UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Efficient verification of observational equivalences of cryptographic processes : theory and practice

THÈSE

présentée et soutenue publiquement le ***01/02/2021***

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Itsaka Rakotonirina

Composition du jury

Rapporteurs :	David Basin	ETH Zurich
	Tamara Rezk	Inria Sophia Antipolis
Examinateurs :	Myrto Arapinis	University of Edinburgh
	Vincent Cheval	Inria Nancy — *co-directeur de thèse*
	Thomas Jensen	Inria Rennes
	Steve Kremer	Inria Nancy — *directeur de thèse*
	Catuscia Palamidessi	Inria Saclay

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

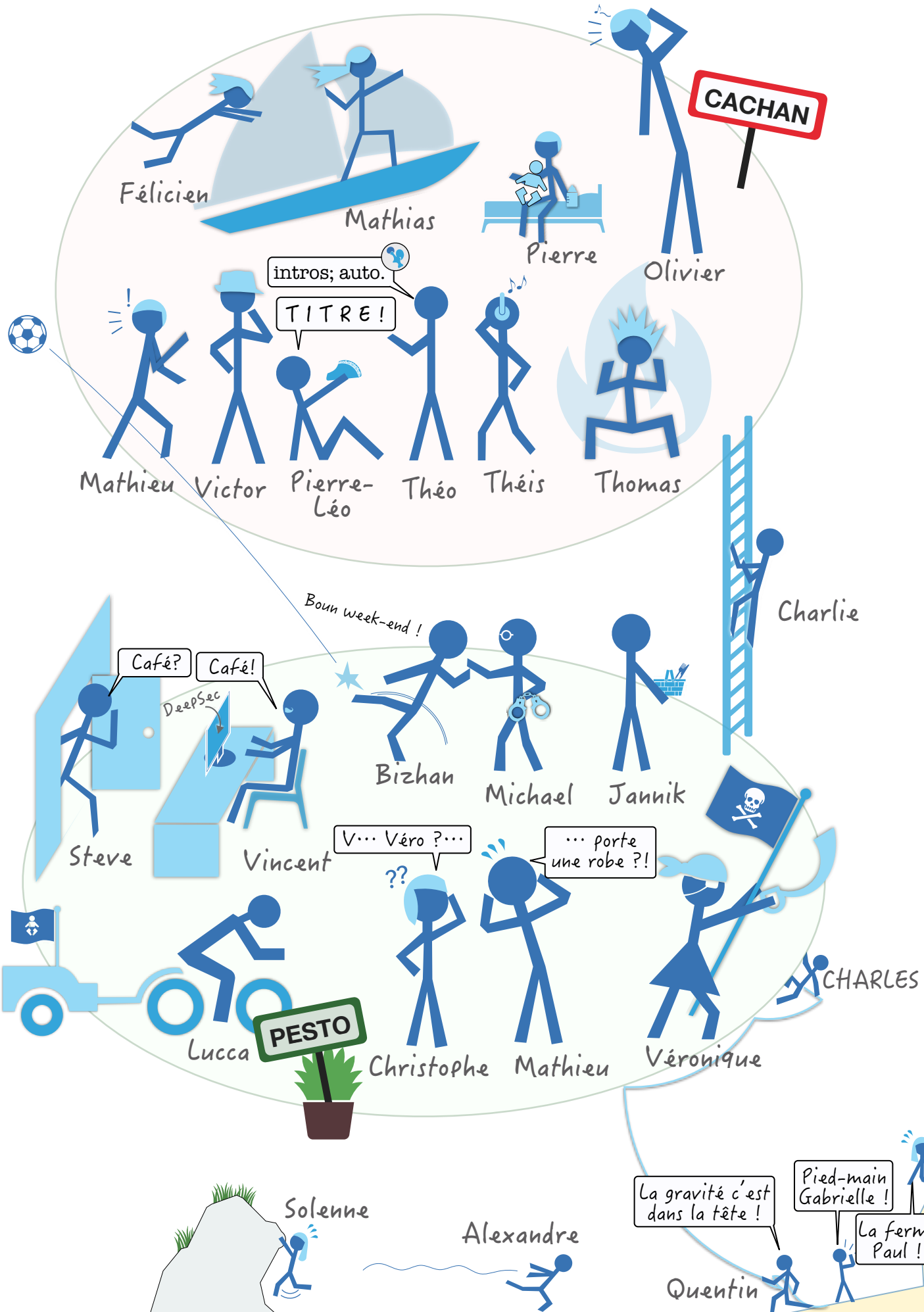
Acknowledgements / Remerciements

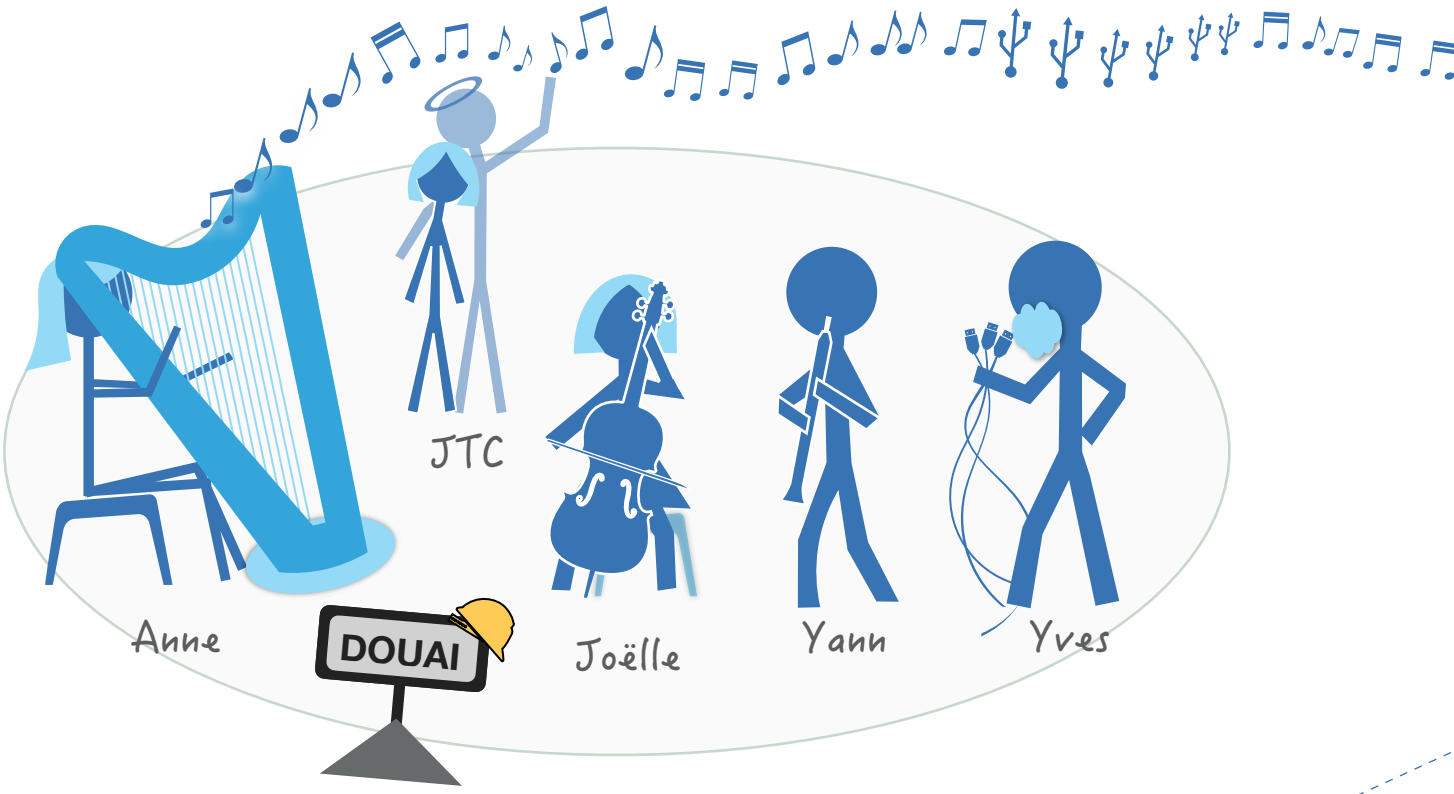
I have many persons to thank for their help and support, and I will not be able to mention all of them. I trust that all will know that I am thankful for our interactions and that they kept me moving forward! I drew some pictures[1] to convey my feelings to those I interacted the most with during these three years — I hope they will appreciate it.

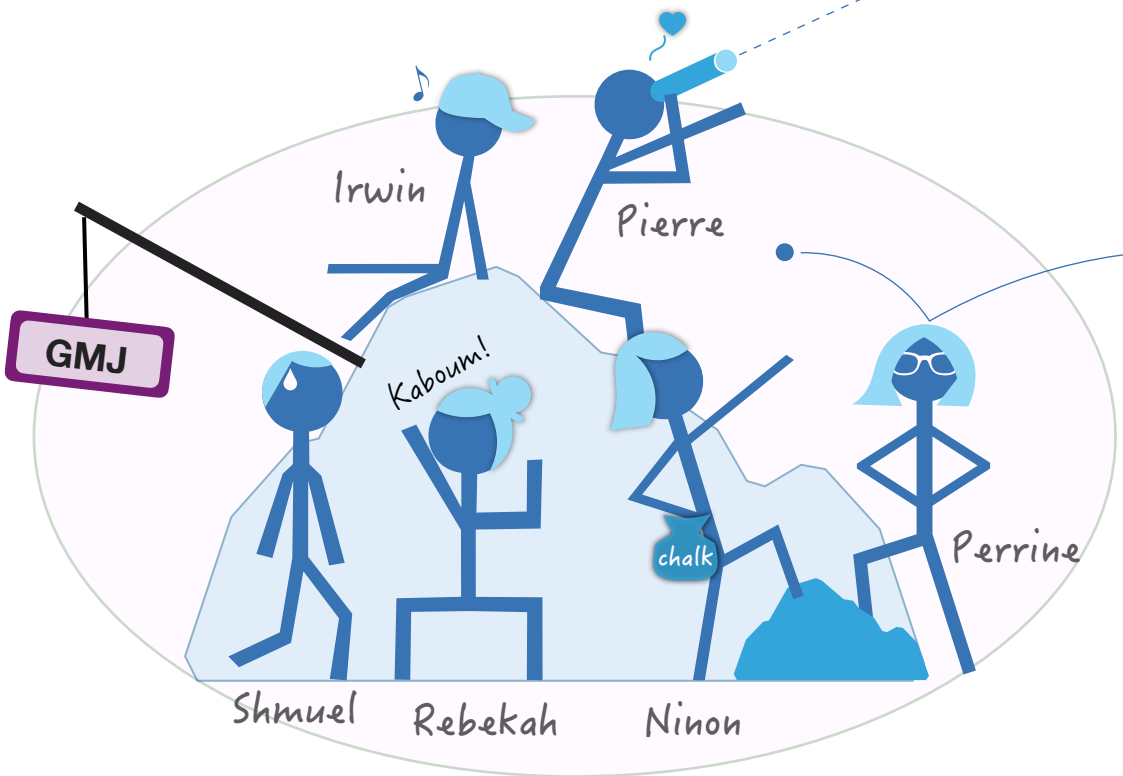
I also thank the jury[2] for the time and effort they spent to read my manuscript, and for the resulting insightful comments. On a more formal tone, the research leading to these results has received funding from the ERC under the EU's H2020 research and innovation program (grant agreements No 645865-SPOOC). I also had the opportunity to earn a fellowship from Google during this thesis, I also thank them for the resulting financial support to the team project.

