}$, thus $a$ is only associated to $a'$. □

**Practical application.** Condition (a) of Lemma 24 is usually easily checked manually; UKano leaves it to the user. Condition (b) may in general be very difficult to verify. In practice, once frame opacity is known to hold, condition (b) actually follows immediately from simple properties of the idealisation function, since checking that honest outputs cannot be confused in executions of $\mathcal{M}_{\Pi,\overline{id}}$ is equivalent to checking that they cannot be confused in idealised executions. Often, the idealisation function uses only function symbols that do not occur in $\mathsf{E}$ and such that at least one session variable $x^{\mathsf{n}} \in \mathcal{X}^{\mathsf{n}}$ occurs in ideal($\ell$) for each honest output label $\ell$. Checking that the idealisation function enjoys these properties is straightforward. Let us now show that it implies condition (b) of Lemma 24.

**Proposition 32.** *Let $\Pi = (\overline{k}, \overline{n}_I, \overline{n}_R, \dagger_I, \dagger_R, \mathcal{I}, \mathcal{R})$ be a protocol such that $fn(\mathcal{I}) \cap fn(\mathcal{R}) \neq \emptyset$ (shared case). Consider an idealised operator* ideal($\cdot$) *such that, for any label $\ell \in \mathcal{L}$ occurring in the honest execution of $\Pi$, the following holds:*

- ideal($\ell$) $\in \mathcal{T}(\Sigma_c^s, \mathcal{X}^{\mathsf{i}} \cup \mathcal{X}^{\mathsf{n}})$ *where $\Sigma_c^s \subseteq \Sigma_c$ contains function symbols that never occur in equations of $\mathsf{E}$ and*
- vars(ideal($\ell$)) $\cap\, \mathcal{X}^{\mathsf{n}} \neq \emptyset$.

*If $\Pi$ satisfies frame opacity for the idealised operator* ideal($\cdot$) *then condition (b) of Lemma 24 holds.*

*Proof.* Consider an execution $\mathsf{ta}$ of $\mathcal{M}_{\Pi,\overline{id}}$ where agent $a_1$ performs an output with label $\ell$ and handle $w$, and agent $a_2 \neq a_1$ performs another output with label $\ell'$ and handle $w'$. We assume that $\ell$ occurs in the honest execution of $\Pi$ and we note $\Phi$ the resulting frame from the above

execution. Assume, for the sake of contradiction, that $\Phi(w) =_{\mathsf{E}} \Phi(w')$. Since the protocol ensures frame opacity for the idealised operator ideal(·), we deduce that:

$$\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta})(w) =_{\mathsf{E}} \Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta})(w').$$

By hypothesis on ideal($\ell$), it holds that $\mathrm{vars}(\mathrm{ideal}(\ell)) \cap \mathcal{X}^{\mathsf{n}} \neq \emptyset$. Therefore, some name $\mathrm{fr}(a_1, x^{\mathsf{n}})$ occurs in $\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta})(w)$ at some position, under non-malleable function symbols only (*i.e.* function symbols in $\Sigma^s_c$ that do not occur in equations in $\mathsf{E}$). Thus, the same function symbols must occur along the path to that position in $\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta})(w')$, and $\mathrm{fr}(a_2, x^{\mathsf{n}}) =_{\mathsf{E}} t$ for some sub-term $t$ of $\Phi^{\mathrm{fr}}_{\mathrm{ideal}}(\mathsf{ta})(w')$. Since the name $\mathrm{fr}(a_1, x^{\mathsf{n}})$ does not occur in $t$ by construction, this implies that the equational theory is degenerate (all terms can be equated) which we have ruled out by assumption (see Subsection 2.1.1 in Chapter 2). □

### 10.1.3 The Tool UKano

As mentioned earlier, the tool UKano [UKA] automatises the encodings described in the previous section. It takes as input a ProVerif model specifying the protocol to be verified (and the identity names $\overline{id}$) and returns:

1. whether frame opacity could be established or not: in particular, it infers an idealisation operator that, when in the shared case, satisfies the assumptions of Proposition 32 discussed in Subsection 10.1.2;

2. and the list of conditionals for which condition (i) of well-authentication holds.

If frame opacity holds and condition (i) of well-authentication holds for all conditionals — possibly with some exceptions for conditionals the user can identify as safe — then the tool concludes that that the protocol given as input ensures unlinkability and anonymity w.r.t. $\overline{id}$. Note that the tool detects whether $fn(\mathcal{I}) \cap fn(\mathcal{R}) = \emptyset$ or not and adapts the queries for verifying item (i) of well-authentication accordingly.

Our tool uses heuristics to build idealised operators that always satisfy requirements discussed in Subsection 10.1.2 (*i.e.* assumptions of Proposition 32). Three different heuristics for automatically building idealisation operators have been implemented and can be used using different options. The default heuristic follows the canonical syntactical construction described in Section 9.2 except that sub-terms having a function symbol at top-level that is involved in the equational theory will be replaced by a fresh session name in order to comply with hypothesis of Proposition 32. However, when using the option `-ideal-full-syntax` in the non-shared case, the tool fully adopts the canonical syntactical construction (and displays a warning message when in the shared case since all requirements are not met in this case). Further, the option `-ideal-greedy` can be used to modify the heuristics as follows:

- idealisation of a tuple is a tuple of idealisations of the corresponding sub-terms and

- idealisation of any other term is a fresh session variable in $\mathcal{X}^{\mathsf{n}}$.

Such an idealised operator is much less precise (*i.e.* it often leads to false negatives) but since idealised messages are much simpler, it allows better performance when it works. Finally, the user can also define its own idealisations. The tool UKano always checks their conformity though (*i.e.* assumptions of Proposition 32 when in the shared case).

At a technical level, we built UKano on top of ProVerif. We only re-used the lexer, parser and AST of ProVerif and build upon those a generator and translator of ProVerif models implementing our sufficient conditions via the above encodings. This effort represents about 2k OCaml LoC. The official page of the tool UKano can be found at `http://projects.lsv.ens-cachan.fr/ukano/`, the sources at `http://github.com/LCBH/UKano`, and the manual at `http://github.com/LCBH/UKano/wiki`.

## 10.2 Case Studies

In this section we apply our verification method to several case studies. We rely on our tool UKano to check whether the protocol under study satisfies frame opacity and well-authentication as defined in Chapter 9. We also discuss some variations of the protocols to examine how privacy is affected. Remind that if privacy can be established for concurrent sessions (*i.e.* $\dagger_I = \dagger_R = !$) then it implies privacy for all other scenarios (when one or the two roles play sessions only sequentially). We thus implicitly model protocols with concurrent sessions and discuss alternative scenarios only when attacks are found.

As explained in Section 10.1, UKano relies on ProVerif; the source code of our tool and material to reproduce results can be found in [UKA]. Before diving into our real-world case studies, we recall that all our running examples developed so far have been automatically analysed with our tool UKano (see Table 9.1). Furthermore, frame opacity can be automatically established for all of them using any heuristics to build idealisation (except Example 53 for which only the default and the fully syntactical heuristics allow to conclude). Furthermore, all case studies discussed in this section except two (*i.e.* PACE in Subsection 10.2.4 and ABCDH in Subsection 10.2.5) have been automatically verified using our tool UKano without any manual effort. We discuss little manual efforts needed to conclude for PACE and ABCDH in the dedicated sections.

We used UKano v0.2 based on ProVerif v1.92 on a computer with following specifications:

- OS: Linux 3.10-2-amd64 #1 SMP Debian 3.10.5-1x86_64 GNU/Linux
- CPU / RAM: Intel(R) Xeon(R) CPU X5650 @ 2.67GHz / 47GO

### 10.2.1 Hash-Lock Protocol

We consider the Hash-Lock protocol as described in [JW09]. This is an RFID protocol that has been designed to achieve privacy even if no formal proof is given. The protocol relies on a hash function, and can be informally described as follows.

$$\begin{array}{rcll} \mathsf{Reader} & \to & \mathsf{Tag}: & n_R \\ \mathsf{Tag} & \to & \mathsf{Reader}: & n_T,\ \mathsf{h}(n_R, n_T, k) \end{array}$$

This protocol falls into our generic class of 2-party protocols in the shared case, and frame opacity and well-authentication can be automatically established in less than 0.01 second. We can therefore conclude that the protocol preserves unlinkability (note that anonymity does not make sense here). All implemented heuristics (see those three heuristics in Section 10.1) were able to successfully establish frame opacity automatically.

### 10.2.2 LAK Protocol

We present an RFID protocol first introduced in [LAK06], and we refer to the description given in [VDR08]. To avoid traceability attacks, the main idea is to ask the tag to generate a nonce and to use it to send a different message at each session. We suppose that initially, each tag has his own key $k$ and the reader maintains a database containing those keys.

The protocol is informally described below ($\mathsf{h}$ models a hash function). In the original version (see *e.g.* [VDR08]), in case of a successful execution, both parties update the key $k$ with $\mathsf{h}(k)$ (they always store the last two keys). Our framework does not allow one to model protocols that rely on a mutable state. Therefore, we consider here a version where the key is not updated at the end of a successful execution allowing the key $k$ to be reused from one session to another. This protocol lies in the shared case since the identity name $k$ is used by the reader and the tag.

$$\begin{array}{rcll} \mathsf{Reader} & \to & \mathsf{Tag}: & r_1 \\ \mathsf{Tag} & \to & \mathsf{Reader}: & r_2,\ \mathsf{h}(r_1 \oplus r_2 \oplus k) \\ \mathsf{Reader} & \to & \mathsf{Tag}: & \mathsf{h}(\mathsf{h}(r_1 \oplus r_2 \oplus k) \oplus k \oplus r_1) \end{array}$$

Actually, this protocol suffers from an authentication attack. The protocol does not allow the reader to authenticate the tag. This attack can be informally described as follows (and already exists on the original version of this protocol). By using algebraic properties of $\oplus$, an attacker can impersonate a tag by injecting previously eavesdropped messages. Below, $I(\mathsf{A})$ means that the attacker plays the role $\mathsf{A}$.

$$\begin{array}{rcll} I(\mathsf{Reader}) & \to & \mathsf{Tag}: & r_1 \\ \mathsf{Tag} & \to & \mathsf{Reader}: & r_2,\ \mathsf{h}(r_1 \oplus r_2 \oplus k) \\ \\ \mathsf{Reader} & \to & \mathsf{Tag}: & r_1' \\ I(\mathsf{Tag}) & \to & \mathsf{Reader}: & r_2^I,\ \mathsf{h}(r_1 \oplus r_2 \oplus k) \\ \mathsf{Reader} & \to & \mathsf{Tag}: & \mathsf{h}(\mathsf{h}(r_0 \oplus r_1 \oplus k) \oplus k \oplus r_1') \end{array}$$

where $r_2^I = r_1' \oplus r_1 \oplus r_2$, thus $\mathsf{h}(r_1 \oplus r_2 \oplus k) =_{\mathsf{E}} \mathsf{h}(r_1' \oplus r_2^I \oplus k)$.

Due to this, the protocol does not satisfy our well-authentication requirement even with sessions in sequence for $\mathsf{Tag}$ and $\mathsf{Reader}$. Indeed, the reader can end a session with a tag whereas the tag has not really participated to this session. In other words, the reader passes a test (which does not correspond to a safe conditional) with success, and therefore performs a $\tau_{\mathsf{then}}$ action whereas it has not interacted honestly with a tag.

Actually, this trace can be turned into an attack against the unlinkability property. Indeed, by continuing the previous trace, the reader can send a new request to the tag generating a fresh nonce $r_1''$. The attacker $I(\mathsf{Tag})$ can again answer to this new request choosing his nonce $r_2''$

accordingly, *i.e.* $r_2'' = r_1'' \oplus r_1 \oplus r_2$. This execution, involving two sessions of the reader talking to the same tag, cannot be mimicked in the single session scenario, and corresponds to an attack trace.