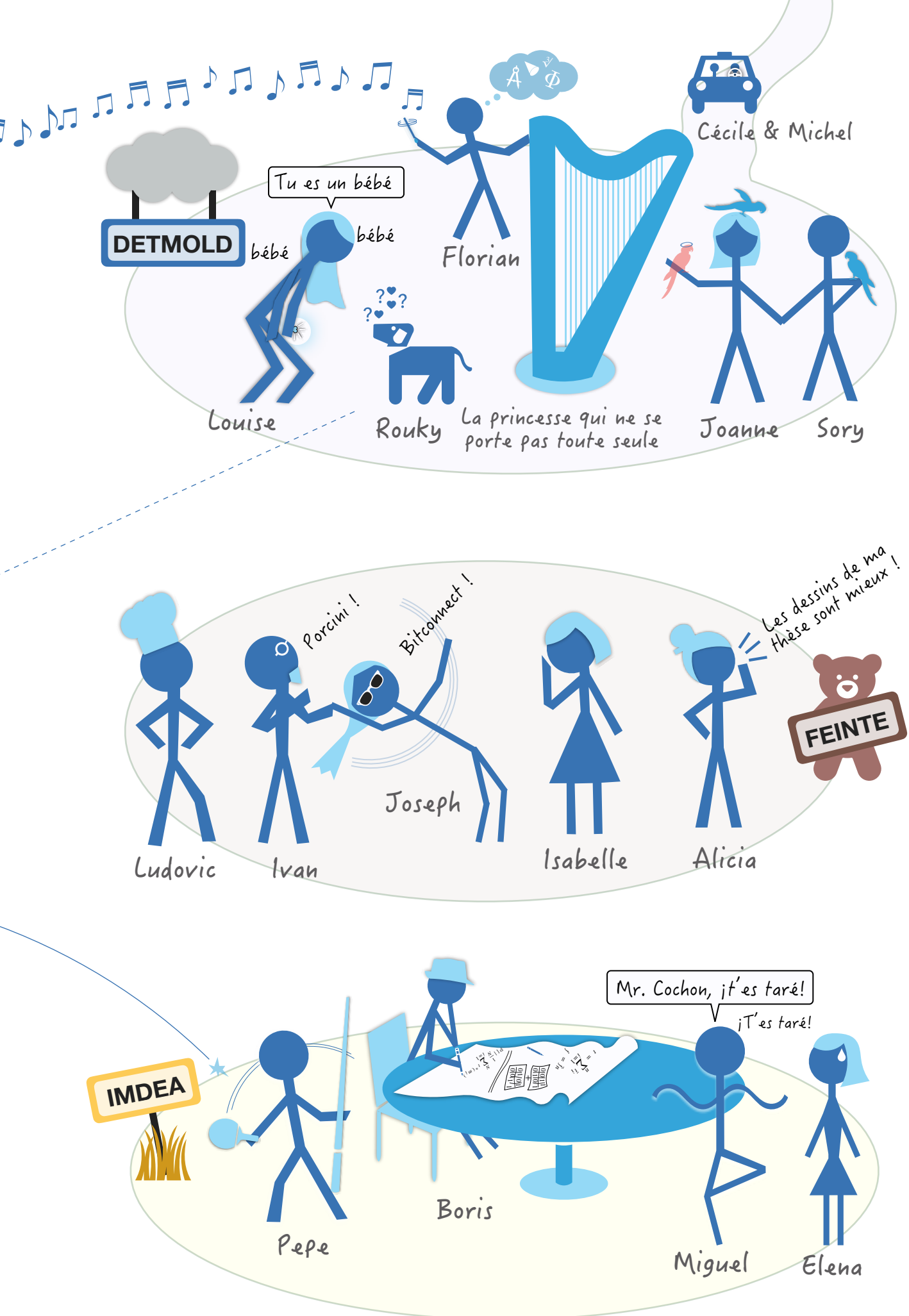
[1] using a presentation software, because making slides is the only way I have to draw something decent

[2] and all people that helped me proofread my thesis!


Félicien
Mathias
Pierre
CACHAN
Olivier
intros; auto.
TITRE !
Mathieu
Victor
Pierre-Léo
Théo
Théis
Thomas
Boun week-end !
Charlie
Café?
Café!
DeepSec
Steve
Vincent
Bizhan
Michael
Jannik
V··· Véro ?···
??
··· porte une robe ?!
PESTO
Lucca
Christophe
Mathieu
Véronique
CHARLES
Solenne
Alexandre
La gravité c'est dans la tête !
Pied-main Gabrielle !
La ferme Paul !
Quentin


JTC
Anne
DOUAI
Joëlle
Yann
Yves


Irwin
Pierre
GMJ
Kaboum!
chalk
Perrine
Shmuel
Rebekah
Ninon


DETMOLD
Tu es un bébé
bébé
bébé
Florian
Cécile & Michel
Louise
Rouky
La princesse qui ne se porte pas toute seule
Joanne
Sory
Porcini !
Bitconnect !
Joseph
Ludovic
Ivan
Isabelle
Alicia
Les dessins de ma thèse sont mieux !
FEINTE
IMDEA
Pepe
Boris
Mr. Cochon, ¡t'es taré!
¡T'es taré!
Miguel
Elena

Contents

General introduction

We motivate and introduce the research problem studied in this thesis, namely the *automated verification of security protocols*. But before anything, let us solve a little riddle which, surprisingly, nails down many key questions of this research field despite its playful tone; we will explicit the link with our contributions after that.

1 The greedy delivery guy riddle

Bob would like to buy a cake from a renowned backery run by Isabelle. The shop is too far away for him to go in person and he thus asks for a delivery. Unfortunately the delivery guy is notorious for being greedy: he tends to compulsively eat the cakes he should deliver during his errands. This is a tricky situation: Isabelle and Bob *need the delivery guy* but are also aware of the fact that *he cannot be trusted*.

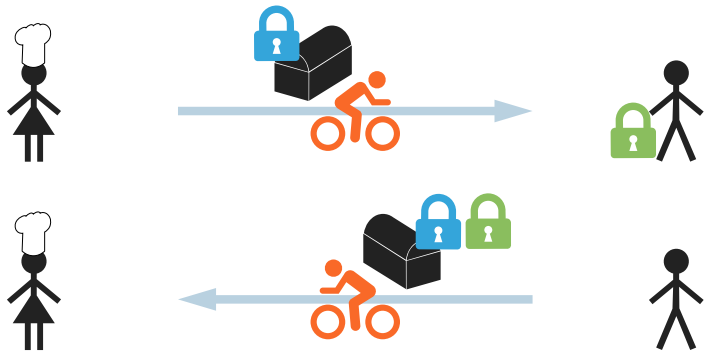
After discussing the issue by phone, they realise that they both have *locks* at home, meaning that whenever they would need to rely on this untrusted postal service, they could put deliveries in locked chests to mitigate the risk of theft.

Question. How can we transfer the cake from Isabelle's bakery to Bob's house safely?

1.1 For the sake of a cake

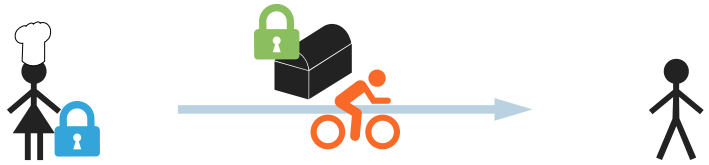
The difficulty of the riddle is that, although Isabelle and Bob each have a lock, neither of them can open the other's. Isabelle could try to lock the chest and later send the key to Bob; however the delivery guy may then end up with both the locked chest and the key and could therefore open the former. A solution has to be found where both Isabelle and Bob keep their keys close to them.

A first solution A possible workaround is to combine the two locks as follows. First, Isabelle puts the cake in a chest, locks it, and asks the delivery guy to bring it to Bob:



So far the delivery guy cannot eat the cake but Bob cannot open the chest either. Then Bob adds his own lock to the chest, so that it is locked with both, and sends it back to Isabelle:

Upon receiving the chest, Isabelle removes her own lock from it and sends it back:



Bob hence receives the cake only sealed by his own lock and can therefore recover it. No keys were sent during the process and there was always at least one layer of lock on the different chests given to the delivery guy. Therefore we have:

> ***Property*** *(security of the cake-delivery protocol)*
> The delivery guy cannot eat a cake Isabelle is willing to send to Bob using this protocol.

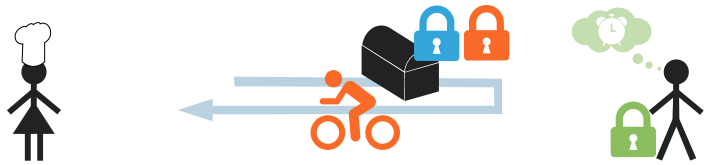
... but is it really true? There are actually many ways for the delivery guy to obtain the cake, the most straightforward one being to force the locks with the appropriate tools. The above property therefore relies on implicit assumptions about the delivery guy:

1. *perfect lock security*: he cannot break the locks or make the cake ooze through potential cracks in the chest (which permits to eat part of the cake without breaking anything).
2. *key incorruptibility*: he cannot sneak into Isabelle's or Bob's houses to steal the keys.
3. *passivity*: despite his greed, he never lies to the recipient about the content of the chest, that is, he follows the flow of the protocol without actively disrupting it.

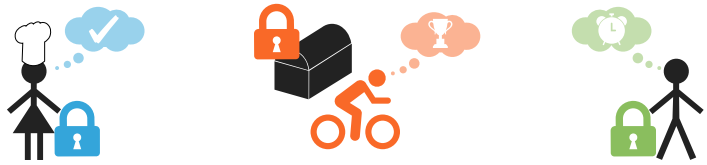
Without these these three assumptions, the security goal is *a priori* not achieved. But are they realistic? Is it possible to protect the cake even without (some of) them?

Against an active delivery guy We suppose that the locks and the chest are of very high quality and that the keys are well guarded, which means we keep the first two assumptions. In other terms we focus on the *logical structure* of the protocol rather than the solidity of security equipment. But even then its security is quite weak: it heavily relies on the passivity assumption which may be unrealistic if, for instance, Isabelle's cakes are worth a very high price. A delivery guy could actively try to disrupt the protocol's security by, for example, buying his own lock and stealing the cake as follows. First, Isabelle locks her cake in the chest

as usual. But then the delivery guy deviates from his instructions, not delivering the chest to Bob and putting his own lock on it instead, and bringing it back to Isabelle:



Unable to tell that this is not Bob's lock, Isabelle thinks the protocol is proceeding fine. She therefore removes her lock from the chest and asks the delivery guy to bring it to Bob:



The final situation is then the following: Isabelle did not notice any problem and the delivery guy is able to unlock the chest with his own key. This *attack* on the cake's security did not require to force any lock, it simply exploits a breach in the structure of the protocol.

Patching the protocol Designing a theft-resistant protocol in the context of an active delivery guy is actually close to impossible with this system of locks. Isabelle and her clients therefore decide to subscribe to a large-scale organisation of *public locks*:

1. anyone in the organisation is given a *personal key*;
2. anyone can request a lock that can only be opened by the key of *one specific member*.

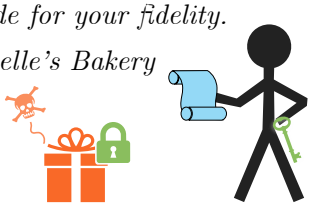
A simple cake-delivery protocol (V2) thus consists of Isabelle putting her cake in a chest and using a lock that can only be opened by Bob (obtained through the organisation). Assuming the locks of the organisation have the claimed properties, we have this time:

> ***Property*** *(security of the cake-delivery protocol V2)*
> An active delivery guy cannot eat a cake Isabelle is willing to send to Bob using V2.