More importantly, this scenario can be seen as a traceability attack on the original version of the protocol (the stateful version) leading to a practical attack. The attacker will first start a session with the targeted tag by sending it a nonce $r_0$ and storing its answer. Then, later on, he will interact with the reader as described in the second part of the attack scenario. Two situations may occur: either the interaction is successful meaning that the targeted tag has not been used since its last interaction with the attacker; or the interaction fails meaning that the key has been updated on the reader's side, and thus the targeted tag has performed a session with the reader since its last interaction with the attacker. This attack shows that the reader may be the source of leaks exploited by the attacker to trace a tag. This is why we advocate for the strong notion of unlinkability we used, taking into account the reader and considering it as important as the tag.

We may note that the same protocol was declared untraceable in [VDR08] due to the fact that they have in mind a weaker notion of unlinkability.

To avoid the algebraic attack due to the properties of the xor operator, we may replace this operator using the pairing operator. The resulting protocol is a 2-party protocol that falls into our class, and for which frame opacity and well-authentication can be established (with concurrent sessions) using UKano, again in less than 0.01 second. Therefore, Theorem 8 allows us to conclude that it preserves unlinkability. Note also that frame opacity can be automatically checked using any heuristic described in Section 10.1.

### 10.2.3 BAC Protocol and some others

An e-passport is a paper passport with an RFID chip that stores the critical information printed on the passport. The International Civil Aviation Organization (ICAO) standard [ICA04] specifies several protocols through which this information can be accessed. Before executing the Basic Access Control (BAC) protocol, the reader optically scans a weak secret from which it derives two keys $k_E$ and $k_M$ that are then shared between the passport and the reader. Then, the BAC protocol establishes a key seed from which two sessions keys are derived. The session keys are then used to prevent skimming and eavesdropping on the subsequent communication with the e-passport.

In [ACRR10], two variants of the BAC protocol are described and analysed w.r.t. the unlinkability property as formally stated in this part. We refer below to these two variants as the French version and the United Kingdom (U.K.) version. The U.K. version is claimed unlinkable (with no formal proof) whereas an attack is reported on the French version. To explain the difference between the two versions, we give a description of the passport's role[1] in Figure 10.3. The relevant point is the fact that, in case of failure, the French version sends a different error

[1] We do not model the getChallenge constant message that is used to initiate the protocol but it is clear this message does not play any role regarding the security of the protocol.

$$\begin{array}{ll}
\mathsf{Tag} \to \mathsf{Reader}: & n_T \\
\mathsf{Reader} \to \mathsf{Tag}: & \{n_R, n_T, k_R\}_{k_E}, \mathsf{mac}_{k_M}(\{n_R, n_T, k_R\}_{k_E}) \\
\mathsf{Tag} \to \mathsf{Reader}: & \{n_T, n_R, k_T\}_{k_E}, \mathsf{mac}_{k_M}(\{n_T, n_R, k_T\}_{k_E})
\end{array}$$

*The BAC protocol using Alice & Bob notation between* Tag *(*i.e. *passport) and* Reader *is depicted above. The corresponding process modelling* Tag *is defined below, where* $m = \mathsf{senc}(\langle n_T, \langle \pi_1(\mathsf{sdec}(x_E, k_E)), k_T\rangle\rangle, k_E)$.

$$\begin{array}{l}
T(k_E, k_M) \;= \nu n_T.\nu k_T.\mathtt{out}(c_T, n_T).\mathtt{in}(c_T, x). \\
\mathsf{let}\ x_E = \pi_1(x), x_M = \pi_2(x), z_{\mathsf{test}} = \mathsf{eq}(x_M, \mathsf{mac}(x_E, k_M))\ \mathtt{in} \\
\quad \mathsf{let}\ z'_{\mathsf{test}} = \mathsf{eq}(n_T, \pi_1(\pi_2(\mathsf{sdec}(x_E, k_E))))\ \mathtt{in} \\
\qquad\qquad\qquad \mathtt{out}(c_T, \langle m, \mathsf{mac}(m, k_M)\rangle) \\
\quad \mathsf{else}\ \mathtt{out}(\mathsf{error}_{\mathsf{Nonce}}) \\
\mathsf{else}\ \mathtt{out}(\mathsf{error}_{\mathsf{Mac}})
\end{array}$$

*We consider the signature given in Example 1 (Chapter 2) augmented with a function symbol* mac *of arity* 2*. This is a public constructor whose purpose is to model message authentication code, taking as arguments the message to authenticate and the mac key. There is no rewriting rule and no equation regarding this symbol. We also assume public constants to model error messages. The U.K. version of the protocol does not distinguish the two cases of failure,* i.e. $\mathsf{error}_{\mathsf{Mac}}$ *and* $\mathsf{error}_{\mathsf{Nonce}}$ *are the same constant, whereas the French version does.*

**Figure 10.3** Description of the BAC protocol

message indicating whether the failure occurs due to a problem when checking the mac, or when checking the nonce. This allows the attacker to exploit this conditional to learn if the mac key of a Tag is the one used in a given message $\langle m, \mathsf{mac}(m, k)\rangle$. Using this, he can very easily trace a tag $T$ by first eavesdropping an honest interaction between the tag $T$ and a reader.

The U.K. version of the BAC protocol is a 2-party protocol according to our definition. Note that since the two error messages are actually identical, we can merge the two let instructions, and therefore satisfy our definition of being a responder role. Then, we automatically proved frame opacity and well-authentication using UKano. It took less than 10 seconds. Therefore, Theorem 8 allows us to conclude that unlinkability is indeed satisfied.

Regarding the French version of this protocol, it happens that the passport's role is neither an initiator role, nor a responder role according to our formal definition. Indeed, our definition of a role, and therefore of a 2-party protocol does not allow to model two sequences of tests that will output different error messages in case of failure. As illustrated by the attack on the French version of the BAC protocol, imposing this syntactic condition is actually a good design principle w.r.t. unlinkability.

Once the BAC protocol has been successfully executed, the reader gains access to the information stored in the RFID tag through the Passive and Active Authentication protocols (PA and AA). They are respectively used to prove authenticity of the stored information and prevent cloning attacks, and may be executed in any order. A formal description of these protocols is available in [ACD12]. These two protocols also fall into our class and our conditions can be checked automatically both for unlinkability and anonymity properties. We can also use our

technique to analyse directly the three protocols together (*i.e.* the U.K. version of the BAC together with the PA and AA protocols in any order). We analysed both orders: on one hand, we analysed BAC then PA then AA and, on the other hand, we analysed BAC then AA then PA. We thus prove unlinkability and anonymity w.r.t. all private data stored in the RFID chip (name, picture, etc.). UKano concludes within 4 minutes to establish both well-authentication and frame opacity (for any order PA/AA). Note that frame opacity can also be automatically checked using any heuristic described in Section 10.1. However, heuristics producing more complex idealisations (*e.g.* syntactical canonical construction) are less efficient: frame opacity can be established in 3 minutes using the greedy heuristic and in 20 minutes using the syntactic construction.

### 10.2.4 PACE Protocol

The Password Authenticated Connection Establishment protocol [PAC] (PACE) has been proposed by the German Federal Office for Information Security (BSI) to replace the BAC protocol. It has been studied in the literature [BFK09], [BDFK12], [CSD+12] but to the best of our knowledge, no formal proofs about privacy have been given. Similarly to BAC, the purpose of PACE is to establish a secure channel based on an optically-scanned key $k$. The protocol includes four main steps (see Figure 10.4):

- The tag randomly chooses a random number $s_T$, encrypts it with the symmetric key $k$ shared between the tag and the reader and sends the encrypted random number to the reader (message 1).
- Both the tag and the reader perform a Diffie-Hellman exchange (messages 2 & 3), and derive $G$ from $s_T$ and $g^{n_R n_T}$.
- The tag and the reader perform a Diffie-Hellman exchange based on the parameter $G$ computed at the previous step (messages 5 & 6).
- The tag and the reader derive a session key $k'$ which are confirmed by exchanging and checking the authentication tokens (messages 8 & 9).

A description in Alice & Bob notation is given in Figure 10.4. Moreover, at step 6, the reader will not accept as input a message which is equal to the previous message that it has just sent.

To formalise such a protocol, we consider the following signature:

$$\Sigma_c = \{\mathsf{senc},\ \mathsf{sdec},\ \mathsf{dh},\ \mathsf{mac},\ \mathsf{gen},\ \mathsf{g},\ \mathsf{ok}\} \text{ and } \Sigma_d = \{\mathsf{neq}\}.$$

Except g and ok which are public constants, all these function symbols are public constructor symbols of arity 2. The destructor neq has already be defined in Section 2.3 (in Chapter 2). The symbol dh is used to model modular exponentiation whereas mac will be used to model message authentication code. We consider the equational theory E defined by the following equations:

$$\begin{aligned} \mathsf{sdec}(\mathsf{senc}(x,y),y) &= x \\ \mathsf{dh}(\mathsf{dh}(x,y),z) &= \mathsf{dh}(\mathsf{dh}(x,z),y) \end{aligned}$$

1. Tag → Reader : $\{s_T\}_k$
2. Reader → Tag : $g^{n_R}$
3. Tag → Reader : $g^{n_T}$
4. Both parties compute $G = \mathsf{gen}(s_T, g^{n_R n_T})$.
5. Reader → Tag : $G^{n'_R}$
6. Tag → Reader : $G^{n'_T}$
7. Both parties compute $k' = G^{n'_R n'_T}$
8. Reader → Tag : $\mathsf{mac}(G^{n'_T}, k')$
9. Tag → Reader : $\mathsf{mac}(G^{n'_R}, k')$

**Figure 10.4** PACE in Alice & Bob notation

This protocol falls into our generic class of 2-party protocols. We take

$$\Pi_{\mathsf{PACE}} = (k, \{s_T, n_T, n'_T\}, \{n_R, n'_R\}, !, !, \mathcal{I}_{\mathsf{PACE}}, \mathcal{R}_{\mathsf{PACE}})$$

where the $\mathcal{R}_{\mathsf{PACE}}$ process (reader), described in Figure 10.5, is a responder role (we do not detail the continuation $R'$ and we omit trivial conditionals). The process modelling the role $\mathcal{I}_{\mathsf{PACE}}$ can be obtained in a similar way.

$$\begin{aligned}
\mathcal{R}_{\mathsf{PACE}} := \quad & \mathtt{in}(c_R, y_1).\\
& \mathtt{out}(c_R, \mathsf{dh}(\mathsf{g}, n_R)).\mathtt{in}(c_R, y_2).\\
& \mathtt{out}(c_R, \mathsf{dh}(G, n'_R)).\mathtt{in}(c_R, y_3).\\
& \mathsf{let}\ y_{\mathsf{test}} = \mathsf{neq}(y_3, \mathsf{dh}(G, n'_R))\ \mathtt{in}\\
& \quad \mathtt{out}(c_R, \mathsf{mac}(y_3, k'));\\
& \quad \mathtt{in}(c_R, y_4).\\
& \quad \mathsf{let}\ y_5 = \mathsf{eq}(y_4, \mathsf{mac}(\mathsf{dh}(G, n'_R), k'))\ \mathtt{in}\ R'.
\end{aligned}$$

where $G = \mathsf{gen}(\mathsf{sdec}(y_1, k), \mathsf{dh}(y_2, n_R))$ and $k' = \mathsf{dh}(y_3, n'_R)$.

**Figure 10.5** Process $\mathcal{R}_{\mathsf{PACE}}$

Unfortunately, ProVerif cannot handle the equation above on the dh operator (due to some termination issues). Instead of that single equation, we consider the following equational theory that is more suitable for ProVerif:

$$\begin{aligned}
\mathsf{dh}(\mathsf{dh}(\mathsf{g}, y), z) &= \mathsf{dh}(\mathsf{dh}(\mathsf{g}, z), y)\\
\mathsf{dh}(\mathsf{dh}(\mathsf{gen}(x_1, x_2), y), z) &= \mathsf{dh}(\mathsf{dh}(\mathsf{gen}(x_1, x_2), z), y)
\end{aligned}$$

This is sufficient for the protocol to work properly but it obviously lacks equations that the attacker may exploit.

First, we would like to highlight an imprecision in the official specification [PAC] that may lead to practical attacks on unlinkability. As the specification seems to not forbid it, we could have assumed that the decryption operation in $G = \mathsf{gen}(\mathsf{sdec}(y_1, k), \mathsf{dh}(y_2, n_R))$ is implemented in such a way that it may fail when the key $k$ does not match with the key of the ciphertext $y_1$. In that case, an attacker could eavesdrop a first message $c^0 = \mathsf{senc}(s_T^0, k^0)$ of a certain tag $T^0$ and then, in a future session, it would let the reader optically scan a tag $T$ but replace its challenge $\mathsf{senc}(s_T, k)$ by $c^0$ and wait for an answer of the reader. If it answers, he learns that the decryption did not fail and thus $k = k^0$: the tag $T$ is actually $T^0$. We discovered this attack

using our method since, in our first attempt to model the protocol, we modelled $\mathsf{sdec}(\cdot,\cdot)$ as a destructor (that may fail) and the computation of $G$ as an evaluation:

$$\mathsf{let}\ G = \mathsf{gen}(\mathsf{sdec}(y_1, k), \mathsf{dh}(y_2, n_R))\ \mathtt{in}\ [...]$$

This test has to satisfy our requirement in order to declare the protocol well-authenticating. But this conditional computing $G$ is not safe and does not satisfy the requirements of Definition 63 (the attack scenario described is a counter-example). The same attack scenario shows that the protocol does not ensure unlinkability (this scenario cannot be observed when interacting with $\mathcal{S}_\Pi$). Similarly to the attack on LAK, we highlight here the importance to take the reader into account and give it as much importance as the tag in the definition of unlinkability. Indeed, it is actually a leakage from the reader that allows an attacker to trace a specific tag.

Second, we report on an attack[2] that we discovered using our method on some models of PACE found in the literature [BFK09],[BDFK12],[CSD+12]. Indeed, in all those papers, the first conditional of the reader

$$\mathsf{let}\ y_{\mathsf{test}} = \mathsf{neq}(y_3, \mathsf{dh}(G, n'_R))\ \mathtt{in}$$

is omitted. Then the resulting protocol does not satisfy the well-authentication condition. To see this, we simply have to consider a scenario where the attacker will send to the reader the message it has outputted at the previous step. Such an execution will allow the reader to execute its role until the end, and therefore execute $\tau_{\mathsf{then}}$, but the resulting trace is not an honest one. Again, this scenario can be turned into an attack against unlinkability as explained next. As before, an attacker could eavesdrop a first message $c^0 = \mathsf{senc}(s_T^0, k^0)$ of a certain tag $T^0$. Then, in a future session, it would let the reader optically scan a tag $T$ but replace its challenge $\mathsf{senc}(s_T, k)$ by $c^0$. Independently of whether $k$ is equal to $k^0$ or not, the reader answers $g^{n_R}$. The attacker then plays the two rounds of Diffie-Hellman by reusing messages from the reader (he actually performs a reflection attack). More precisely, he replies with $g^{n_T} = g^{n_R}$, $G^{n'_T} = G^{n'_R}$ and $\mathsf{mac}(G^{n'_R}, k') = \mathsf{mac}(G^{n'_T}, k')$. The crucial point is that the attacker did not prove he knows $k$ (whereas he is supposed to do so to generate $G$ at step 4) thanks to the reflection attack that is not detected. Now, the attacker waits for the reader's answer. If it is positive (the process $R'$ is executed), he learns that $k = k^0$: the tag $T$ is actually the same as $T^0$.

Third, we turn to PACE as properly understood from the official specification: when the latter test is present and the decryption may not fail. In that case, we report on a new attack. UKano found that the last test of the reader violates well-authentication. This is the case for the following scenario: the message $\mathsf{senc}(s_T, k)$ sent by a tag $T(k, n_T)$ is fed to two readers $R(k, n_R^1), R(k, n_R^2)$ of same identity name. Then, the attacker just forwards messages from one reader to the other. They can thus complete the two rounds of Diffie-Hellman (note that the test avoiding reflection attacks holds). More importantly, the mac-key verification phase (messages 8 and 9 from Figure 10.4) goes well and the attacker observes that the last conditional of the two readers holds. This violates well-authentication but also unlinkability because the latter scenario

[2]For that different attack, we obviously consider that decryption is a constructor, and thus cannot fail.

cannot be observed at all in $\mathcal{S}_\Pi$: if the attacker makes two readers talk to each other in $\mathcal{S}_\Pi$ they cannot complete a session because they must have different identity names. In practice, this flaw seems hard to exploit but it could be a real privacy concern: if a tag initiates multiple readers, an attacker may learn which ones it had initiated by forwarding messages from one to another. It does not seem to be realistic in the e-passport scenario, but could be harmful in other contexts. It seems that, in the e-passport context, a modelling with sequential sessions would be more realistic. We come back to such a modelling in Chapter 11.

Finally, we propose a simple fix to the above attack by adding tags avoiding confusions between reader's messages and tag's messages. It suffices to replace messages 8 and 9 from Figure 10.4 by respectively $\mathsf{mac}(\langle \mathsf{c}_r, G^{n'_T} \rangle, k')$ and $\mathsf{mac}(\langle \mathsf{c}_t, G^{n'_R} \rangle, k')$ where $\mathsf{c}_r, \mathsf{c}_t$ are public constants, and adding the corresponding checks. Well-authentication can be automatically established using UKano in around 1 minute. However, the model produced by UKano to verify the frame opacity condition had to be slightly simplified (while preserving the semantics) in order to make ProVerif conclude more quickly. With this little extra manual effort, we verified frame opacity in less than 2 minutes. Therefore, PACE with tags preserves unlinkability in the model considered here.

### 10.2.5 Attributed-Based Authentication Scenario Using ABCDH Protocol

Most authentication protocols are identity-based: the user needs to provide his identity and prove to the service provider he is not trying to impersonate somebody else. However, in many contexts, the service provider just needs to know that the user has some non-identifying attributes (*e.g.* age, gender, country, membership). For instance, a bookshop just needs to have the proof that the user has the right to borrow books and does not need to know the full identity of the user. Attribute-based authentication protocols solve this problem and allow a user to prove to another user, within a secure channel, that he has some attributes without disclosing its identity.

We used our method to automatically establish unlinkability of a typical use case of such a protocol taking part to the IRMA project[3]. We analysed a use case of the protocol ABCDH as defined in [AH13]. This protocol allows a smartcard $C$ to prove to some Verifier $V$ (which is also a smartcard) that he has the required attributes. The protocol aims at fulfils this goal without revealing the identity of $C$ to $V$ or to anyone else. One of its goal is also to avoid that any other smartcard $C'$ replays those attributes later on. The protocol should also ensure unlinkability of $C$. To the best of our knowledge, there was no prior formal analysis of that security property for this protocol.

The key ingredient of this protocol is *attribute-based credential* (ABC). It is a cryptographic container for attributes. In ABC, attributes are signed by some issuers and allow for *selective disclosure* (SD): it is possible to produce a zero-knowledge (ZK) proof revealing a subset of attributes signed by the issuer along with a proof that the selected disclosed attributes are actually in the credential. This non-interactive proof protocol can be bound to some fresh data to avoid replay attacks. We shall use the notation $\mathsf{SD}(\overline{a_i}; n)$ to denote the selective disclosure of

[3]For more information about IRMA ("I Reveal My Attributes"), see `https://www.irmacard.org`.

attributes $\overline{a_i}$ bound to $n$. Note that $\mathsf{SD}(\emptyset; n)$ (no attribute is disclosed) still proves the existence of a credential. There are two majors ABC schemes: Microsoft U-Prove [PZ13] and IBM's Idemix [CLN12]. We decided to model IBM's scheme (since it is the one that is used in IRMA) following the formal model given in [CMS10]. It involves complex cryptographic primitives (*e.g.* commitments, blind signature, ZK proofs) but ProVerif can deal with them all. In this scheme, each user has a master secret never revealed to other parties. Issuers issue credentials bound to the master secret of users (note that users are known to issuers under pseudonyms). A SD consists in a ZK proof bound to $n$ proving some knowledge: knowledge of the master secret, knowledge of a credential bound to the master secret, knowledge that the credential has been signed by the given organisation, knowledge that the credential contains some given attributes.

We analyse the ABCDH [AH13] using the model of SD from [CMS10] used in the following scenario:

- an organisation $\mathsf{O}_{\text{age}}$ issues credentials about the age of majority;
- an organisation $\mathsf{O}_{\text{check}}$ issues credentials giving the right to check the age of majority;
- a user $C$ wants to watch a movie rated adult-only due to its violent contents; his has a credential from $\mathsf{O}_{\text{age}}$ with the attribute `adult`;
- a movie theatre $V$ wants to verify whether the user has the right to watch this movie; it has a credential from $\mathsf{O}_{\text{check}}$ with the attribute `canCheckAdult`.

The scheme is informally given in Figure 10.6. $n_V, n_C$ and $n$ are fresh nonces. Functions $f_i$ are independent hash functions; we thus model them as free constructor symbols. The construction $\mathsf{SD}(\cdot;\cdot)$ is not modelled atomically and follows [CMS10] but we do not describe here its details.

1. Verifier $\rightarrow$ Client : $\mathsf{dh}(\mathrm{g}, n_V), \mathsf{SD}(\texttt{canCheckAdult}; f_1(\mathsf{dh}(\mathrm{g}, n_V)))$
2. Client $\rightarrow$ Verifier : $\mathsf{dh}(\mathrm{g}, n_C), \mathsf{SD}(\emptyset; f_1(\mathsf{dh}(\mathrm{g}, n_V), \mathsf{dh}(g, n_C)))$
3. Verifier $\rightarrow$ Client : $\mathsf{senc}(\langle \texttt{0x00}, \mathsf{ok}\rangle, k)$
4. Client $\rightarrow$ Verifier : $\mathsf{senc}(\langle \texttt{0x01}, \mathsf{ok}\rangle, k)$
5. Verifier $\rightarrow$ Client : $\mathsf{senc}(\langle n; \texttt{requestAdult}\rangle, k)$
6. Client $\rightarrow$ Verifier : $\mathsf{senc}(\langle \texttt{adult}; \mathsf{SD}(\texttt{adult}; f_3(n, \texttt{seed}))\rangle, k)$

**Figure 10.6** ABCDH (where seed $= \mathsf{dh}(\mathsf{dh}(\mathrm{g}, n_C), n_V)$ and $k = f_2(\text{seed})$)

This 2-party protocol is in our class and falls in the category disjoint; *i.e.* $\overline{k}_\cap = \emptyset$ and $\overline{k}_\mathcal{I}, \overline{k}_\mathcal{R} \neq \emptyset$.

The complete model of this protocol is quite complex and can be found in [UKe]. Unfortunately, no heuristic currently implemented in UKano allowed to verify frame opacity fully automatically. Nevertheless, based on the file generated by UKano for the fully syntactical heuristic and after some manual transformations, we were able to verify frame opacity in about 40 minutes. Regarding well-authentication, due to the length of the protocol, the queries are also quite long. Because of the latter and the high complexity of the underlying term algebra, it required too much time for ProVerif to terminate. After some manual transformations and simplifications of

the generated file, we successfully established well-authentitcation for this protocol in about 2 hours and 20 minutes.