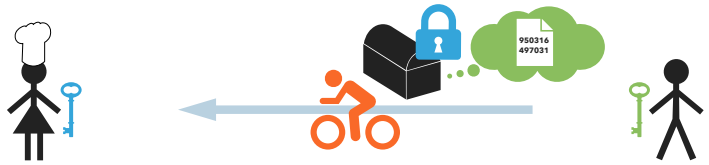
1.2 A darker riddle

Impersonation This is however not the end of the story. Knowing that Bob likes Isabelle's cakes, an enemy of him requests a lock that can only be opened by Bob, and uses it to send him a poisoned cake with the message "*We offer you this as a token of gratitude for your fidelity. Isabelle's backery*". Bob eats it which reveals one weakness of the protocol: we can never be sure that the sender of a delivery is the one they pretend to be. After Bob returns from the hospital, he and Isabelle decide to update the cake-delivery protocol as follows to provide a form of *authentication*.

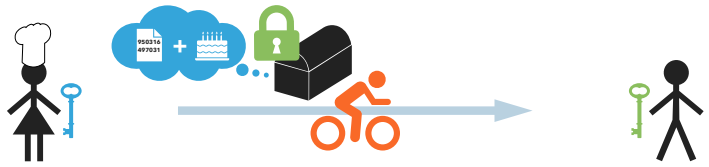
We offer you this as a token of gratitude for your fidelity.
— Isabelle's Bakery



We assume that both Isabelle and her client (Bob here) have obtained, from the organisation, locks than can only be opened by the other (referred to as *the client's lock* and *Isabelle's lock*, respectively). This time the client initiates the protocol. He writes on a piece of paper a (long) password of his choice that will serve to identify his command. Then he sends this paper to Isabelle inside a chest sealed with her lock, and joins a note to the locked chest indicating his name so that Isabelle can know who sending the delivery back to:



Upon receiving the chest, Isabelle reads the name of the client, opens the chest with her key, puts the cake inside next to the password paper, and eventually seals everything again with the client's lock. Then she sends everything back to the client:



When the client receives the delivery, he checks that the cake is accompanied by the password that he put initially. If it is not the case he remains careful and destroys the cake in case it is poisoned. If we call V3 this new version of the protocol, we would like to have:

> ***Expected property*** *(security of the cake-delivery protocol V3)*
> An active delivery guy cannot eat a cake Isabelle is willing to send to Bob using V3, nor can he poison Bob if Bob is only willing to accept cakes he thinks are from Isabelle.

Intuitively the security relies on the fact that, provided the password chosen by Bob is long and unpredictable enough, if someone wants to send him a poisoned cake by pretending to be Isabelle, there is only a negligible probability that he will manage to guess the password by luck (and therefore to make Bob eat the poisoned cake). However, the situation is steadily getting out of control: the protocol is starting to involve many security mechanisms and since it is difficult to predict an active delivery guy's behaviour, it is becoming increasingly hard to convince ourselves that security holds—that is, that we have not overlooked one among the infinitely-many potential courses of action.

Automated analysis Regarding the current situation, informally arguing for the security of the protocol cannot be satisfying—we would be likely to forget a potential attack scenario. Instead, we rely on two principles:

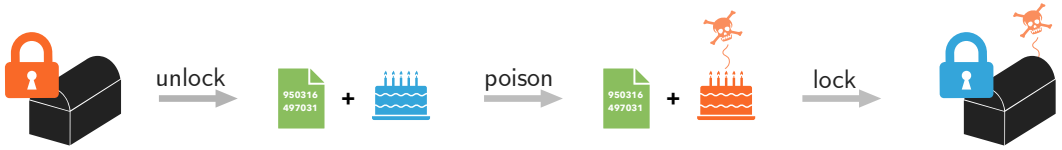
1. *A rigorous reasoning*, expliciting the hypotheses of the situation and carefully *proving* that security holds like a mathematical theorem.
2. *A mechanised proof*: treating one attack scenario is feasible by hand but reasoning mistakes and missing scenarios are to be expected due to fatigue when treating hundreds of them. Automating part of the proof process mitigates this stamina issue and its consequences.

There exist many automated tools that would permit to study the security of the cake delivery protocol. We develop one in this thesis: the *DeepSec prover*. We will detail later the technical features, novelties and limitations of the tool and how it compares to other similar analysers; for now let us simply give the conclusions we obtained by using it to analyse V3:

1 The protocol V3 is secure provided the delivery guy is not one of the clients of the backery (and has no accomplices among them).
2 If the delivery guy is a client or has an accomplice, he can poison other clients.

For the second point, the attack found by DeepSec can be described as follows. The protocol starts as usual with Bob choosing a password for his command to Isabelle, putting it in a locked chest, and writing his name on a note attached to the order. In parallel the delivery guy also purchases a cake to be picked later at the backery. He then forwards Bob's command to Isabelle *but* replaces Bob's name by his own:

Isabelle follows the instruction of the protocol and removes her lock, puts the cake in the chest (thinking that the password comes from the delivery guy), and locks the chest *with the delivery guy's lock*. He can therefore open it, poison the cake, and relock it with Bob's lock.



Eventually he comes back to Bob and gives him the poisoned delivery, who believes it to be safe due to the presence of his password and eats the poisoned cake. Therefore:

> ***Property*** *((in)security of the cake delivery protocol V3)*
> If the delivery guy has an accomplice among Isabelle's clients, he can poison Bob even if Bob and Isabelle use V3 and Bob is only willing to accept cakes from Isabelle.

A natural patch is to make the client put his name *inside* the chest, next to the password, to prevent the delivery guy from modifying it. If we call V4 this final version of the protocol, DeepSec managed to prove in a few seconds (on a personal computer) that there existed no attack scenarios involving up to 9 parallel *roles* (one role being either a client ordering a cake or Isabelle answering to a command). Although this does not exclude potential attacks involving more roles, we observe that attacks in real protocols usually require no more than 3 or 4 roles, giving worth to this security guarantee. Isabelle can now send a gift cake to Bob by requesting him to initiate a session of the

We offer you a cake as a token of gratitude for your fidelity. You can get it by initiating a session of V4 with us.
— Isabelle's Bakery

protocol with her backery. Naturally the delivery guy could do a *phishing attack*, the favourite one of email spammers, by sending a fake gift message "*You can get your free cake by initiating a session of V4 with us! (use our lock attached to this message). Isabelle's backery*" and joining his own lock to the letter. Our security statements hence assume that Bob always sticks to the protocol, that is, that he only uses locks provided by the organisation as required.

2 Verification of security protocols

2.1 Link with the riddle

Although this thesis has not been primarily motivated by sending cakes, it has some common ground with the story of Isabelle and Bob. We study *security protocols* that are sequences of instructions concurrently executed by different entities sending and receiving data to each others. These protocols include our everday connections to the 3/4/5G network but also online payment, electronic voting, or even the RFID chips on electronic passports. All these examples share two important characteristics:

1. the data sent during communications is *sensitive*, a lack of confidentiality or integrity has economical or political consequences (think about online payments or electronic elections);
2. the protocols are operated over *untrusted networks* such as the Internet: anyone could use an antenna to spy on communications, or even interfere with them.

In particular, just as in the riddle, security protocols are designed with the idea in mind that a message sent from an entity A to an entity B may be *intercepted or modified* during its transmission through the network; this is often modelled by a notion of *attacker* or *adversary*, that is, an abstract entity controlling the network. In the riddle, the attacker is the delivery guy as he is the one to decide how messages are distributed and may modify them to actively disrupt the protocol's flow to his advantage. The sensitive assets to be protected from him are then the cake and its integrity (i.e., the fact that clients do not accept cakes that have been poisoned during the delivery).

The chests and locks that Isabelle and Bob use to impede the actions of the adversary are then the analogue of *cryptographic primitives*, that are mathematical tools used to secure communications. For example the public locks of Isabelle and Bob correspond to *public-key encryption* which intuitively makes a message unintelligible to anyone except one chosen person. The locks used in the first variant of the riddle (where the locks are not publicly available) rather recall *symmetric encryption*, where encryption, i.e., the mechanism rendering a message unintelligible, also requires to possess the key.

2.2 Takeaway message(s)

Although security protocols provide significant benefits to many users—for instance electronic voting can ease the access to democracy to disabled people, foreign nationals or militaries in field operations—computer communications also tend to increase the *scalability of vulnerabilities*, giving rise to spectacular cyberattacks. Due to the sensitive nature of the information (economical, political, medical...) and the huge number of users, this makes it critical to ensure that security protocols are not breached. In particular, looking back at the riddle, we can exhibit the following core security principles:

▷ *Takeaway 1: identifying the threat model is crucial.*

As we have seen, considering realistic capabilities for the adversary is important: is he passive or active? Is he able to break the cryptographic primitives? Can he compromise keys (by breaking into Bob's house for example)? Does he have accomplices among the other participants? This has direct consequences on whether the protocol will be deemed secure or not and, subsequently, impacts the interpretation of a potential conclusion that it is so.

▷ *Takeaway 2: analysing security protocols is complex.*

Due to the fact that many participants operate in parallel, depending on the order they act, we obtain different executions of protocol (for example in the riddle, delivering and treating several commands in a different order yields a different flow of messages). There are exponentially many such *interleavings* of the participant actions. Adding the arbitrary interferences of the active adversary to the equation, security analysis becomes a non-trivial problem even for simple examples such as the cake delivery protocol (where each participant only has 1 or 2 instructions to execute) with a fixed number of participants (like we did when analysing V4 with DeepSec). Scaling to real-world case studies requires to consider much more involved scenarios and is therefore particularly challenging.

Besides there is an inherent asymmetry between the goal of the adversary (finding *at least one flaw* of the protocol to exploit) and the goal of the protocol designers (avoiding *all flaws* of the protocol) which makes the task of the latter harder.

▷ *Takeaway 3: automated tools can help.*

As shown by the attack on V3 found by DeepSec, the subtle weak points of protocols are more easily discovered by automated tools as they systematically consider all possible scenarios. On the contrary when a tool successfully produces a security proof, we should always keep in mind the hypotheses of the model when interpreting the result (in the riddle for example, we only focused on weaknesses of the logical structure of the protocol by assuming the chest-lock system to be unbreakable).

▷ *Takeaway 4: particular care is needed when modelling the protocol and security properties.*

How we describe the protocol and the security property to be verified can also induce some implicit hypotheses. In the riddle for example, one may easily forget to model the possibility that a client could be an accomplice of the attacker, which would have made us miss a security breach in the case of V3. More generally, overlooking some capabilities of the adversary, restricting the attack scenario, or unconsciously assuming that some protocol participants are honest are rather common pitfalls when studying security protocols.

2.3 In this thesis: symbolic verification of equivalence properties

Symbolic models In this thesis we are interested in the analysis of security protocols relying on *symbolic models*, that are representations of the protocols idealising the properties of some cryptographic mechanisms in play. For example in the riddle, we assumed the chests and locks to be unbreakable and the client passwords to be unguessable; but of course it is not strictly impossible that the delivery guy has tools powerful enough to break the locks, or manages to guess a 100-character password by luck. Our approach therefore searches for vulnerabilities in the *logical structure* of the protocol and overlooks those caused by the imperfection of cryptography or implementation details. Typically, the chest-lock-key system

is simply defined by its *functional properties*, that is, by the fact that a lock can be opened by the corresponding key. We express this by a *rewrite rule* written

$$\mathsf{unlock}(\mathsf{lock}(x, \mathsf{publicLock}(k)), k) \to x$$

where

1. $\mathsf{publicLock}(k)$ models the lock obtained from the organisation and unlocked by the key k;
2. $\mathsf{lock}(x, l)$ models an object x sealed in a chest locked with l;
3. $\mathsf{unlock}(y, k)$ models an attempt to unlock a chest y with a key k.

The absence of other rules to define the behaviour of locks and chests models an idealising assumption than nothing else can be done to a lock except opening it with the right key (in particular it cannot be forced open).

These kinds of models can be traced back to 1981 with the seminal work of Dolev and Yao [DY81]. A large amount of theoretical follow-up work focused on understanding the precise limits of decidability and the computational complexity of particular protocol classes [DEK82, DLMS99, RT03, DLM04, CC05, KKNS14]. Arguably by building on this better theoretical understanding of the problem, some first automated prototypes for security analysis in this context arose, leading after more than 30 years of active research in the field to efficient and mature tools such as ProVerif [Bla16] and Tamarin [SMCB13] to only cite the most prominent ones. These tools are able to automatically verify full fledged models of widely deployed protocols and standards, such as the TLS protocol for secure http [BBK17, CHH+17], the Signal messaging protocol [KBB17, CGCG+18], authentication protocols of the upcoming 5G standard [BDH+18b], or deployed multi-factor authentication protocols [JK18].

Reachability and indistinguishability The abovementioned results extensively cover verification for the class of *reachability properties*. Such properties formalise properties of the form "*there exist no executions of the protocol potentially involving the attacker leading to a situation where security is broken*". The two properties of the riddle can be formalised as such:

1. *Theft resistance* (or *confidentiality* of the chest's content): there exist no executions of the protocol permitting the delivery guy to eat the cake.
2. *Resilience to impersonation* (or *authentication*): there exist no executions of the protocol where Bob accepts a delivery that is not from Isabelle.

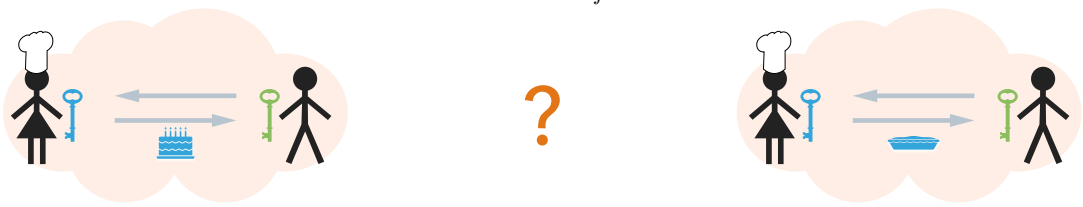
Reachability properties are indeed sufficient to model authentication properties and various flavors of confidentiality, even in complex scenarios with different kinds of compromise [BC14]. Another classical class of properties are *indistinguishability properties* and express that the adversary cannot distinguish between two hypothetical situations. Consider for example:

A client orders a cake from Isabelle and eats it after receiving it.

Situation 1: normal client *Situation 2: immune client*

Upon eating a poisoned cake a normal client gets sick; if the cake is not poisoned nothing happens. On the contrary an immune client never gets sick whatever he eats. If the protocol can allow to observe a difference between these two types of clients, this means that there exists a sequence of actions of the delivery guy that can make one of them eat a poisoned cake. Rephrasing, the fact that no actions of the adversary can distinguish between these two situations is another way of formalising the resilience to impersonation of the cake delivery protocol. It is actually a general fact: reachability properties can be encoded as indistinguishability statements as above. However, indistinguishability can also be used to formalise stronger security notions; consider for example the two hypothetical situations:

A client orders a cake from Isabelle.



Situation 1: chocolate cake *Situation 2: apple pie*

The indistinguishability of these two situations expresses a property that subsumes theft resistance: if the delivery guy has a way to eat the cake then he can of course tell whether it is a chocolate cake or an apple pie[3], but the ability to distinguish them also captures more subtle breaches (for example, the delivery guy being able to smell or see a portion of the cake). More generally, indistinguishability statements allow to express stronger security notions that aim at excluding any form of partial information leakage on sensitive data.

▶ *Remark: verification of V4*
When we used DeepSec to analyse the security of V4, we verified a single indistinguishability property embedding the above two. The file describing it in DeepSec's syntax can be found at [Rak20]. We refer to Chapter 1 in the body of the thesis for more details about DeepSec's framework and to Chapter 2 for more insight on our modelling approach. For tutorials and downloading the tool, see https://deepsec-prover.github.io/. ◀

With the same approach, we could also formalise *anonymity* using the two situations:

A client orders a cake from Isabelle.

Situation 1: the client is Bob *Situation 2: the client is Alice*

The inability to distinguish between the two situations models that the adversary cannot infer what the identity of the client is. This arguably does not make much sense in the context of the riddle where the delivery guy gives the cake to the client in person, but still gives some insight on how to model anonymity in other protocols.

[3] up to potential taste disorders, which would make his obsession for Isabelle's cakes rather ununderstandable

Process equivalences Indistinguishability properties are conveniently modelled as observational equivalences in a cryptographic process calculus, such as the *applied pi calculus* [ABF17]—a classical symbolic model of protocols that we will use in this thesis. Similarly as above they can be used to model strong flavors of *secrecy* in terms of non-interference or as a "real-or-random" experiment. Equivalences are also the tool of choice to model many other privacy-preserving properties. To cite a few, such properties include:

1. *anonymity* [AF04]: is the adversary able to obtain some information about the identities of the participants of the protocol?
2. *unlinkability* [ACRR10, FHMS19]: is the adversary able to track multiple sessions of a user? (typically to track users of e-passport)
3. *vote privacy* [DKR09]: in an electronic election, is the adversary able to learn more information on a participant vote than what is already revealed by the result of the election?

Equivalence properties, instances of so-called *hyperproperties*, are inherently more complex than reachability properties. Both the theoretical understanding and tool support are thus more recent and more brittle. This state of affairs triggered a large amount of recent works—including this thesis—to improve them.

Modelling restrictions Most security properties are however undecidable in general and some restrictions are therefore needed when aiming at decidability results (or in terms of implementation: at fully automated tools with guaranteed termination). Most of the results of this thesis are provided in the following context:

1. *bounded processes*: the number of protocol participants (or alternatively, the number of parallel protocol sessions) is fixed, similarly to our security analysis of V4 in the riddle;
2. we only consider cryptographic primitives that can be represented by *constructor-destructor subterm convergent rewriting systems* (to be defined in the body of the thesis).

The restriction to bounded processes has been introduced in [RT03] when proving reachability properties to be decidable and NP complete if the number of protocol sessions is a parameter of the problem. This gives an interesting insight on the complexity of the problem despite its undecidability in the unbounded case, while remaining non-trivial due to the presence of the active attacker. This restriction to bounded processes typically allows us to obtain decidability and complexity results for equivalence properties in this thesis.

Computational models Although we do not provide new results for them, we can also mention *computational models* for comparison. Also first considered in the 80's (see for example Goldwasser and Micali [GM84]) these models propose a more precise approach of cryptography, taking *probabilities* into account. For example instead of a boolean "secure/not secure" result as we provide in the symbolic analysis of the cake delivery protocol, a computational analysis attempts to bound the probability of the attacker breaching security (which takes into account the probability to guess the client password, to break the locks...).

The resulting analysis is therefore more complete in that it takes more details into account, but is also much harder to automate. Note however that in some contexts we have *computational soundness*, namely symbolic proofs imply computational proofs [CKW11]. Most of the formalisations of security we propose in this thesis are actually inspired by computational definitions, cast in the symbolic model by abstracting probabilities and idealising cryptography.

3 Outline of the contributions

3.1 Part I: Symbolic models of privacy in cryptographic protocols

In the Part I of this thesis we introduce the model of security protocols we use (Chapter 1) and demonstrate through some examples how to use it in practice (Chapter 2). We also provide some reminders of complexity theory. As previously explained our approach can be summed up by two main characteristics:

1. we study *symbolic models* focusing on the logical structure of protocols;
2. we study *equivalence properties* that are the tool of choice to model privacy in this model.

More technically we model processes in a variant of the *applied pi calculus* [ABF17], a symbolic model of concurrent processes inspired by the pi calculus with an active attacker and support for cryptographic primitives. We study several notions of process equivalences for modelling security, namely *trace equivalence* and *labelled bisimilarity*.

In Chapter 2 we then propose some models of classical cryptographic protocols within our model, as well as formalisations of various privacy-type properties using equivalence properties. This chapter has several goals:

1. Providing some examples that illustrate how the applied pi calculus can be used in practice and will serve as benchmarks for our verification techniques in the next chapters.
2. Illustrating the fact that writing models for a bounded number of sessions is sometimes more subtle than doing it for reachability properties.
3. Debating the security definitions used in the literature in the context of anonymity, unlinkability and vote privacy, and proposing some models of them in the applied pi calculus.

3.2 Part II: DEciding Equivalence Properties in SECurity protocols

In Part II we then focus on the DeepSec prover, the protocol verifier we present in this thesis. More precisely, we present:

1. in Chapters 3 and 4, a theoretical decision procedure for trace equivalence and labelled bisimilarity of bounded processes (provided they only use cryptographic primitives that can be represented by a constructor-destructor subterm convergent rewriting system);
2. an implementation of this decision procedure in the case of trace equivalence and an evaluation of its performances[4] (in Chapter 3 as well);
3. in Chapter 5, a new proof technique for trace equivalence that we call *equivalence by session*. Its goal is to exploit the process symmetries that often arise during practical verification, allowing us to increase the efficiency of DeepSec by orders of magnitude.

3.3 Part III: A guide to the complexity of equivalences

Finally in Part III we also contribute to the theoretical understanding of the problem by proving decidability and complexity results for equivalence properties and by integrating them in an exhaustive survey similar results of the literature. More precisely:

1. in Chapter 6 we prove the termination of the algorithm of DeepSec and show that the problem it solves is coNEXP complete;

[4]Note: Vincent Cheval is the main implementor of this decision procedure

2 in Chapter 7 we picture the current state of the art by surveying decidability and complexity results for equivalence properties (of bounded or unbounded processes). This includes some new results we prove or strengthen, and an effort to present all results in a unified model—which allowed us to identify some subtle variations in the statements of the decision problems across the literature.

3.4 Publications and other contributions

The following conference papers have been published during the time of this thesis:

[CKR18a] Vincent Cheval, Steve Kremer, Itsaka Rakotonirina. DEEPSEC: Deciding equivalence properties in security protocols, theory and practice. In *IEEE Symposium on Security and Privacy (S&P)*, 2018.

[CKR18b] Vincent Cheval, Steve Kremer, Itsaka Rakotonirina. The DEEPSEC prover. In *International Conference on Computer Aided Verification (CAV)*, 2018.

[CKR19] Vincent Cheval, Steve Kremer, Itsaka Rakotonirina. Exploiting symmetries when proving equivalence properties for security protocols. In *ACM Conference on Computer and Communications Security (CCS)*, 2019.

[RK19] Itsaka Rakotonirina, Boris Köpf. On aggregation of information in timing attacks. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2019.

[CKR20] Vincent Cheval, Steve Kremer, Itsaka Rakotonirina. The hitchhiker's guide to decidability and complexity of equivalence properties in security protocols. In *Logic, Language, and Security. Essays Dedicated to Andre Scedrov on the Occasion of His 65th Birthday (ScedrovFest65)*, 2020.

[DLFP+21] Antoine Delignat-Lavaud, Cedric Fournet, Bryan Parno, Jonathan Protzenko, Tahina Ramananandro, Jay Bosamiya, Joseph Lallemand, Itsaka Rakotonirina, Yi Zhou. A Security Model and Fully Verified Implementation for the IETF QUIC Record Layer. In *IEEE Symposium on Security and Privacy (S&P)*, 2021.