To sum up, we were able to verify unlinkability of the ABCDH protocol with minor manual efforts in around 3 hours.

### 10.2.6 DAA Join & DAA Sign

A Trusted Platform Module (TPM) is a hardware device aiming at protecting cryptographic keys and at performing some cryptographic operations. Typically, a user may authenticate himself to a service provider relying on such a TPM. The main advantage is to physically separate the very sensitive data from the rest of the system. On the downside however, such devices may be used by malicious agents to breach users' privacy by exploiting their TPMs. Direct Anonymous Attestation (DAA) protocols have been designed to let TPMs authenticate themselves whilst providing accountability and privacy.

In a nutshell, some issuers issue credentials representing membership to a group to the TPM using group signatures via the *DAA join* sub-protocol. Those credentials are bound to the internal secret of the TPM that must remain unknown to the service provider. Then, when a TPM is willing to prove to a verifier its membership to a group, it uses the *DAA sign* sub-protocol. We analysed the RSA-based DAA join and sign protocols as described in [SRC15]. Both protocols rely on complex cryptographic primitives (*e.g.* blind signature, commitments, and Zero Knowledge proofs) but ProVerif can deal with them all. Note that the authors of [SRC15] have automatically established a game-based version of unlinkability of the combination of DAA Join and DAA Sign using ProVerif. We have analysed both protocols separately since the combination of the two protocols is a 3-party protocol.

#### DAA Join

In the RSA-based DAA join protocol, the TPM starts by sending a credential request in the form of a commitment containing its internal secret, some session nonce and the public key of the issuer. The issuer then challenges the TPM with some fresh nonces encrypted asymmetrically with the public key of the TPM. After having received the expected TPM's answer, the issuer sends a new nonce as second challenge. To this second challenge, the TPM needs to provide a ZK proof bound to this challenge proving that he knows the internal secret on which the previous commitment was bound. Finally, after verifying this proof, the issuer blindly signs the commitment allowing the TPM to extract the required credential.

1. TPM $\rightarrow$ Issuer : $N_I, U$
2. Issuer $\rightarrow$ TPM : $\mathsf{aenc}((n, n_e), \mathrm{pub}_{\mathrm{TPM}})$
3. TPM $\rightarrow$ Issuer : $\mathsf{h}((U, (n, n_e)))$
4. Issuer $\rightarrow$ TPM : $n_i$
5. TPM $\rightarrow$ Issuer : $n_t, \mathsf{ZK}((\mathrm{tsk}, v'), \mathrm{zeta}_{\mathrm{I}}, N_I, U, (n_t, n_i))$
6. Issuer $\rightarrow$ TPM : $\mathsf{clsign}((e, v'', U), \mathrm{sk}_I)$

**Figure 10.7** DAA Join

We give in Figure 10.7 an Alice & Bob description of the protocol between the TPM and the issuer. The message $\text{zeta}_\text{I} = \mathsf{h}((0, \text{bsnI}))$ relies on bsnI: using a fresh bsnI allows to ensure that the session of DAA Join will be unlinkable from previous ones. The message $\text{tsk} = \mathsf{h}((\text{DAAseed}, \mathsf{h}(\text{pub}_\text{I})), \text{cnt}, 0)$ combines the internal secret of the TPM (*i.e.* DAAseed) with the public key of the issuer (*i.e.* $\text{pub}_\text{I}$). The commit message $N_I = \mathsf{commit}(\text{zeta}_\text{I}, \text{tsk})$ binds $\text{zeta}_\text{I}$ with the internal secret while the commit message $U = \mathsf{clommit}(\text{pub}_\text{I}, n_v, \text{tsk})$ expresses a credential request. The goal of the TPM will be to get the message $U$ signed by the issuer. More precisely, the issuer will blindly sign the message $U$ after making sure that the TPM can decrypt challenges encrypted with its public key (step 2.) and that he can provide a fresh ZK proof showing he knows its internal secret binds in $U$ and $N_I$ (step 5.). Finally, if all checks are successful, the issuer will blindly sign the credential request $U$ (step 6.). We note $\mathsf{clsign}((e, v'', U), \text{sk}_I)$ the blind signature of a commitment $U$ with signature key $\text{sk}_I$ and nonces $e, v''$. Note that $\mathsf{ZK}(\bullet, \bullet)$ has two arguments: the first one should contain private data and the second one should contain public data. One can always extract public data from ZK proofs and one can check if both public and private data match as expected.

This protocol falls in our class and lies in the shared case[4] (*i.e.* $fn(\mathcal{I}) \cap fn(\mathcal{R}) \neq \emptyset$). We automatically analysed this protocol with UKano and established both frame opacity and well-authentication in less than 5 seconds. frame opacity can be established using any heuristic implemented in UKano but the greedy heuristic allows one to obtain simpler idealised messages which yield better performance (2 seconds instead of 28 seconds for the fully syntactical heuristic).

### DAA Sign

Once a TPM has obtained such a credential, it may prove its membership using the DAA sign protocol. This protocol is played by a TPM and a verifier: the verifier starts by challenging the TPM with a fresh nonce (step 1.), the latter then sends a complex ZK proof bound to this nonce (step 2.). The latter ZK proof also proves that the TPM knows a credential from the expected issuer bound to a secret he knows (essentially a message $\mathsf{clsign}((e, v'', U), \text{sk}_I)$ received in a previous session of DAA join). The verifier accepts only if the ZK can be successfully checked (step 3.).

We give in Figure 10.8 an Alice & Bob description of the protocol between a verifier and the TPM willing to sign a message $m$ using its credential $\text{cred} = \mathsf{clsign}((e, v'', U), \text{sk}_I)$ he received from a past DAA join session. From its credential cred, the TPM will compute a new credential dedicated to the current sign session: $\text{cred}' = \mathsf{clcommit}((\text{pub}_\text{I}, \text{cred}), n_c)$. Indeed, if the TPM had directly used cred then two sessions of DAA sign would have been trivially linkable.

Again, this protocol falls in our class and lies in the non shared case. Indeed, we model infinitely many different TPMs may take part to the DAA sign protocol with any verifier whose role is always the same (he has no proper identity). Using the canonical syntactical idealisation for frame opacity, we automatically established (using UKano) frame opacity and well-authentication

[4]Before executing the join protocol, the TPM and the issuer should establish a one-way authenticated channel that is not specified by the DAA scheme.

1. Verifier → TPM : $n_V$
2. TPM → Verifier : $(\text{zeta}_\text{I}, \text{pub}_\text{I}, N_I, \text{cred}', n_t,$
   $\mathsf{ZK}((\text{tsk}, n_c), (\text{zeta}_\text{I}, \text{pub}_\text{I}, N_I, \text{cred}', (n_T, n_V, m))))$
3. Verifier → TPM : accept/reject

**Figure 10.8** DAA Sign

and thus unlinkability in less than 3 seconds. Frame opacity can be automatically established using the fully syntactical heuristic but not using other heuristics (greedy and the default heuristic).

### Summary

We now summarises our results in Table 10.1. We notably indicate the verification time in seconds to verify both conditions. When there is an attack, we give the time ProVerif takes to show that one of the condition fails to hold. We note ✓ for a condition automatically checked using our tool UKano and ✗ when the condition does not hold. Note that all positive results were established automatically using our tool UKano (which is based on ProVerif) without any manual effort (except for the cases indicated by ✓* for which little manual efforts were needed).

| Protocol | Frame opacity | Well-auth. | Unlinkability | Verification time |
|---|---|---|---|---|
| Hash-Lock | ✓ | ✓ | **safe** | 0.00s |
| LAK (stateless) | – | ✗ | attack | – |
| Fixed LAK | ✓ | ✓ | **safe** | 0.00s |
| BAC | ✓ | ✓ | **safe** | 8.41s |
| BAC/PA/AA | ✓ | ✓ | **safe** | 183.40s |
| BAC/AA/PA | ✓ | ✓ | **safe** | 198.28s |
| PACE (faillible dec) | – | ✗ | attack | 31.81s |
| PACE (as in [BFK09]) | – | ✗ | attack | 61.43 |
| PACE | – | ✗ | attack | 83.72s |
| PACE with tags | ✓* | ✓ | **safe** | 169.91 |
| DAA sign | ✓ | ✓ | **safe** | 2.94s |
| DAA join | ✓ | ✓ | **safe** | 4.68s |
| ABCDH (irma) | ✓* | ✓* | **safe** | 8479.76s |

**Table 10.1** Summary of our case studies

*Chapter 11*

# Conclusion

We have identified two conditions, namely well-authentication and frame opacity, which imply anonymity and unlinkability for a wide class of protocols. Additionally, we have shown that these two conditions can be checked automatically using the tool ProVerif, and we have mechanised their verification in a tool called UKano. This yields a new verification technique to check anonymity and unlinkability for an unbounded number of sessions. It has proved quite effective on various case studies. In particular, it has brought first-time unlinkability proofs for the BAC protocol (e-passport) and ABCDH protocol. Our case studies also illustrated that our methodology is useful to discover attacks against unlinkability and anonymity as illustrated by the new attacks we found on PACE and LAK.

In the next section, we discuss limitations of the present technique and list main avenues for future work.

## 11.1 Regarding Mechanisation and the Tool UKano

We start with limitations and possible improvements for the practical aspects of our method.

**Optimising generated models.** For complex protocols (*e.g.* PACE and ABCDH), UKano produces models for verifying our conditions that are rather large on which ProVerif may not terminate in reasonable time. While this was expected – the more complex the protocols are, the longer is the verification time – we think this effect could be diminished by optimising generated models. For instance, the manual modifications needed to make ProVerif terminate for PACE (Subsection 10.2.4) or for ABCDH (Subsection 10.2.5) are based on simple observations and could be mechanised in future versions of the tool UKano. We now discuss the two optimisations we needed to make ProVerif terminate in reasonable time.

First, when verifying well-authentication, one has to verify one query per conditional. Each query essentially states that if the underlying conditional holds for an agent then there must be a sequence of events modelling the fact that this agent is associated to another. For complex protocols, this sequence of events may be very long (*e.g.* 12 actions for ABCDH). However, it is possible

to reduce the size of such queries by relying on the other, smaller queries associated to other tests. For instance, assume the honest execution is of the form $\mathtt{out}(c_I, w_1).\mathtt{in}(c_R, w_1).\tau^1_{\mathsf{then}}.\mathtt{out}(c_R, w_2).\mathtt{in}(c_I, w_2).\tau^2_{\mathsf{then}}.\ldots$. If we already know that all conditionals $\tau^1_{\mathsf{then}}, \ldots .\tau^{k-1}_{\mathsf{then}}$ except the last one $\tau^k_{\mathsf{then}}$ satisfy well-authentication then, in order to show that $\tau^k_{\mathsf{then}}$ satisfies well-authentication as well, it should be sufficient to produce a query stating that if $\tau^k_{\mathsf{then}}$ holds for an agent $a$ then there exists an agent $a'$ that is associated with $a$ "between" the actions $\tau^{k-2}_{\mathsf{then}}$ and $\tau^k_{\mathsf{then}}$ and thus skipping all events associated to actions before $\tau^{k-2}_{\mathsf{then}}$.

Second, note that models generated for verifying frame opacity heavily rely on nested conditionals in order to overcome false negatives stemming from a conditional making sense on the left (real protocol) but not on the right (idealised frame); see Subsection 10.1.1 for a discussion. However, in many cases, a simple analysis of the protocol suffices to infer than some conditionals could not cause such mismatches and do not call for such costly precautions. We hope to be able to automate such analysis in order to produce simpler models.

Finally, a more efficient and also ambitious way to verify frame opacity could consist in developping a dedicated Horn clauses generation. One could adapt the ProVerif generation of clauses modelling executions of biprocesses for always prioritizing the left-process and then add clauses ProVerif generates for verifying static equivalence on resulting bi-frames.

**Other tools as back-ends.** Currently, our conditions are checked in UKano using ProVerif as back-end which, despite its great flexibility, supports only a limited kind of equational theory. In particular, the full Diffie-Hellman theory or associative-commutative theories needed for xor (widely used in RFID protocols) are not supported. However, we could consider using other tools such as Tamarin and Maude−NPA. For instance, we already successfully proved well-authentication for Example 52 in Tamarin by translating queries described in Subsection 10.1.2 as lemmas statements. We think such encodings could be mechanised. We would also like to investigate the case of frame opacity. We could then leverage the richer cryptographic primitives modelling capabilities of Tamarin (*e.g.* thanks to recent efforts [DDKS17]).

**Better idealisation heuristics.** Finally, we think that it is possible to make UKano build idealisations in a cleverer way, roughly by detecting when a term can be idealised by a nonce or when, on the contrary, it is crucial to retain its shape in order to remain precise enough. Hence, UKano could then use an adaptive heuristic by greedily idealising terms by session nonces when possible and use the more costly syntactical heuristic otherwise.

## 11.2 Regarding our Conditions and our Main Theorem

We also identify a number of research problems aimed at increasing the scope of our technique.

**Stateful protocols.** We would like to investigate the extension of our main theorem to the case of protocols with persistent state (from one session to the other). This is certainly technically challenging, but would make it possible to model more protocols (*e.g.* many RFID protocols

are stateful), or at least model them more faithfully. We think that a good starting point is to define a notion of *synchronised states* between agents' internal states. Intuitively, agents having synchronised states may successfully execute a session (or its continuation) but never in the other case. Next, we shall strengthen well-authentication by requiring that agents that pass a conditional must be associated and must have synchronised states all the way. This way, we may be able to reuse our proof technique. Indeed, control-flow *and* internal states of agents would then be reduced to the association relation between agents. We would like to push this idea further and formalise it.

**Beyond 2-party.** Our method only applies on 2-party protocols. For instance, this was the reason why we have not modelled DAA join in combination with DAA sign (see Subsection 10.2.6) since the resulting protocol would be a 3-party protocol. Going beyond 2-party protocols would thus greatly extend the scope of our technique.

**More generic notion of honest trace.** As already mentioned in Subsection 10.2.4, we have shown that the PACE protocol does not ensure unlinkability when sessions of e-passports and readers can be executed concurrently. However, one could argue that executing sessions concurrently does not seem practical. Indeed, e-passports RFID chips' content are assumed to be impossible to clone. It is thus expected that there exists at most one physical e-passport equipped with given identity names.

Therefore, we have investigated the scenario where sessions can be executed only sequentially. We have turned to Tamarin since ProVerif is not able to model faithfully such scenarios. We wrote a Tamarin model encoding well-authentication and found that this condition does not hold even with the tagged version (see Subsection 10.2.4). This contrasts with the positive result obtained with ProVerif. Actually, this comes from the fact that Tamarin models Diffie-Hellman exponentiation in a more faithful way than ProVerif. Some behaviours that were not possible in the ProVerif model become possible, and it happens that well-authentication is not satisfied in such a model. Indeed, the attacker can alter the Diffie-Hellman shares, as informally depicted in Figure 11.1, without impacting successive agents' conditionals. This is problematic because

| | | | | |
|---|---|---|---|---|
| 1. | Tag | $\rightarrow$ | Reader : | $\{s_T\}_k$ |
| 2. | Reader | $\rightarrow$ | Attacker : | $g^{n_R}$ |
| 2′. | Attacker | $\rightarrow$ | Tag : | $(g^{n_R})^X$ |
| 3. | Tag | $\rightarrow$ | Attacker : | $g^{n_T}$ |
| 3′. | Attacker | $\rightarrow$ | Reader : | $(g^{n_T})^X$ |

**Figure 11.1** Example of successful but dishonest interaction ($X$ can be any message)

successful tests will pass (independently of the message $X$) while such interactions are not honest according to our current definition of honest trace (see Definition 50). Obviously, this interaction is not an attack on unlinkability at all.

We have ideas to tackle this issue. First, one could extend the notion of "honest trace associated to a protocol" (Definition 51) as follows: to any protocol, we associate a set of symbolic

traces that are roughly traces with (possibly) variables in recipes. For instance, for PACE, one may use $\mathsf{tr}_h = \mathtt{out}(c_I, w_1).\mathtt{in}(c_R, \mathsf{dh}(w_1, X)).\mathtt{out}(c_R, w_2).\mathtt{in}(c_I, \mathsf{dh}(w_2, X)).\ldots$ in addition to the standard non-symbolic trace. However, in order to adapt our proof technique, we need to make sure that whatever the recipes chosen to fill in the variables (*e.g.* $X$ in $\mathsf{tr}_h$), the resulting concrete trace can be executed by the protocol and the produced frame has always the same idealisation. Remark that this is the case for $\mathsf{tr}_h$ in the case of the PACE protocol. We have hopes to adapt our theorem and our method this way in order to deal with PACE and other examples based on such Diffie-Hellman exchanges.

**Getting rid of equivalence.** We have conjectured that, for standard cryptographic primitives, frame opacity should be verifiable with enough precision via a conjunction of reachability properties (*e.g.* secrecy of encryption keys) and simple, syntactical checks (*e.g.* fresh nonce in ciphertext). We would like to prove such a result. A good starting point would be to consider (a)symmetric encryption, mac, signature and pairs defined via reduction rules with an empty set of equations $\mathsf{E}$. We are convinced that the above can be established for such term algebras. We think that seeking for such a result in a way that can be applied to real-world case studies is an attractive and exciting research goal since it would allow to completely reduce the verification of complex equivalence based-properties such as unlinkability into purely reachability verification problems. The practical implication is that the verification could then be carried out using one of the numerous and highly efficient tools dealing with reachability only.

**Compositional verification.** We believe that our conditions could be verified more compositionally than the original anonymity and unlinkability properties. We notably have insights about a way to verify frame opacity more compositionally. Indeed, assuming well-authentication item (i) has been previously established, we know that if a protocol session reaches an unsafe conditional, this session has necessarily followed the honest execution and all its outputs are thus fixed. This remark allows us to intuitively split a protocol at the level of a given unsafe conditional and verify frame opacity for a system made of (the combination of): (i) the first part of the two roles before the given conditional, (ii) the last part of the two roles where all honest, expected messages that the first parts of the roles would have sent are directly given to the attacker. Note that the second system is intuitively made of more processes in parallel but we stress the fact that each process is half the size of the original roles.

## 11.3 Reusing Core Ideas of the Methodology

We believe that the overall methodology developed in this part could be applied in other contexts where privacy is critical: *e.g.* e-voting, attribute-based credentials, blockchain technologies, transparent certificate authorities. We have already successfully done so for the case of vote-privacy for a large class of e-voting protocols [CH17]. We now try to sum up the main steps of the overall methodology. We stress the fact that we have followed those steps when developing the present method but also when devising the method for proving vote-privacy [CH17].

First, one has to delimit an interesting class of protocols including numerous real-world case studies and a privacy goal one typically want to establish for protocols in that class. Then, inspired by known attacks on the privacy goal for protocols in the class, one should try to identify main and deep reasons that enable those attacks. One way to achieve that consists in classifying known attacks into a few categories.

Second, one must devise a condition per class of attack that should neatly capture the ingredient enabling those attacks. Here, the main idea is to take advantage of (i) working with only a class of protocols and (ii) focusing on one simple aspect of the (often) complex privacy goal. Thanks to that, conditions obtained this way can be much simpler than the original privacy goal and can be verified much more easily and with more precision. Note that, in our case (for the present method and for [CH17]), we were able to nicely divide the problem between control-flow issues (*e.g.* handled by well-authentication) and the other issues (*e.g.* relations between messages handled by frame opacity). This is of great interest since it allows to get over control-flow issues when looking at the other aspects. More precisely, once one has made sure that the typical control-flow issues do not break the privacy goals, one can focus on a simpler problem roughly by focusing on the honest execution and forgetting that many other behaviours are possible. This has also been very helpful and interesting for the method for proving vote-privacy [CH17].

Third, one shall make sure one has gathered enough conditions and then prove the soundness theorem; *i.e.* the combination of conditions always implies the privacy goal for protocols in the class. Finally, one may take advantage of the fact that the complex privacy goal has been split up into simpler sufficient conditions to mechanise the verification of the simpler conditions.

We believe that this methodology is a successful approach that helps to advance the state-of-the-art for the highly complex problem of the verification of privacy-related properties, and paves the way for further improvements.

*Chapter 12*

# General Conclusion

## 12.1 Summary

When I started this thesis in 2013, several verification tools were already capable of analysing privacy-related properties. There were some recent decision procedures for a bounded number of sessions implemented in tools such as Apte, Akiss and Spec, and, pragmatic, yet highly efficient, semi-decision procedures for an unbounded number of sessions implemented *e.g.* in ProVerif and Tamarin. As shown throughout this thesis, having such (semi-)decision procedures is not enough. The aim of this thesis is to bridge the gap between, on one hand, such (semi-)decision procedures and, on the other hand, actual, practical verification of privacy goals for security protocols. For each line of work, we have identified an important issue limiting its practical impact. We have addressed those issues separately.

In Part B, we have focused on the state space explosion problem of decision procedures for a bounded number of sessions. We have developed new partial order reduction techniques compatible with equivalence verification than can be nicely integrated in frameworks on which those procedures are based.

Part C has been dedicated to solving the precision issue of semi-decision procedures for an unbounded number of sessions. We have developed a new method for verifying unlinkability and anonymity that does not suffer from this precision issue. We think that the underlying methodology allows to shed some light on those privacy goals. We also hope to reuse this methodology in broader contexts.

For addressing both issues, we have proceeded back and forth between theory and practice. Practical aspects have motivated and guided our developments. But our solutions to the identified problems have required highly theoretical developments. Furthermore, we have been continuously checking the practical relevance of our solutions by putting them into practice on real-world case studies and by conducting benchmarks.

## 12.2 Future Work

We have already listed avenues for future work and interesting open questions throughout the thesis (see sections 5.5 and 6.5 for Part B and Chapter 11 for Part C). Nevertheless, we now recall a few of them and discuss a few more that we find particularly interesting.

**POR without action-deterministic assumption.** Dropping the action-deterministic assumption whilst keeping the soundness and completeness of POR techniques is an interesting challenge. We have already discussed the difficulties which arise without this assumption in Section 5.5. Solving them for a variant of our POR techniques would allow to greatly extend their scope and thus improve verification efficiency in many more cases.

**POR for backward search.** We would also be interested in adapting our POR techniques for backward search in the unbounded case (*e.g.* in the Tamarin tool). The potential benefits for Tamarin would be twofold. First, automatic verification would be more efficient. Second, interactive-proofs would be less cumbersome since fewer cases would have to be explored and considered.

**Use our privacy verification method for the bounded case.** One could wonder whether our privacy verification method based on sufficient conditions could be adapted for the bounded case. We think that it is the case and we see potential benefits and interests in doing so. First, we claim that, replacing replications ! and repetitions ¡ by "bounded" replications $!^k$ and bounded repetitions $¡^k$, one can adapt the definitions of $\mathcal{M}_\Pi, \mathcal{S}_\Pi$, $\mathcal{M}_{\Pi,\overline{id}}$, strong unlinkability and anonymity (there is a need to match the number of total sessions on both sides though). We believe that our conditions straightforwardly restricted to the bounded cases allow us to prove that they always imply unlinkability and anonymity for this setting as well. There might be interests to develop such a method for two mains reasons.

First, compared to the verification in the unbounded case, frame opacity could then be decided instead of semi-decided. We thus benefit from the completeness of tools for the bounded case (*e.g.* Apte, Akiss).