Let us comment more precisely on their content.

1 [CKR18a] presents our decidability and complexity results at the core of the DeepSec prover, therefore corresponding to Chapters 3,4,6. Note however that the performances of DeepSec and the presentation of the theory have significantly improved since this publication.

2 [CKR18b] is a tool paper presenting DeepSec, therefore with an emphasis on the engineering effort deployed. As above, both the performances and the user interface have significantly evolved since then.

3 [CKR19] describes the optimisation technique presented in Chapter 5, although the version presented in this manuscript is done in a slightly more general framework.

4 [CKR20] finally contains the decidability and complexity results of Chapter 7 without significant changes.

The additional two contributions [RK19, DLFP+21] correspond to side work on other topics with different contributors, focusing more on the security implications of how cryptographic primitives and protocols are implemented. They do not correspond to chapters of this thesis.

5 In [RK19] we study to which extent particular implementations of encryption primitives such as RSA or AES are subject to so-called *timing attacks*. This kind of vulnerabilities, known since early works in 1996-1998 on modular exponentiation [Koc96, DKL+98], relies

on information leakage due to the execution time of programs. We provide in this paper an attacker model more suited for explaining and understanding practical timing attacks, studying more particularly attack strategies by divide-and-conquer. We formalise sufficient condition for them to be correct under the form of independence statements and study their complexity.

6 I also intermittently participated to a long-time effort for a security analysis of the QUIC protocol in [DLFP+21], carried among others by Microsoft Research. QUIC is a protocol initially proposed by Google in 2012 and that is currently under standardisation by the IETF. Although the initial motivation of the IETF was to design the protocol so that it inherits security properties from TLS 1.3 (which required years of work to prove), many TLS abstractions were broken in the process, resulting in a new protocol with new cryptographic constructions but few security insights on them. I contributed to prove in the F* proof assistant some functional properties of the QUIC transport and study privacy issues due to the weak authentication of some transport parameters.

Part I

Symbolic models of privacy in cryptographic protocols

Introduction

An analysis of the protocol's logic

In this first part of the thesis (Chapters 1 and 2) we introduce the model we use all along this thesis and present detailed examples that will serve as illustrations of the formalism as well as opportunities to discuss practical security modelling questions.

Concurrent processes As security protocols are distributed by essence, *concurrent process calculi* such as the pi calculus [MPW92] are a natural fit for modelling them. However the calculus is quite minimal and the resulting models would therefore have to rely on complex, often unnatural or at least error-prone, encodings. This was the reason behind the design of *more specialised extensions* of the pi calculus including support for cryptography and an explicit model of an active attacker controlling the communication network. We can cite for example the spi calculus [AG99] which provides support for a fixed set of common primitives such as encryption and pairs; but we are here more interested in its later generalisation, the *applied pi calculus* [ABF17], that can handle in a more compact and uniform way arbitrary primitives using *equational theories*.

This kind of model is *symbolic* (by opposition to *computational models*) in that it abstracts probabilities to focus on the logical structure of protocols. Such models find their origin in the seminal work of Dolev and Yao in the 80's [DY81]. A nonce, randomness or encryption key will be modelled by a so-called *name* that is assumed uncomputable by the adversary at the beginning of the protocol execution (which may change at some point during the execution since the adversary spies on the communication network and, therefore, may learn the value of this data from the flow of messages). The equational theories also abstract the behaviour of cryptographic primitives by a set of equations modelling their functional properties—and nothing more, which is an idealising assumption. This simplifying abstraction typically facilitates the automation of security analyses as we will see in Part II.

Equivalence properties Regarding the modelling of security properties, we are here interested in *equivalence properties*, typically used to model indistinguishability statements between two hypothetical situations. This kind of approach is well-suited for modelling various flavours of privacy-type properties such as (strong notions of) secrecy, anonymity, unlinkability, or even ballot privacy in the context of electronic voting. More precisely we will study a running example throughout Chapter 1 to illustrate the various notions we introduced in a simple context, and study more precise modelling concerns in Chapter 2 (including discussions on how to

capture an accurate notion of privacy without unconsciously restricting the potential attack scenarios). We study:

1. the *private authentication protocol* [AF04] and study what kind of violations of *authentication*, *anonymity*, and *secrecy* can be uncovered using our approach;
2. the *Basic Access Control* (BAC) protocol [For04] implemented in European e-passports with a focus on *unlinkability*;
3. a mixnet variant of the *Helios* e-voting protocol [Adi08] with a focus on *ballot privacy*.

In the three cases we evaluate in particular the importance of considering *compromised sessions*—namely sessions of the protocol executed by dishonest participants—and discuss the resulting security implications.

Chapter 1:

Model: the applied pi-calculus

Summary.

This chapter details the model of cryptographic protocols we study in this thesis as well as the class of security properties we aim at verifying. Our formalism is strongly inspired by the *applied pi-calculus* [ABF17] focusing on the *protocol's logic* with idealised cryptography. We consider security properties that can be expressed using *behavioural equivalences*, typically different flavours of privacy.

NB. *This chapter also includes some reminders of complexity theory (Section 4).*

1 Cryptographic primitives and messages

> ***Section summary***
>
> We first describe the model of cryptographic primitives and protocol messages. This is done using a *term algebra*, cryptographic properties being idealised a set of *rewriting rules*.

1.1 Term algebra

Function symbols Models such as ours are said to be *symbolic* in that they rely on a abstraction of the messages sent during a protocol execution. Rather than representing them as arbitrary bitstrings, this abstraction aims at capturing their structure and their functional properties (and only these). Concretely, a cryptographic primitive is represented by a *function symbol* f that has an *arity* n (i.e., the number of arguments it takes). The symbol is sometimes referred as f/n. A finite set of such symbols

$$\mathcal{F} = \{\mathsf{f}/n,\ \mathsf{g}/m,\ \mathsf{h}/p, \ldots\} \qquad \text{with a partition } \mathcal{F} = \mathcal{F}_{\mathsf{c}} \uplus \mathcal{F}_{\mathsf{d}}$$

is called a *signature*. The elements of $\mathcal{F}_{\mathsf{c}}$ are called *constructors* and those of $\mathcal{F}_{\mathsf{d}}$ *destructors*: intuitively the latter will model functions that fail when their argument are not of a certain form (for example a decryption function that can only process valid ciphertexts). Symbols of arity 0 are specifically called *constants* and model public values such as agent identities or protocol tags. Since our goal is to study security in presence of an adversary with unbounded computational power, they may generate arbitrarily many constants to compute messages which is why we assume an infinite set of constants $\mathcal{F}_0$ and we consider that $\mathcal{F}$ only contains

the symbols of positive arity for simplicity[1]. On the other hand private values such as fresh nonces or long-term keys that are assumed unknown to third parties upon generation, are modelled by an infinite set of *names* $\mathcal{N}$. Finally we may also use *variables* from an infinite set $\mathcal{X}$ that represent bound values. A message sent during a protocol is then represented by:

Definition 1.1 (term)

A *term* is either a name, a variable, or a symbol of arity n applied to n other terms. The set of terms built from the values and functions of $A \subseteq \mathcal{F} \cup \mathcal{F}_0 \cup \mathcal{X} \cup \mathcal{N}$ is written $\mathcal{T}(A)$. Besides if $t \in \mathcal{T}(A)$, we say that t is a *constructor term* if $A \cap \mathcal{F}_{\mathsf{d}} = \varnothing$, a *ground term* if $A \cap \mathcal{X} = \varnothing$, and a *public term* if $A \cap \mathcal{N} = \varnothing$.

▶ *Example 1.1*

A signature $\mathcal{F}$ modelling symmetric encryption and pairs is usually written as follows

$$\mathcal{F} = \mathcal{F}_{\mathsf{c}} \uplus \mathcal{F}_{\mathsf{d}} \qquad \mathcal{F}_{\mathsf{c}} = \{\mathsf{senc}/2, \langle\ ,\ \rangle/2\} \qquad \mathcal{F}_{\mathsf{d}} = \{\mathsf{sdec}/2, \mathsf{fst}/1, \mathsf{snd}/1\}$$

The symbols senc and sdec respectively model encryption and decryption, $\langle u, v\rangle$ models a pair of terms u and v that will intuitively be retrievable using the projection functions fst and snd. We also often use the common condensed notation $\langle u_1, \ldots, u_n\rangle$ to refer to tuples of n nested pairs $\langle u_1, \langle u_2, \langle \ldots, u_n\rangle\rangle\rangle$. The encryption of a plaintext m with a key k would then be modelled by the term $\mathsf{senc}(m, k)$. The fact that sdec is in $\mathcal{F}_{\mathsf{d}}$ rather than $\mathcal{F}_{\mathsf{c}}$ intuitively models that the function fails when its argument is not of the correct form, that is, when attempting to decrypt a message that is not an encryption, or an encryption with the wrong key. Putting it in $\mathcal{F}_{\mathsf{c}}$ instead will model that the decryption always returns a result. However in both cases this only models a deterministic encryption scheme: to include a randomness $r \in \mathcal{N}$ one may use a pair, i.e., consider the term $\mathsf{senc}(\langle m, r\rangle, k)$. An alternative modelling of randomness consists of using dedicated symbols $\mathsf{rsenc}/3, \mathsf{rsdec}/2$ that include it as additional argument, leading to the ciphertext $c = \mathsf{senc}(m, r, k)$ decrypted as $\mathsf{rsdec}(c, k)$. ◀

The set of variables (resp. names, subterms, strict subterms) of a term t is written $vars(t)$ (resp. $names(t)$, $subterms(t)$, $subterms_{\neq}(t)$); all these notations are extended to sets of terms in the natural way. When we want to refer to the specific subterm of t at position p (where p is a sequence of positive integers), we write $t_{|p}$. The function symbol (or the variable or the name) at the root of a term t is written $root(t)$.

Substitutions Given a variable x, we refer by $t\{x \mapsto u\}$ to the term obtained by replacing all occurrences of x in t by the term u. More generally we call a *substitution* a mapping $\sigma = \{x_1 \mapsto u_1, \ldots, x_n \mapsto u_n\}$, where $x_1, \ldots, x_n$ are pairwise distinct variables and form the *domain* of σ written $dom(\sigma)$, and $u_1, \ldots, u_n$ are terms and form the *image* of σ written $img(\sigma)$. They are homomorphically applied to terms, that is, $t\sigma$ is the term obtained by replacing all occurrences of x_i in t by $u_i = x_i\sigma$. The term $t\sigma$ is called an *instance* of t. Going further with the notations inspired from sets, we also write $\sigma \subseteq \sigma'$ to express that σ' is an extension of σ, and $\sigma \cup \sigma'$ the substitution of domain $dom(\sigma) \cup dom(\sigma')$ that extends both σ and σ' (provided they coincide on $dom(\sigma) \cap dom(\sigma')$). The substitution $\varnothing$, i.e., the identity, is rather referred as $\top$. Finally we write $\sigma\sigma'$ the composition $\sigma' \circ \sigma$, i.e., for all terms t, $t\sigma\sigma' = (t\sigma)\sigma'$.

[1] In other formalisations of the applied pi-calculus, e.g., [CKR18a], there are finitely many constants but a dedicated infinite set of *public names*. Here both are merged in $\mathcal{F}_0$ as in, e.g., [CCD15b].