Second, the verification for the bounded case of the two sufficient conditions would be more efficient. Concerning frame opacity, the condition seems easier to establish than unlinkability even in the bounded case since one can verify it with a process that is completely *action-deterministic* (see Definition 14 in Chapter 5) while it is not possible to do so for the original properties of interest (*e.g.* unlinkability). Therefore, one could benefit from the important speedups brought by our POR techniques developed in the Part B while it was not possible to leverage them to directly verify unlinkability. Indeed, if you force agents to use dedicated and pairwise distinct channels then unlinkability is broken by construction. Concerning well-authentication, the condition could be verified for the bounded case using tools such as ProVerif or Tamarin since they generally are precise enough when it comes to verifying purely reachability properties.

We would like to pursue our investigation towards this direction. If it can be completed, our hope is to achieve twofold benefits: (i) better precision compared to verification entirely carried out in tools like ProVerif or Tamarin and (ii) better efficiency compared to verification done entirely in tools like Apte or Akiss.

**Privacy via sufficient conditions approach.** Another line of work we would like to pursue is to adapt the methodology presented in Part C in broader contexts. We have already argued in considerable detail in Chapter 11 that we consider the overall methodology generic enough to have hopes to derive new methods for analysing other privacy goals for other classes of security protocols.

**From privacy to reachability.** We think that seeking to verify frame opacity via reachability properties only, as proposed in Chapter 11, is an exciting research goal. Indeed, achieving the latter, in a more generic setting if possible, would allow to completely getting rid of the complex and costly equivalence verification when it comes to analyse privacy in security protocols.

**Guidelines for privacy.** In our opinion, the privacy via sufficient conditions approach also sheds light on the privacy notions themselves. Indeed, each sufficient condition helps to get a better grasp of necessary ingredients for preserving privacy. It thus might be interesting to translate such conditions into more comprehensive guidelines helping the design of new privacy-enhancing protocols. It may be achieved by carefully extracting essential ingredients behind sufficient conditions presented in Part C, [CH17] and in methods to come based on the same idea. Indeed, it may be the case that patterns emerge from different conditions used in several methods. For instance, one may remark strong similarities between the present *well-authentication condition* and the *dishonest condition* of [CH17].

To conclude, we believe that, although imperfect and still requiring much improvements, existing techniques for verifying privacy have reached enough maturity to guide design, analysis and standardisation. It took many years and much research effort to develop usable methods for verifying reachability properties and a few more years to get the standardisation and industrial community to actually use them. We now have the encouraging example of the TLS Working Group adopting an "analysis-before-deployment" paradigm for drafting TLS 1.3. We hope to see in the near future such examples that also take *privacy* into account.

# Bibliography


[3sh] 3SHAKE website. `https://mitls.org/pages/attacks/3SHAKE`. Accessed: 2016-01-05.

[A+05] A. Armando et al. The AVISPA Tool for the automated validation of internet security protocols and applications. In *Proc. 17th Int. Conference on Computer Aided Verification*, LNCS. Springer, 2005.

[AAJS14] Parosh Abdulla, Stavros Aronis, Bengt Jonsson, and Konstantinos Sagonas. Optimal dynamic partial order reduction. *ACM SIGPLAN Notices*, 49(1):373–384, 2014.

[AB03] Martín Abadi and Bruno Blanchet. Computer-assisted verification of a protocol for certified email. In *Static Analysis*, pages 316–335. Springer, 2003.

[Aba97] Martin Abadi. Secrecy by typing in security protocols. In *Theoretical Aspects of Computer Software*, pages 611–638. Springer, 1997.

[ABD+15] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, et al. Imperfect forward secrecy: How diffie-hellman fails in practice. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 5–17. ACM, 2015.

[ACC+08] Alessandro Armando, Roberto Carbone, Luca Compagna, Jorge Cuellar, and Llanos Tobarra. Formal analysis of saml 2.0 web browser single sign-on: breaking the saml-based single sign-on for google apps. In *Proceedings of the 6th ACM workshop on Formal methods in security engineering*, pages 1–10. ACM, 2008.

[ACD12] Myrto Arapinis, Vincent Cheval, and Stéphanie Delaune. Verifying privacy-type properties in a modular way. In *Proceedings of the 25th IEEE Computer Security Foundations Symposium*, pages 95–109, Cambridge Massachusetts, USA, June 2012. IEEE Computer Society Press.

[ACRR09] Myrto Arapinis, Tom Chothia, Eike Ritter, and Mark Ryan. Untraceability in the applied pi-calculus. In *Proc. International Conference for Internet Technology and Secured Transactions*, pages 1–6. IEEE Computer Society Press, 2009.

[ACRR10] Myrto Arapinis, Tom Chothia, Eike Ritter, and Mark Ryan. Analysing unlinkability and anonymity using the applied pi calculus. In *Proceedings of the IEEE Computer Security Foundations Symposium*. IEEE Comp. Soc. Press, 2010.

[ADMP+09] Ben Adida, Olivier De Marneffe, Olivier Pereira, Jean-Jacques Quisquater, et al. Electing a university president using open-audit voting: Analysis of real-world use of Helios. *EVT/WOTE*, 9:10–10, 2009.

[AF01] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proceedings of the ACM SIGPLAN Symposium on Principles of Programming Languages*. ACM Press, 2001.

[AF04] Martín Abadi and Cédric Fournet. Private authentication. *Theoretical Computer Science*, 322(3), 2004.

[AG97] Martín Abadi and Andrew D Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proc. of the 4th ACM conference on Computer and communications security*, pages 36–47. ACM, 1997.

[AG98] Martín Abadi and Andrew D. Gordon. A bisimulation method for cryptographic protocols. *Nord. J. Comput.*, 5(4):267, 1998.

[AH13] Gergely Alpár and Jaap-Henk Hoepman. A secure channel for attribute-based credentials. In *Proceedings of the 2013 ACM workshop on Digital identity management*, pages 13–18. ACM, 2013.

[AK79] A.V. Anisimov and D.E. Knuth. Inhomogeneous sorting. *International Journal of Computer & Information Sciences*, 8(4):255–260, 1979.

[AMR+12] Myrto Arapinis, Loretta Mancini, Eike Ritter, Mark Ryan, Nico Golde, Kevin Redon, and Ravishankar Borgaonkar. New privacy issues in mobile telephony: fix and verification. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 205–216. ACM, 2012.

[AMRR14] Myrto Arapinis, Loretta Ilaria Mancini, Eike Ritter, and Mark Ryan. Privacy through pseudonymity in mobile telephony systems. In *Network and Distributed System Security Symposium*, 2014.

[And92] Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *J. Log. Comput.*, 2(3), 1992.

[APTa] APTE webpage. `http://projects.lsv.ens-cachan.fr/APTE/`. Accessed: 2016-01-05.

[aptb] Sources of APTE. Available at `https://github.com/APTE/APTE`. Accessed: 2016-01-05.

[AR00] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In *Proc. International Conference on Theoretical Computer Science*, pages 3–22, 2000.

[BAF05] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated Verification of Selected Equivalences for Security Protocols. In *20th Symposium on Logic in Computer Science*, pages 331–340, 2005.

[BAF08] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated verification of selected equivalences for security protocols. *Journal of Logic and Algebraic Programming*, 75(1):3–51, 2008.

[Bau05] Mathieu Baudet. Deciding security of protocols against off-line guessing attacks. In *Proc. 12th Conference on Computer and Communications Security*. ACM, 2005.

[BCDH10] Mayla Brusó, Konstantinos Chatzikokolakis, and Jerry Den Hartog. Formal verification of privacy for RFID systems. In *Proc. 23rd Computer Security Foundations Symposium*, pages 75–88. IEEE Computer Society Press, 2010.

[BCEDH13] Mayla Brusó, Konstantinos Chatzikokolakis, Sandro Etalle, and Jerry Den Hartog. Linking unlinkability. In *Trustworthy Global Computing*, pages 129–144. Springer, 2013.

[BDFK12] Jens Bender, Özgür Dagdelen, Marc Fischlin, and Dennis Kügler. The PACE AA protocol for machine readable travel documents, and its security. In *Financial Cryptography and Data Security*, pages 344–358. Springer, 2012.

[BDH] David Baelde, Stéphanie Delaune, and Lucca Hirschi. A reduced semantics for deciding trace equivalence. Under submission.

[BDH14] David Baelde, Stéphanie Delaune, and Lucca Hirschi. A reduced semantics for deciding trace equivalence using constraint systems. In *Proc. 3rd Conference on Principles of Security and Trust*, pages 1–21. Springer, 2014.

[BDH15] David Baelde, Stéphanie Delaune, and Lucca Hirschi. Partial order reduction for security protocols. In *26th International Conference on Concurrency Theory (CONCUR'15)*, page 497, 2015.

[BDS15] David Basin, Jannik Dreier, and Ralf Sasse. Automated symbolic proofs of observational equivalence. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1144–1155. ACM, 2015.

[BFG+14] Gilles Barthe, Cédric Fournet, Benjamin Grégoire, Pierre-Yves Strub, Nikhil Swamy, and Santiago Zanella-Béguelin. Probabilistic relational verification for cryptographic implementations. In *ACM SIGPLAN Notices*, volume 49, pages 193–205. ACM, 2014.

[BFK09] Jens Bender, Marc Fischlin, and Dennis Kügler. Security analysis of the PACE key-agreement protocol. In *Information Security*, pages 33–48. Springer, 2009.

[BGHB11] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. Computer-aided security proofs for the working cryptographer. In *Advances in Cryptology–CRYPTO 2011*, pages 71–90. Springer, 2011.

[BGRV15] Johannes Borgström, Ramunas Gutkovas, Ioana Rodhe, and Björn Victor. The psi-calculi workbench: A generic tool for applied process calculi. *ACM Trans. Embedded Comput. Syst.*, 14(1):9:1–9:25, 2015.

[BGZB09] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. Formal certification of code-based cryptographic proofs. *ACM SIGPLAN Notices*, 44(1):90–101, 2009.

[BHM08] Michael Backes, Catalin Hritcu, and Matteo Maffei. Automated verification of remote electronic voting protocols in the applied pi-calculus. In *Proceedings of the IEEE Computer Security Foundations Symposium*, pages 195–209. IEEE, 2008.

[BJPV11] Jesper Bengtson, Magnus Johansson, Joachim Parrow, and Björn Victor. Psi-calculi: a framework for mobile processes with nominal data and logic. *Logical Methods in Computer Science*, 7(1), 2011.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[Bla04] Bruno Blanchet. Automatic proof of strong secrecy for security protocols. In *2004 Symposium on Security and Privacy*, pages 86–100. IEEE Computer Society Press, 2004.

[Bla08] Bruno Blanchet. A computationally sound mechanized prover for security protocols. *IEEE Transactions on Dependable and Secure Computing*, 5(4):193–207, 2008.

[BLF+14] Karthikeyan Bhargavan, Antoine Delignat Lavaud, Cédric Fournet, Alfredo Pironti, and Pierre Yves Strub. Triple handshakes and cookie cutters: Breaking and fixing authentication over TLS. In *2014 IEEE Symposium on Security and Privacy*, pages 98–113. IEEE, 2014.

[BMU08] Michael Backes, Matteo Maffei, and Dominique Unruh. Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol. In *Symposium on Security and Privacy*, pages 202–215. IEEE, 2008.

[Bor09] Johannes Borgström. A complete symbolic bisimilarity for an extended spi calculus. *Electr. Notes Theor. Comput. Sci.*, 242(3):3–20, 2009.

[BP03] Bruno Blanchet and Andreas Podelski. Verification of cryptographic protocols: Tagging enforces termination. In *Foundations of Software Science and Computation Structures*, volume 2620 of *LNCS*. Springer Verlag, 2003.

[BPKA15] Lennart Beringer, Adam Petcher, Q Ye Katherine, and Andrew W Appel. Verified correctness and security of OpenSSL HMAC. In *24th USENIX Security Symposium*, pages 207–221, 2015.

[Bru14] Mayla Brusó. *Dissecting Unlinkability*. PhD thesis, Technische Universiteit Eindhoven, 2014.

[BS16] Bruno Blanchet and Ben Smyth. Automated reasoning for equivalences in the applied pi calculus with barriers. In *Proc. 29th Computer Security Foundations Symposium*, 2016.

[BVQ10] Josh Benaloh, Serge Vaudenay, and Jean-Jacques Quisquater. Final report of iacr electronic voting committee. international association for cryptologic research, 2010.

[car] Blackhat'15 talk about cars hacking. `https://www.blackhat.com/us-15/briefings.html#remote-exploitation-of-an-unaltered-passenger-vehicle`. Accessed: 2016-01-05.

[CB13] Vincent Cheval and Bruno Blanchet. Proving more observational equivalences with ProVerif. In *Proc. 2nd Conference on Principles of Security and Trust*, volume 7796 of *LNCS*, pages 226–246. Springer, 2013.

[CC15] Vincent Cheval and Véronique Cortier. Timing attacks in security protocols: symbolic framework and proof techniques. In *Proc. 4th Conference on Principles of Security and Trust*, pages 280–299. Springer, 2015.

[CCCK16] Rohit Chadha, Vincent Cheval, Ştefan Ciobâcă, and Steve Kremer. Automated verification of equivalence properties of cryptographic protocol. *ACM Transactions on Computational Logic*, 2016. To appear.

[CCD10] Vincent Cheval, Hubert Comon-Lundh, and Stéphanie Delaune. Automating security analysis: symbolic equivalence of constraint systems. In *Proc. 5th International Joint Conference on Automated Reasoning*, volume 6173. Springer-Verlag, 2010.

[CCD11] Vincent Cheval, Hubert Comon-Lundh, and Stéphanie Delaune. Trace equivalence decision: Negative tests and non-determinism. In *Proceedings of Conference on Computer and Communications Security*. ACM Press, 2011.

[CCD13a] Vincent Cheval, Véronique Cortier, and Stéphanie Delaune. Deciding equivalence-based properties using constraint solving. *Theoretical Computer Science*, 492, 2013.

[CCD13b] Rémy Chrétien, Véronique Cortier, and Stéphanie Delaune. From security protocols to pushdown automata. In *Proc. 40th International Colloquium on Automata, Languages and Programming*, volume 7966 of *LNCS*, pages 137–149. Springer, 2013.

[CCD15a] Rémy Chrétien, Véronique Cortier, and Stéphanie Delaune. Decidability of trace equivalence for protocols with nonces. In *Proc. 28th Computer Security Foundations Symposium*, pages 170–184. IEEE Computer Society Press, 2015.

[CCD15b] Rémy Chrétien, Véronique Cortier, and Stéphanie Delaune. From security protocols to pushdown automata. *ACM Transactions on Computational Logic*, 17(1:3), September 2015.

[CCK12] Rohit Chadha, Ştefan Ciobâcă, and Steve Kremer. Automated verification of equivalence properties of cryptographic protocols. In *Programming Languages and Systems*, pages 108–127. Springer, 2012.

[CCLD16] Vincent Cheval, Hubert Comon-Lundh, and Stéphanie Delaune. A procedure for deciding symbolic equivalence between sets of constraint systems. *Information and Computation*, 2016. To appear.

[CCP13] Vincent Cheval, Véronique Cortier, and Antoine Plet. Lengths may break privacy – or how to check for equivalences with length. In *Proc. 25th International Conference on Computer Aided Verification (CAV'13)*, volume 8044 of *LNCS*, pages 708–723. Springer Berlin Heidelberg, 2013.

[CDL+99] Iliano Cervesato, Nancy A Durgin, Patrick D Lincoln, John C Mitchell, and Andre Scedrov. A meta-notation for protocol analysis. In *Computer Security Foundations Workshop, 1999. Proceedings of the 12th IEEE*, pages 55–69. IEEE, 1999.

[CDL06] Véronique Cortier, Stéphanie Delaune, and Pascal Lafourcade. A survey of algebraic properties used in cryptographic protocols. *Journal of Computer Security*, 14(1):1–43, 2006.

[CH17] Cas Cremers and Lucca Hirschi. Improving automatic symbolic analysis for e-voting protocols: Sufficient conditions for ballot secrecy. (Under submission) A copy can be found at `http://www.lsv.ens-cachan.fr/~hirschi/pdfs/CH-evoting.pdf`, 2017.

[Che12] Vincent Cheval. *Automatic verification of cryptographic protocols: privacy-type properties*. PhD thesis, École Normale Supérieure de Cachan, 2012.

[Che14] Vincent Cheval. Apte: an algorithm for proving trace equivalence. In *Proceedings TACAS'14*. Springer, 2014.

[Cho06] Tom Chothia. Analysing the mute anonymous file-sharing system using the pi-calculus. In *Formal Techniques for Networked and Distributed Systems-FORTE 2006*, pages 115–130. Springer, 2006.

[CHSvdM16] Cas Cremers, Marko Horvat, Sam Scott, and Thyla van der Merwe. Automated analysis and verification of TLS 1.3: 0-RTT, resumption and delayed authentication. In *IEEE Symposium on Security and Privacy*, 2016.

[CJM00] E. Clarke, S. Jha, and W. Marrero. Partial order reductions for security protocol verification. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 503–518. Springer, 2000.

[CJM03] Edmund M. Clarke, Somesh Jha, and Wilfredo R. Marrero. Efficient verification of security protocols using partial-order reductions. *International Journal on Software Tools for Technology Transfer*, 4(2), 2003.

[CKW11] Véronique Cortier, Steve Kremer, and Bogdan Warinschi. A survey of symbolic methods in computational analysis of cryptographic systems. *Journal of Automated Reasoning*, 46(3-4):225–259, 2011.

[CLD05] Hubert Comon-Lundh and Stéphanie Delaune. The finite variant property: How to get rid of some algebraic properties. In *Proc. International Conference on Rewriting Techniques and Applications*, pages 294–307. Springer, 2005.

[CLN12] Jan Camenisch, Anja Lehmann, and Gregory Neven. Electronic identities need private credentials. *IEEE Security & Privacy*, 10(1):80–83, 2012.

[CM05] Cas JF Cremers and Sjouke Mauw. Checking secrecy by means of partial order reduction. In *System Analysis and Modeling*. Springer, 2005.

[CMS] Kaustuv Chaudhuri, Dale Miller, and Alexis Saurin. Canonical sequent proofs via multi-focusing. In *International Conference On Theoretical Computer Science*.

[CMS10] Jan Camenisch, Sebastian Mödersheim, and Dieter Sommer. A formal model of identity mixer. In *Formal Methods for Industrial Critical Systems*, pages 198–214. Springer, 2010.

[CR12] Yannick Chevalier and Michael Rusinowitch. Decidability of symbolic equivalence of derivations. *Journal of Automated Reasoning*, 48(2), 2012.

[CRSv12] Cas Cremers, Kasper B. Rasmussen, Benedikt Schmidt, and Srdjan Čapkun. Distance Hijacking Attacks on Distance Bounding Protocols. In *33rd IEEE Symposium on Security and Privacy*, pages 113–127. IEEE Computer Society, May 2012.

[CS13] Véronique Cortier and Ben Smyth. Attacking and fixing Helios: An analysis of ballot secrecy. *Journal of Computer Security*, 21(1):89–148, 2013.

[CSD+12] Lassaad Cheikhrouhou, Werner Stephan, Özgür Dagdelen, Marc Fischlin, and Markus Ullmann. Merging the cryptographic security analysis and the algebraic-logic security proof of pace. In *Sicherheit*, pages 83–94, 2012.

[DCL14] Levent Demir, Mathieu Cunche, and Cédric Lauradoux. Analysing the privacy policies of wi-fi trackers. In *Proceedings of the 2014 workshop on physical analytics*, pages 39–44. ACM, 2014.

[DCS12] Yuxin Deng, Iliano Cervesato, and Robert J. Simmons. Relating reasoning methodologies in linear logic and process algebra. In *LINEARITY*, volume 101 of *EPTCS*, 2012.

[DDKS17] Jannik Dreier, Charles Duménil, Steve Kremer, and Ralf Sasse. Beyond Subterm-Convergent Equational Theories in Automated Verification of Stateful Protocols. In *In 6th International Conference on Principles of Security and Trust*, 2017. To appear.

[DDNM90] Pierpaolo Degano, Rocco De Nicola, and Ugo Montanari. A partial ordering semantics for ccs. *Theoretical Computer Science*, 75(3):223–262, 1990.

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *Transactions on Information Society*, 22(6):644–654, 1976.

[DH16] Stéphanie Delaune and Lucca Hirschi. A survey of symbolic methods for establishing equivalence-based properties in cryptographic protocols. *Journal of Logical and Algebraic Methods in Programming*, 2016.

[DJP12] Naipeng Dong, Hugo Jonker, and Jun Pang. Formal analysis of privacy in an ehealth protocol. In *Computer Security–ESORICS 2012*, pages 325–342. Springer, 2012.

[DKR06] Stephanie Delaune, Steve Kremer, and Mark Ryan. Coercion-resistance and receipt-freeness in electronic voting. In *Computer Security Foundations Workshop, 2006. 19th IEEE*, pages 12–pp. IEEE, 2006.

[DKR08] Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, (4), 2008.

[DKR10] Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. Symbolic bisimulation for the applied pi calculus. *Journal of Computer Security*, 18(2):317–377, March 2010.

[DRS08] Stéphanie Delaune, Mark D. Ryan, and Ben Smyth. Automatic verification of privacy properties in the applied pi-calculus. In *Proceedings of the 2nd Joint iTrust and PST Conferences on Privacy, Trust Management and Security (IFIPTM'08)*, volume 263 of *IFIP Conference Proceedings*. Springer, 2008.

[DS81] Dorothy E Denning and Giovanni Maria Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, 1981.

[DSV03] Luca Durante, Riccardo Sisto, and Adriano Valenzano. Automatic testing equivalence verification of spi calculus specifications. *ACM Transactions on Software Engineering and Methodology*, 12(2), 2003.

[DY83] Danny Dolev and Andrew Yao. On the security of public key protocols. *IEEE Transactions on information theory*, 29(2):198–208, 1983.

[EFF16] Eff article. `https://www.eff.org/deeplinks/2016/12/https-deployment-growing-leaps-and-bounds-2016-review`, 2016. Accessed: 2016-01-05.

[EMMS14] Santiago Escobar, Catherine Meadows, José Meseguer, and Sonia Santiago. State space reduction in the maude-nrl protocol analyzer. *Inf. Comput.*, 238:157–186, 2014.

[ESM10] Santiago Escobar, Ralf Sasse, and José Meseguer. Folding variant narrowing and optimal variant termination. In *Rewriting Logic and Its Applications*, pages 52–68. Springer, 2010.

[FDW04] Martin Feldhofer, Sandra Dominikus, and Johannes Wolkerstorfer. Strong authentication for RFID systems using the AES algorithm. In *Cryptographic Hardware and Embedded Systems-CHES 2004*, pages 357–370. Springer, 2004.

[FDW10] Wan Fokkink, Mohammad Torabi Dashti, and Anton Wijs. Partial order reduction for branching security protocols. In *Proceedings of ACSD'10*. IEEE, 2010.

[FG05] Cormac Flanagan and Patrice Godefroid. Dynamic partial-order reduction for model checking software. In *ACM Sigplan Notices*, volume 40, pages 110–121. ACM, 2005.

[FJHG99] THAYER Fabrega, F Javier, Jonathan C Herzog, and Joshua D Guttman. Strand spaces: Proving security protocols correct. *Journal of computer security*, 7(2-3):191–230, 1999.