1.2 Rewriting

Function symbols are only syntactic and it remains to model their properties, e.g., encryption-decryption cancellation (i.e., the fact that $\mathsf{sdec}(\mathsf{senc}(m,k),k) = m$ in Example 1.1). In this thesis we do this using a *rewriting system* $\mathcal{R}$ which is a finite binary relation on terms, i.e., a finite set of pairs of terms (ℓ, r) usually written $\ell \to r$ and called *rewrite rules*. We always assume that ℓ and r do not contain any names. A rewriting system $\mathcal{R}$ induces a relation on terms $\to$ defined as its closure by subsitution and context, i.e., by the inference rules:

$$\frac{(\ell \to r) \in \mathcal{R}}{\ell \to r} \qquad \frac{u \to v \qquad \sigma \text{ substitution}}{u\sigma \to v\sigma} \qquad \frac{u_i \to u_i'}{\mathsf{f}(u_1,\dots,u_n) \to \mathsf{f}(u_1,\dots,u_{i-1},u_i',u_{i+1},\dots,u_n)}$$

Definition 1.2 (theory)

In this thesis we call a *theory* a pair $(\mathcal{F},\mathcal{R})$ with $\mathcal{F}$ a signature and $\mathcal{R}$ a rewriting system. All definitions involving terms at any point are to be understood w.r.t. a theory that is sometimes left implicit for the sake of succinctness.

▶ *Example 1.2*

The following rewrite rules define the theory of pairs and encryption:

$$\mathsf{fst}(\langle x,y\rangle) \to x \qquad \mathsf{snd}(\langle x,y\rangle) \to y \qquad \mathsf{sdec}(\mathsf{senc}(x,y),y) \to x$$

Typically one can decrypt (apply sdec) a ciphertext $\mathsf{senc}(x,y)$ with the corresponding key y to recover the plaintext x. This behaviour is idealised by the absence of other rules for senc and sdec, modelling an assumption that no information can be extracted from a ciphertext except by knowing the decryption key. The analogue for randomised symmetric encryption would be $\mathsf{rsdec}(\mathsf{rsenc}(x,y,z),z) \to x$. Let us also give a toolbox of other common theories:

1 *asymmetric encryption*, which is the analogue with public-key mechanisms:

non-rand.	$\mathcal{F}_\mathsf{c} = \{\mathsf{pk}/1, \mathsf{aenc}/2\}$	$\mathcal{F}_\mathsf{d} = \{\mathsf{adec}/2\}$	$\mathsf{adec}(\mathsf{aenc}(m,\mathsf{pk}(k)),k) \to m$
rand.	$\mathcal{F}_\mathsf{c} = \{\mathsf{pk}/1, \mathsf{raenc}/3\}$	$\mathcal{F}_\mathsf{d} = \{\mathsf{radec}/2\}$	$\mathsf{radec}(\mathsf{raenc}(m,r,\mathsf{pk}(k)),k) \to m$

2 *digital signature*, with a verification mechanism that recovers the signed message:

$$\mathcal{F}_\mathsf{c} = \{\mathsf{pk}/1, \mathsf{sign}/3\} \quad \mathcal{F}_\mathsf{d} = \{\mathsf{verify}/2\} \qquad \mathsf{verify}(\mathsf{sign}(m,r,k),\mathsf{pk}(k)) \to m$$

3 *cryptographic hash*, using a function symbol of positive arity, e.g., $\mathcal{F} = \mathcal{F}_\mathsf{c} = \{\mathsf{h}/1\}$, and no rewrite rules involving this symbol (in which case we say that h is *free*). The absence of rewrite rules models the expected properties of a cryptographic hash function, typically one-wayness or collision resistance.

4 *zero-knowledge proofs*, to prove that a given statement is true without revealing any related information: a typical example in cryptography being to prove to a party that one knows a value x without revealing the value itself. The modelling of such theories in the applied pi-calculus is for instance studied in [BMU08] but we present here a more specific example that will be used in some of the protocol models presented in this thesis. Say that a recipient only accepts a message if the sender provides a proof that they have encrypted it themselves.

For that the sender can provide, together with the ciphertext c, a zero-knowledge proof that they know the plaintext m and the randomness r used during the encryption. This proof is modelled by the term $p = \mathsf{zkp}(c, m, r)$ and the recipient can check it with the test $\mathsf{checkzkp}(p, c) = \mathsf{ok}$ in the theory defined by $\mathcal{F}_\mathsf{c} = \{\mathsf{zkp}/3\}$, $\mathcal{F}_\mathsf{d} = \{\mathsf{checkzkp}/2\}$ and, for $\mathsf{ok} \in \mathcal{F}_0$, the rewrite rule

$$\mathsf{checkzkp}(\mathsf{zkp}(\mathsf{raenc}(m, r, \mathsf{pk}(k)), m, r), \mathsf{raenc}(m, r, \mathsf{pk}(k))) \to \mathsf{ok}$$ ◀

If $t \to^* u$ where $\to^*$ is the reflexive transitive closure of $\to$ and u is not reducible by $\to$, we say that u is a *normal form* of t. We only consider rewriting systems $\mathcal{R}$ that are *convergent*, i.e., 1 there exist no infinite rewriting sequences and 2 whenever $t \to^* u_1$ and $t \to^* u_2$, there needs exist a term v such that $u_1 \to^* v$ and $u_2 \to^* v$. In particular all terms t have a unique normal form written $t{\downarrow}$. This induces a notion of well-formedness of a term, stating that all destructor applications succeed yielding a valid message:

Definition 1.3 (message)

Let $(\mathcal{F}, \mathcal{R})$ be a convergent theory and t be a term. We say that t is a *message*, written $\mathsf{msg}(t)$, when for all $u \in \mathit{subterms}(t)$, $u{\downarrow}$ is a constructor term.

This captures the previously-mentioned intuition that a destructor is a function symbol that may fail (where failing means not being reduced to a constructor term, i.e., a valid protocol message). The predicate msg is extended to sequences of terms $\mathsf{msg}(t_1, \ldots, t_n)$ in the natural way. This predicate is used to restrict to protocols that only send and accept well-formed messages; in particular, putting a function symbol f in $\mathcal{F}_\mathsf{c}$ rather than in $\mathcal{F}_\mathsf{d}$ models an assumption that f always returns a valid result.

▶ *Example 1.3*

We refer to the theories introduced in Example 1.2. If $m \in \mathcal{F}_0$ and $k \in \mathcal{N}$, $\mathsf{sdec}(\mathsf{senc}(m, k), k)$ is a message of normal form m; on the contrary the terms $\mathsf{sdec}(m, k)$ and $\mathsf{sdec}(\mathsf{senc}(m, k), k')$, $k \neq k'$, are both in normal form and contain a destructor and are therefore not messages. We also mention a less intuitive example $\mathsf{fst}(\langle m, \mathsf{sdec}(m, k)\rangle)$ which is *not* a message although it reduces to m which is a one. ◀

Definition 1.4 (equality modulo theory)

Two terms u and v are *equal modulo* a convergent theory E, written $u =_E v$, if $\mathsf{msg}(u, v)$ and $u{\downarrow} = v{\downarrow}$ w.r.t. E. Thus $=_E$ is an equivalence relation when restricted to messages and $u =_E u$ *iff* $\mathsf{msg}(u)$.

1.3 Classical classes of theories

In addition of convergence two classes of theories are particularly important in this thesis.

Definition 1.5 (subterm convergent)

A theory $(\mathcal{F}, \mathcal{R})$ is *subterm convergent* when $\mathcal{R}$ is convergent and for all rules $(\ell \to r) \in \mathcal{R}$, r is either a strict subterm of ℓ or a ground term in normal form (w.r.t. $\mathcal{R}$).

This class of theories has been studied significantly in the context of protocol analysis, see for example [AC06, Bau07, BAF08, CKR18a, CDK12, CBC11]. All theories discussed in Example 1.2 are subterm convergent; typical examples of non-subterm convergent theories are *blind signature* and *trapdoor commitment* (see [CDK12] for details). We also consider:

Definition 1.6 (constructor-destructor)

A theory $(\mathcal{F}, \mathcal{R})$ is *constructor-destructor* if all rewrite rules $(\ell \to r) \in \mathcal{R}$ are of the form $\ell = d(t_1, \ldots, t_n)$ with $d \in \mathcal{F}_{\mathsf{d}}$ and $t_1, \ldots, t_n, r$ constructor terms.

Such theories have been studied for example in [BAF08, CCLD11, CKR18a]. Intuitively a theory is constructor-destructor if the rewrite rules only describe the behaviours of destructor symbols. This gives rewriting sequences convenient properties: for example a term t may be rewritten only at positions where there is a destructor, and the number of destructors appearing in a term strictly decreases at each rewriting step. On the other hand one could also consider their complete opposite, namely theories where no functions can fail:

Definition 1.7 (equational)

In this thesis a theory $(\mathcal{F}, \mathcal{R})$ is said to be *equational* if $\mathcal{F} = \mathcal{F}_{\mathsf{c}}$, that is, $\mathcal{F}_{\mathsf{d}} = \varnothing$.

Equational theories model an assumption that all functions always return a valid message. Considering the equational variant of the theory of pairs for example (that is, with the symbols fst and snd in $\mathcal{F}_{\mathsf{c}}$) the term $\mathsf{fst}(u)$ is valid message even if u is not a pair. When refering to the theories introduced in Example 1.2, we add the explicit mention "constructor-destructor" or "equational" (e.g., "constructor-destructor pairs" or "equational pairs") to distinguish between the variant presented in Example 1.2 and the one where all symbols are put in $\mathcal{F}_{\mathsf{c}}$.

2 Protocols in an adversarial environment

Section summary

We now model protocols using a calculus of *concurrent processes*. They come with an *operational semantics* modelling their executions over a network controlled by an active attacker.

2.1 Processes

In the applied pi-calculus, protocols are modelled by *processes* that can be seen as concurrent programs sending and receiving messages. Unlike the original pi calculus [MPW92] messages are modelled by terms and not merely names, making the modelling process often more intuitive or accurate. Assuming a theory $(\mathcal{F}, \mathcal{R})$, their syntax is defined by the following grammar:

$$
\begin{array}{rll}
P, Q ::= & 0 & \text{(null process)} \\
& \mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q & \text{(conditional)} \\
& u(x).P & \text{(input)} \\
& \overline{u}\langle v \rangle.P & \text{(output)} \\
& P \mid Q & \text{(parallel)}
\end{array}
$$

where u, v are terms and x a variable. Intuitively 0 models a terminated process, a conditional $\mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q$ executes either P or Q depending on whether u and v are equal modulo theory, and $P \mid Q$ models two processes executed concurrently. The constructs $u(x).P$ and $\overline{u}\langle v\rangle.P$ model, respectively, inputs and outputs on a communication channel u. When the channel u is known to the attacker, e.g., when $u \in \mathcal{F}_0$, executing an output on u adds it to the adversary's knowledge and inputs on u are provided by the adversary possibly forwarding a previously stored message, or computing a new message from previous outputs. Otherwise the communication is performed silently without adversarial interferences (*internal communication*). To model an unbounded number of protocol sessions we add the constructs

$$\begin{array}{lll} P, Q ::= & \mathsf{new}\ k.P & \text{(new name)} \\ & !\,P & \text{(replication)} \end{array}$$

The replication $!\,P$ models an unbounded number of parallel copies of P, and $\mathsf{new}\ k.P$ creates a fresh name k unknown to the attacker; in particular $!\,\mathsf{new}\ k.P$ models an unbounded number of sessions, each with a different fresh key. The fragment of the calculus without replication is referred as *finite* or *bounded*. Another notable subclass is the original pi-calculus [MPW92], referred as the *pure* fragment, that can be retrieved with the empty theory ($\mathcal{F}$ and $\mathcal{R}$ empty).