[GDP] GDPR. `http://ec.europa.eu/justice/data-protection/reform/index_en.htm`. Accessed: 2016-01-05.

[God91] Patrice Godefroid. Using partial orders to improve automatic verification methods. In *Computer-Aided Verification*, pages 176–185. Springer Berlin Heidelberg, 1991.

[GP05] Deepak Garg and Frank Pfenning. Type-directed concurrency. In *Proceedings of the International Conference on Concurrency Theory*, volume 3653 of *Lecture Notes in Computer Science*. Springer, 2005.

[GRCC15] Gurchetan S Grewal, Mark D Ryan, Liqun Chen, and Michael R Clarkson. Du-vote: Remote electronic voting with untrusted computers. In *2015 IEEE 28th Computer Security Foundations Symposium*, pages 155–169. IEEE, 2015.

[GvLH+96] Patrice Godefroid, J van Leeuwen, J Hartmanis, G Goos, and Pierre Wolper. *Partial-order methods for the verification of concurrent systems: an approach to the state-explosion problem*, volume 1032. Springer Heidelberg, 1996.

[HBD16] Lucca Hirschi, David Baelde, and Stéphanie Delaune. A method for verifying privacy-type properties: the unbounded case. In *37th Symposium on Security and Privacy (Oakland'16)*, pages 564–581. IEEE, 2016.

[hel] Princeton election server. `https://princeton.heliosvoting.org/`. Accessed: 2016-01-05.

[Hira] L. Hirschi. APTE with POR. `http://www.lsv.ens-cachan.fr/~hirschi/apte_por`. Accessed: 2016-01-05.

[Hirb] Lucca Hirschi. SPEC with dependency constraints. `http://www.lsv.ens-cachan.fr/~hirschi/spec.php`. Accessed: 2016-01-05.

[Hir13] Lucca Hirschi. Réduction d'entrelacements pour l'équivalence de traces. RR LSV-13-13, Laboratoire Spécification et Vérification, ENS Cachan, France, September 2013.

[HNW98] Michaela Huhn, Peter Niebert, and Heike Wehrheim. Partial order reductions for bisimulation checking. In Vikraman Arvind and Ramaswamy Ramanujam, editors, *Foundations of Software Technology and Theoretical Computer Science, 18th Conference, Chennai, India, December 17-19, 1998, Proceedings*, volume 1530 of *Lecture Notes in Computer Science*, pages 271–282. Springer, 1998.

[Hüt03] Hans Hüttel. Deciding framed bisimilarity. *Electronic Notes in Theoretical Computer Science*, 68(6):1–18, 2003.

[ICA04] PKI for machine readable travel documents offering ICC read-only access. Technical report, International Civil Aviation Organization, 2004.

[ISO09] Iso 15408-2: Common criteria for information technology security evaluation - part 2: Security functional components, July 2009.

[JW09] Ari Juels and Stephen A Weis. Defining strong privacy for RFID. *ACM Transactions on Information and System Security*, 13(1):7, 2009.

[K+16] Steve Kremer et al. To du or not to du: A security analysis of du-vote. In *2016 IEEE European Symposium on Security and Privacy*, pages 473–486. IEEE, 2016.

[KR05] S. Kremer and M. D. Ryan. Analysis of an electronic voting protocol in the applied pi-calculus. In *Proc. 14th European Symposium on Programming*, volume 3444 of *LNCS*, pages 186–200. Springer-Verlag, 2005.

[LAK06] Sangshin Lee, Tomoyuki Asano, and Kwangjo Kim. RFID mutual authentication scheme based on synchronized secret information. In *Symposium on cryptography and information security*, 2006.

[log] Logjam website. `https://mitls.org/pages/attacks/Logjam`. Accessed: 2016-01-05.

[Low97] Gavin Lowe. A hierarchy of authentication specifications. In *Proc. 10th Computer Security Foundations Workshop*, pages 18–30. IEEE Computer Society Press, 1997.

[Mil03] Dale Miller. Encryption as an abstract data type. *Electr. Notes Theor. Comput. Sci.*, 84, 2003.

[MNP02] M.Boreale, R. De Nicola, and R. Pugliese. Proof techniques for cryptographic processes. *SIAM Journal on Computing*, 31(3):947–986, 2002.

[MS92] R. Milner and D. Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *Proc. 19th International Colloquium on Automata, Languages, and Programming*, volume 623 of *LNCS*, pages 685–695. Springer Verlag, 1992.

[MS01] J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proceedings of Conference on Computer and Communications Security*. ACM Press, 2001.

[MS07] Dale Miller and Alexis Saurin. From proofs to focused proofs: A modular proof of focalization in linear logic. In *Proceedings of the EACSL Annual Conference on Computer Science Logic*, volume 4646. Springer, 2007.

[MSCB13] S. Meier, B. Schmidt, C. Cremers, and D. Basin. The Tamarin Prover for the Symbolic Analysis of Security Protocols. In *Proc. 25th International Conference on Computer Aided Verification*, volume 8044 of *LNCS*, pages 696–701. Springer, 2013.

[MSDL99] J Mitchell, A Scedrov, N Durgin, and P Lincoln. Undecidability of bounded security protocols. In *Workshop on Formal Methods and Security Protocols*, 1999.

[MVB10] Sebastian Mödersheim, Luca Viganò, and David A. Basin. Constraint differentiation: Search-space reduction for the constraint-based analysis of security protocols. *Journal of Computer Security*, 18(4), 2010.

[nava] Navizon website. `https://www.navizon.com/`. Accessed: 2016-01-05.

[navb] Shops can track you via your smartphone, privacy watchdog warns. https://www.theguardian.com/technology/2016/jan/21/shops-track-smartphone-uk-privacy-watchdog-warns. Accessed: 2016-01-05.

[NS78] Roger M Needham and Michael D Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[PAC] Technical advisory group on machine readable travel documents (tag/mrtd). http://www.icao.int/Meetings/TAG-MRTD/TagMrtd22/TAG-MRTD-22_WP05.pdf. Accessed: 2016-01-05.

[PAdM10] Olivier Pereira, Ben Adida, and Olivier de Marneffe. Bringing open audit elections into practice: Real world uses of Helios. swiss e-voting workshop, 2010.

[Pel98] Doron Peled. Ten years of partial order reduction. In *Proceedings of International Conference on Computer-Aided Verification*, volume 1427. Springer, 1998.

[PM15] Adam Petcher and Greg Morrisett. The foundational cryptography framework. In *Principles of Security and Trust*, pages 53–72. Springer, 2015.

[PvdM16] Kenneth G Paterson and Thyla van der Merwe. Reactive and proactive standardisation of TLS. In *Security Standardisation Research*, pages 160–186. Springer, 2016.

[PZ13] Christian Paquin and Greg Zaverucha. U-prove cryptographic specification v1.1 (revision 3), December 2013.

[RSA78] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[SA06] Koushik Sen and Gul Agha. Automated systematic testing of open distributed programs. In *Fundamental Approaches to Software Engineering*, pages 339–356. Springer, 2006.

[SEMM14] Sonia Santiago, Santiago Escobar, Catherine Meadows, and José Meseguer. A formal definition of protocol indistinguishability and its verification using maude-npa. In *Security and Trust Management*, pages 162–177. Springer, 2014.

[SRC15] Ben Smyth, Mark D Ryan, and Liqun Chen. Formal analysis of privacy in direct anonymous attestation schemes. *Science of Computer Programming*, 111:300–317, 2015.

[SS96] Steve Schneider and Abraham Sidiropoulos. CSP and anonymity. In *Proc. International Conference on Computer Security*, pages 198–218. Springer, 1996.

[SSB+16] Altaf Shaik, Jean-Pierre Seifert, Ravishankar Borgaonkar, N. Asokan, and Valtteri Niemi. Practical attacks against privacy and availability in 4G/LTE mobile communication systems. In *23nd Annual Network and Distributed System Security Symposium*, 2016.

[SW03] Davide Sangiorgi and David Walker. *The pi-calculus: a Theory of Mobile Processes*. Cambridge university press, 2003.

[TD10] Alwen Tiu and Jeremy E. Dawson. Automating open bisimulation checking for the spi calculus. In *Proceedings of the IEEE Computer Security Foundations Symposium*. IEEE Computer Society Press, 2010.

[Tiu07] Alwen Tiu. A trace based bisimulation for the spi calculus. In *Programming Languages and Systems*, pages 367–382. Springer, 2007.

[TKL+12] Samira Tasharofi, Rajesh K Karmani, Steven Lauterburg, Axel Legay, Darko Marinov, and Gul Agha. Transdpor: A novel dynamic partial-order reduction technique for testing actor programs. In *Formal Techniques for Distributed Systems*, pages 219–234. Springer, 2012.

[Tur17] Joseph Turow. *The Aisles Have Eyes*. Yale University Press, 2017.

[UKA] UKano official webpage. `http://projects.lsv.ens-cachan.fr/ukano/`. Accessed: 2016-01-05.

[UKe] UKano case studies. `https://github.com/LCBH/UKano/wiki#our-case-studies`. Accessed: 2016-01-05.

[vdBVdR15] Fabian van den Broek, Roel Verdult, and Joeri de Ruiter. Defeating IMSI Catchers. *Proceedings of the 2015 ACM Conference on Computer and Communications Security - CCS'15*, 2015.

[VDMR08] Ton Van Deursen, Sjouke Mauw, and Saša Radomirović. Untraceability of RFID protocols. In *Information Security Theory and Practices. Smart Devices, Convergence and Next Generation Networks*, pages 1–15. Springer, 2008.

[VDR08] Ton Van Deursen and Sasa Radomirovic. Attacks on RFID protocols. *IACR Cryptology ePrint Archive*, 2008:310, 2008.

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

**Titre :** Vérification automatique de la protection de la vie privée : entre théorie et pratique

**Mots clefs :** Protocoles de sécurité, vérification, vie privée, méthodes formelles

**Résumé :** Notre société de l'information s'appuie de façon cruciale sur notre capacité à échanger des données protégées par des protocoles cryptographiques. Étant donné leur prédominance et leur importance, il est important de garantir qu'ils accomplissent leurs objectifs, tels que la protection de la vie privée. Idéalement, ces garanties sont obtenues par des méthodes formelles qui, via leurs fondements mathématiques, permettent une analyse rigoureuse. Seulement, ces méthodes sont limitées par des problèmes de passage à l'échelle et de précision que nous nous proposons de résoudre dans cette thèse.
Pour le premier problème, nous développons des méthodes de réduction d'ordre partiel pour réduire drastiquement l'espace d'exploration. Pour le second, nous proposons une nouvelle approche de vérification. Nous définissons deux propriétés facilement vérifiables qui impliquent la non-traçabilité et l'anonymat.
Nous montrons l'impact pratique important de nos travaux via leur implémentation et leur application à des protocoles industriels menant à la découverte de nouvelles attaques et de preuves de sécurité.

**Title :** Automated Verification of Privacy in Security Protocols: Back and Forth Between Theory & Practice

**Keywords :** Security Protocols, verification, privacy, formal methods

**Abstract :** Our information society notably relies on secure information exchanges typically achieved by security protocols. Given, their ubiquitous and critical nature, we need guarantees that they meet their goals such as privacy properties. Ideally, those guarantees are established via formal methods providing mathematical frameworks to analyse security protocols. However, existing methods suffer from scalability and precision issues that the present thesis aims to address.
First, to mitigate the scalability problem, we develop partial order reduction techniques enabling to dramatically reduce the search space to explore when analysing security protocols. Second, we solve a critical precision issue by adopting a privacy via sufficient conditions approach. We show that two well-designed and easily verifiable conditions always imply unlinkability and anonymity.
We confirm the practical relevance of our contributions by implementing them and using them for analysing real-life security protocols, finding new attacks and establishing new proofs.

**Université Paris-Saclay**
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France