▶ *Remark: bound and free variables*
The sets $vars(P)$ and $names(P)$ only refer to the *free* variables and names of a process, respectively; that is, variables that are not bound by an input and names that are not bound by a new operator. For example $vars(c(x).0) = \varnothing$ and $names(\mathsf{new}\ k.0) = \varnothing$. ◀

▶ *Example 1.4*
We define a process modelling the protocol for private authentication described in [AF04] as a running example through the paper. Writing sk_X, pk_X the secret and public keys of an agent X, its control flow can be described in informal Alice-Bob notation as follows:

$$\begin{array}{lll} X \to B : & \mathsf{raenc}(\langle N_X, pk_X\rangle, r_1, pk_B) & \\ B \to X : & \mathsf{raenc}(\langle N_X, N_B, pk_B\rangle, r_2, pk_A) & \text{if } X = A \\ & \mathsf{raenc}(N_B, r_2, pk_B) & \text{if the decryption fails or } X \neq A \end{array}$$

where N_X, N_B, r_1, r_2 are freshly generated nonces. Here the agent B accepts authentication requests from the agent A but not from other parties. The security goals stated in [AF04] are

1. *Authentication and secrecy:* After a successful instance of the protocol between A and B, N_A and N_B are shared secrets (i.e., A and B know their values but not the attacker).
2. *Anonymity:* Upon observing and interfering with a session of the protocol, the attacker cannot tell whether it was run by A and B or by other agents.
3. *Private authentication:* The attacker cannot infer the set of identities accepted by B.

The last two items justify the decoy message $\mathsf{raenc}(N_B, pk_B)$ that B sends upon failure: from an outside observer there should not be an observable difference between the situations where B accepts the connection or not. The roles of X and B can be specified as follows in the applied pi calculus; each process takes as an argument its secrey key s, the public key p of the

agent it aims at communicating with, its fresh session nonce n and we let $t = \mathsf{radec}(x, s)$:

$$\begin{aligned}\mathsf{Send}(s,p,n) = {} & \mathsf{new}\ r.\\ & \overline{c}\langle \mathsf{raenc}(\langle n, \mathsf{pk}(s)\rangle, r, p)\rangle.\\ & c(x)\end{aligned} \qquad \begin{aligned}\mathsf{Rcpt}(s,p,n) = {} & d(x).\,\mathsf{new}\ r.\\ & \mathsf{if}\ \mathsf{snd}(t) = p\ \mathsf{then}\\ & \quad \overline{d}\langle \mathsf{raenc}(\langle \mathsf{fst}(t), n, \mathsf{pk}(s)\rangle, r, p)\rangle\\ & \mathsf{else}\ \overline{d}\langle \mathsf{raenc}(n, r, \mathsf{pk}(s))\rangle\end{aligned}$$

where $c, d \in \mathcal{F}_0$. The fact that the two processes operate on different channels c, d models an assumption that the attacker can distinguish between the messages of one or the other. ◀

2.2 Semantics in an adversarial environment

Attacker's knowledge The semantics of processes includes a model of the attacker's knowledge aggregated during the protocol's execution. We model the attacker's observations when spying on the communication network by a *frame*, which is a substitution of the form

$$\Phi = \{\mathsf{ax}_1 \mapsto t_1, \ldots, \mathsf{ax}_n \mapsto t_n\} \qquad t_i \text{ messages in normal form}$$

where t_i are the outputs performed during the protocol and $\mathcal{AX} = \{\mathsf{ax}_i\}_{i\in\mathbb{N}}$ is a set of special variables called *axioms* that serve as handles to the adversary for building new terms. The terms t_i enable deductions as they aggregate; for example after observing a ciphertext and the decryption key the attacker can also obtain the plaintext by decrypting. Formally:

Definition 1.8 (recipe, deduction)
A *recipe* is a term $\xi \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_0 \cup \mathcal{AX})$. We say that a recipe ξ *deduces* a message u from a frame Φ when $\xi\Phi =_E u$. The set of axioms appearing in ξ is written $\mathit{axioms}(\xi)$.

A recipe describes how the adversary can compute a message from the observations aggregated in Φ. The fact that a recipe cannot contain names models the fact that they are assumed unknown to the attacker. Naturally they are not anymore if they get revealed themselves; for example in $\Phi = \{\mathsf{ax}_1 \mapsto \mathsf{senc}(t, k), \mathsf{ax}_2 \mapsto k\}$ even if deducing the term t requires to decrypt ax_1 with the key k (which is not allowed to occur directly in the recipe), this is possible by using $\xi = \mathsf{sdec}(\mathsf{ax}_1, \mathsf{ax}_2)$.

Operational semantics The behaviour of processes is formalised by an operational semantics operating on *extended processes* $A = (\mathcal{P}, \Phi(A))$, where $\mathcal{P}$ is a multiset of processes modelling the state of the processes executed in parallel, and $\Phi(A)$ is the frame indicating the outputs the attacker has recorded during the execution. It takes the form of a labelled transition relation $\xrightarrow{\alpha}$ whose label α is called an *action*, which is either:

1. a public *input action* $\xi_c(\xi_t)$ modelling the reception of a message forged by the adversary, where ξ_c (resp. ξ_t) is a recipe indicating how the adversary has computed the input's channel (resp. the term to be input);
2. a public *output action* $\overline{\xi_c}\langle \mathsf{ax}_i\rangle$ modelling a message sent on the network (and therefore to the adversary), where ξ_c is a recipe for the output's channel, and the underlying output term is added to the frame under axiom ax_i;
3. an *unobservable action* τ which represents an internal action, such as the evaluation of a conditional or a communication on a private channel.

$$(\{\!\{\mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q\}\!\} \cup \mathcal{P}, \Phi) \xrightarrow{\tau} (\{\!\{R\}\!\} \cup \mathcal{P}, \Phi) \quad \begin{array}{l} R = P \text{ if } u =_E v \\ R = Q \text{ otherwise} \end{array} \quad (\textsc{Test})$$

$$(\{\!\{u(x).P\}\!\} \cup \mathcal{P}, \Phi) \xrightarrow{\xi_c(\xi_t)} (\{\!\{P\{x \mapsto \xi_t\Phi\downarrow\}\}\!\} \cup \mathcal{P}, \Phi) \quad \text{if } \mathsf{msg}(\xi_t\Phi) \text{ and } \xi_c\Phi =_E u \quad (\textsc{In})$$

$$(\{\!\{\overline{u}\langle v\rangle.P\}\!\} \cup \mathcal{P}, \Phi) \xrightarrow{\overline{\xi_c}\langle \mathsf{ax}\rangle} (\{\!\{P\}\!\} \cup \mathcal{P}, \Phi \cup \{\mathsf{ax} \mapsto v\downarrow\}) \quad \begin{array}{l} \text{if } \mathsf{msg}(v) \text{ and } \xi_c\Phi =_E u \\ \text{and } \mathsf{ax} \in \mathcal{AX} \setminus dom(\Phi) \end{array} \quad (\textsc{Out})$$

$$(\{\!\{\overline{u}\langle v\rangle.P, w(x).Q\}\!\} \cup \mathcal{P}, \Phi) \xrightarrow{\tau} (\{\!\{P, Q\{x \mapsto v\}\}\!\} \cup \mathcal{P}, \Phi) \quad \begin{array}{l} \text{if } \mathsf{msg}(v),\ u =_E w, \text{ and} \\ u \text{ not deducible from } \Phi \end{array} \quad (\textsc{Comm})$$

$$(\{\!\{\mathsf{new}\ k.P\}\!\} \cup \mathcal{P}, \Phi) \xrightarrow{\tau} (\{\!\{P\{k \mapsto k'\}\}\!\} \cup \mathcal{P}, \Phi) \quad \text{if } k' \in \mathcal{N} \setminus names(P, \mathcal{P}, \Phi) \quad (\textsc{New})$$

$$(\{\!\{P \mid Q\}\!\} \cup \mathcal{P}, \Phi) \xrightarrow{\tau} (\{\!\{P, Q\}\!\} \cup \mathcal{P}, \Phi) \quad (\textsc{Par})$$

$$(\{\!\{!\,P\}\!\} \cup \mathcal{P}, \Phi) \xrightarrow{\tau} (\{\!\{!\,P, P\}\!\} \cup \mathcal{P}, \Phi) \quad (\textsc{Repl})$$

Figure 1.1 Operational semantics of the applied pi-calculus

The relation is formalised in Figure 1.1. The rule (Test) compares u and v modulo theory which, we recall, requires that $\mathsf{msg}(u, v)$, i.e., all destructors in u and v should succeed for the test to be positive. In particular a test $\mathsf{if}\ u = u\ \mathsf{then}\ P\ \mathsf{else}\ Q$ will execute Q if $\mathsf{msg}(u)$ does not hold. The rules (In) and (Out) tackle public communications in that the channel u they are performed on should be deducible by the attacker (using recipe ξ_c). As expected this kind of communications are routed by the adversary: public outputs are sent to them, increasing the frame, and public inputs are computed by them using previous outputs, i.e., recipe ξ_t. On the contrary (Comm) models a internal communication on a channel u non deducible to the adversary. However, as noted in [BCK20], there exist several variants of this rule:

1. The *classical semantics*, which is the original semantics of the applied pi-calculus [ABF17], where internal communications are possible on any channel u (i.e., the condition that u should not be deducible from Φ is discarded).
2. The *private semantics* which is the one described in Figure 1.1: internal communications are only allowed on channels that are not computable by the adversary. This models an attacker that continuously spies on compromised channels, rather than the classical attacker that simply has the capability to do so.
3. The *eavesdrop semantics* that allows internal communication on any channel like the classical semantics, but the attacker gets knowledge of the underlying message if the channel is compromised. This semantics has been introduced in [BCK20].

These variations may seem unimportant in that they preserves reachability properties. However equivalence properties can be impacted by the choice of the semantics, as discussed in [BCK20] (for example the classical and private semantics induce incomparable notions of equivalence). All results of this thesis (decidability, complexity, case studies) are to be understood w.r.t. the private semantics but most can be adapted to the other two.

Definition 1.9 (trace)

If A_0 be an extended process, a *trace* of A_0 is a sequence of successive reductions steps $t : A_0 \xrightarrow{\alpha_1} A_1 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_n} A_n$. If tr is the word obtained by removing τ actions from the word $\alpha_1 \cdots \alpha_n$, this trace may be referred succinctly as $t : A_0 \overset{\mathsf{tr}}{\Rightarrow} A_n$. Specifically, if ε is the empty word, $t : A_0 \overset{\varepsilon}{\Rightarrow} A_n$ is used to mean that t is a sequence of τ-transitions.

▶ *Example 1.5*

We now illustrate how our running example can be executed in the operational semantics. We let two agents of respective secret keys $sk_A, sk_B \in \mathcal{N}$ and nonces $N_A, N_B \in \mathcal{N}$. An instance of the protocol between A and B is thus modelled, using the notations of Example 1.4, by the process $P = \bar{A} \mid \bar{B}$ where

$$\bar{A} = \mathsf{Send}(sk_A, \mathsf{pk}(sk_B), N_A) \qquad \bar{B} = \mathsf{Rcpt}(sk_B, \mathsf{pk}(sk_A), N_B)$$

We name the three messages sent during the protocol as follows, for two fresh names r_1, r_2:

$$\begin{aligned} m_A &= \mathsf{raenc}(\langle N_A, \mathsf{pk}(sk_A)\rangle, r_1, \mathsf{pk}(\mathsf{sk}_B)) \\ m_B &= \mathsf{raenc}(\langle N_A, N_B, \mathsf{pk}(sk_B)\rangle, r_2, \mathsf{pk}(\mathsf{sk}_A)) \\ m'_B &= \mathsf{raenc}(N_B, r_2, \mathsf{pk}(sk_B)) \end{aligned}$$

We assume that the public keys $\mathsf{pk}(sk_A)$ and $\mathsf{pk}(sk_B)$ are known to the attacker, which can be modelled by an initial frame $\Phi_0 = \{\mathsf{ax}_1 \mapsto \mathsf{pk}(sk_A), \mathsf{ax}_2 \mapsto \mathsf{pk}(sk_B)\}$. An other possibility is to prefix the process P with two outputs of $\mathsf{pk}(sk_A)$ and $\mathsf{pk}(sk_B)$ respectively, producing the frame Φ_0 after two applications of rule (Out). The normal execution of the protocol is:

$$\begin{aligned} (\{\!\{P\}\!\}, \Phi_0) &\xrightarrow{\tau} (\{\!\{\bar{A}, \bar{B}\}\!\}, \Phi_0) \\ &\xrightarrow{\bar{c}\langle \mathsf{ax}_3\rangle} (\{\!\{c(x), \bar{B}\}\!\}, \Phi_1) && \text{with } \Phi_1 = \Phi_0 \cup \{\mathsf{ax}_3 \mapsto m_A\} \\ &\xrightarrow{d(\mathsf{ax}_3)\,\bar{d}\langle \mathsf{ax}_4\rangle} (\{\!\{c(x), 0\}\!\}, \Phi_2) && \text{with } \Phi_2 = \Phi_1 \cup \{\mathsf{ax}_4 \mapsto m_B\} \\ &\xrightarrow{c(\mathsf{ax}_4)} (\{\!\{0, 0\}\!\}, \Phi_2) \end{aligned}$$

In this execution the attacker only forwards messages, that is, each input action uses the last axiom added to the frame as a recipe. However the adversary may actively engage in the protocol, for example to try to infer whether B accepts communications from a third agent C. For that they could generate a fresh nonce $N \in \mathcal{F}_0$, send it to B pretending it is from C and observe how it responds. With the initial frame $\Phi'_0 = \Phi_0 \cup \{\mathsf{ax}_3 \mapsto \mathsf{pk}(sk_C)\}$ and $r \in \mathcal{F}_0$, this scenario corresponds to the trace:

$$\begin{aligned} (\{\!\{P\}\!\}, \Phi'_0) &\xrightarrow{\tau} (\{\!\{\bar{A}, \bar{B}\}\!\}, \Phi'_0) \\ &\xrightarrow{d(\mathsf{raenc}(\langle N, \mathsf{ax}_3\rangle, r, \mathsf{ax}_2))} (\{\!\{\bar{A}, \bar{d}\langle m'_B\rangle\}\!\}, \Phi'_0) \\ &\xrightarrow{\bar{d}\langle \mathsf{ax}_4\rangle} (\{\!\{\bar{A}, 0\}\!\}, \Phi'_0 \cup \{\mathsf{ax}_4 \mapsto m'_B\}) \end{aligned}$$

This does not leak information assuming the attacker cannot distinguish m_B from m'_B. ◀

When defining security against an active attacker we quantify over all traces which means we consider all possible executions in an active adversarial environment. Thus even the bounded fragment yields an infinite transition system if the theory contains a non-constant function symbol (as this allows to build an unbounded number of messages).

2.3 Extensions of the calculus

Although it already allows to model most of the protocol mechanisms that can be encountered in practice, there exist a couple of common extensions of the applied pi-calculus. All can be encoded into the original calculus and thus do not increase its expressivity. They should rather be seen as convenient modelling tools and we will use several of them throughout this thesis, either for modelling purposes or for simplifying some complexity proofs (and in the latter case we naturally consider the impact of the encoding on the size of the process).

Non-deterministic choice A first classical operator is the *non-deterministic choice*: $P+Q$ is a process that can be executed either as P or as Q. Its operational semantics can therefore be described by adding the following rule to those of Figure 1.1:

$$(\{\!\{P+Q\}\!\} \cup \mathcal{P}, \Phi) \xrightarrow{\tau} (\{\!\{R\}\!\} \cup \mathcal{P}, \Phi) \qquad \text{if } R \in \{P, Q\} \qquad \text{(Choice)}$$

This reduction can easily be encoded as an internal communication on a fresh private channel. Typically $P + Q$ can be encoded by the process $\mathsf{new}\ d.(\overline{d}\langle u\rangle.P \mid d(x).Q)$ where u is an arbitrary term and x is a fresh variable. However using a native operator rather than an encoding permits to develop specific optimisations when implementing automated tools. On a theoretical aspect, we also provide in Part III a formal proof that extending the calculus with this operator does not affect the complexity of the decision problems studied in this thesis.

Let bindings and patterns A common convenient tool when defining processes is the *let* construct, binding a term to a variable to increase readability. For example given a message t, the process $\mathsf{let}\ x = t\ \mathsf{in}\ P$ can be seen as a shortcut for $P\{x \mapsto t\}$. However we more generally allow patterns, for example writing $\mathsf{let}\ \langle x, y\rangle = t\ \mathsf{in}\ P\ \mathsf{else}\ Q$ for a process executing $P\{x \mapsto \mathsf{fst}(t), y \mapsto \mathsf{snd}(t)\}$ if t is a message and a pair, and executing Q otherwise. Formally we extend the syntax of the calculus with

$$P, Q \quad ::= \quad \mathsf{let}\ u = v\ \mathsf{in}\ P\ \mathsf{else}\ Q$$

where v is a term and u is a constructor term called a *pattern*, namely a term such that:

1. *It is possible to test that a term matches u using conditionals without* else *branches.*
   Referring to the example above, it is possible to test that a term t is of the form of $u = \langle x, y\rangle$ with the test "$\mathsf{if}\ t = \langle \mathsf{fst}(t), \mathsf{snd}(t)\rangle\ \mathsf{then}$". On the contrary this excludes patterns like $u = \langle \mathsf{rsenc}(x, y, z), \mathsf{rsenc}(x', y, z')\rangle$ that would accept any pair of ciphertexts encrypted using the same randomness. Formally we require that there exist terms $t_1, \ldots, t_n, t'_1, \ldots, t'_n$ (possibly containing a fixed variable x) such that for all ground terms t, t is an instance of u *iff* for all $i \in [\![1, n]\!]$, $\mathsf{msg}(t_i\{x \mapsto t\}, t'_i\{x \mapsto t\})$ and $t_i\{x \mapsto t\}{\downarrow} = t'_i\{x \mapsto t\}{\downarrow}$.
2. *All free variables of u effectively appearing in the rest of the process can be extracted by applying destructors to the pattern.*
   Referring again to the example $u = \langle x, y\rangle$, this condition simply states that x and y can be used later in the process since they could simply be replaced by $\mathsf{fst}(u)$ and $\mathsf{snd}(u)$ respectively. This excludes however patterns such as $u = \mathsf{h}(x)$ where h is a free function symbol. Formally we require that for all variables $x \in vars(u) \cap vars(P)$ (i.e., variables appearing in u and P that are not bound by a previous input or let), there exists a term

context C without free variables such that $\mathsf{msg}(C[u])$ and $C[u]{\downarrow} = x$. The requirement that C does not contain free variables excludes, for example, the pattern $\mathsf{senc}(0, y)$ that accepts the constant 0 encrypted by any key. On the contrary, a pattern $u = \mathsf{rsenc}(x, y, k)$ is valid if $k \in \mathcal{N}$ and $y \notin vars(P)$.

These technical conditions are here to ensure that the let binding can be simulated within the original calculus by using conditionals and destructors. In terms of semantics we have

$$(\{\!\{\mathsf{let}\ u = v\ \mathsf{in}\ P\ \mathsf{else}\ Q\}\!\} \cup \mathcal{P}, \Phi) \xrightarrow{\tau} (\{\!\{R\}\!\} \cup \mathcal{P}, \Phi) \qquad \begin{array}{l} R = P\sigma \text{ if } v =_E u\sigma \\ R = Q \text{ otherwise} \end{array} \qquad \text{(Let)}$$

Similarly, some of our decidability results are also stated in a fragment of the calculus that do not contain any conditionals (if) but use patterns instead of input variables. The semantics of such inputs generalises the rule (In) of Figure 1.1 as follows:

$$(\{\!\{c(u).P\}\!\} \cup \mathcal{P}, \Phi) \xrightarrow{\xi_c(\xi_t)} (\{\!\{P\sigma\}\!\}, \Phi) \qquad \text{if } \xi_c\Phi =_E c,\ \mathsf{msg}(\xi_t\Phi) \text{ and } \xi_t\Phi{\downarrow} = u\sigma \qquad \text{(Pat-In)}$$

The semantics of $c(u).P$ is therefore close to that of $c(x).\,\mathsf{let}\ u = x\ \mathsf{in}\ P$, although not strictly equivalent (the former cannot receive inputs not fitting the pattern whereas the latter can but aborts the execution after it happens).

Private function symbols We may use private function symbols, namely function symbols assumed unknown to the adversary, as a modelling trick to express relations that are not computable by the adversary. For example, when modelling public identities $id_1, \ldots, id_n$ each possessing their own decryption key, the keys in question can be represented by the terms $\mathsf{sk}(id_1), \ldots, \mathsf{sk}(id_n)$ where $\mathsf{sk}/1$ is a private function symbol. This provides a more convenient syntax to express key ownership, and keeping the symbol sk private avoids giving the adversary the unrealistic ability to derive the secret key of an agent from its identity.

Formalising this only requires to change the definition of recipes by imposing that they should not contain private function symbols. In terms of encoding they can be also be simulated by using a name p that will always remai n unknown to the adversary, used as an additional argument to all private function symbols. In the example above, $\mathsf{sk}/1$ would thus be modelled by a usual function symbol $\mathsf{sk}'/2$ and the secret keys by $\mathsf{sk}'(id_1, p), \ldots, \mathsf{sk}'(id_n, p)$.

3 Security properties: equivalences

Section summary

We present the security properties we are interested in, namely *equivalence properties*. We define *static equivalence*, *trace equivalence* and *labelled bisimilarity*, three process equivalences capturing various flavours of privacy-type properties. Two other process equivalences will be introduced later in this thesis (Chapter 5).

3.1 Against a passive attacker: static equivalence

We first define the notion of *static equivalence* that is often used to model security against a passive attacker in that it is only an equivalence of frames, i.e., it does not involve the operational semantics. It expresses that the knowledge obtained by eavesdropping in two different

situations does not permit the attacker to distinguish them. For example no differences can be observed between $\{\mathsf{ax}_1 \mapsto k\}$ and $\{\mathsf{ax}_1 \mapsto k'\}$ if $k, k' \in \mathcal{N}$ because, intuitively, two fresh nonces look like random bitstrings from an external observer's point of view. However the situation is different with the frames

$$\Phi = \{\mathsf{ax}_1 \mapsto k, \mathsf{ax}_2 \mapsto k\} \qquad \Psi = \{\mathsf{ax}_1 \mapsto k', \mathsf{ax}_2 \mapsto k\} \qquad \text{with } k \neq k'$$

Indeed, even if no differences can be made between k and k' in isolation, the attacker observed two identical messages in the first situation Φ but two different messages in the second situation Ψ. In particular we say that the equality test "$\mathsf{ax}_1 = \mathsf{ax}_2$" distinguishes the two frames (because it holds in Φ but not in Ψ). In addition, when the theory is constructor-destructor, we also assume that the adversary can observe destructor failures to distinguish frames, i.e., use the predicate msg. This allows for example to distinguish the following frames:

$$\Phi = \{\mathsf{ax}_1 \mapsto \langle k_1, k_2 \rangle\} \qquad \Psi = \{\mathsf{ax}_1 \mapsto k\}$$

Indeed the recipe $\mathsf{fst}(\mathsf{ax}_1)$ yields a message in the first situation but triggers a decryption failure in the second. Static equivalence has been extensively studied in the literature (see for example [AC06, CDK12, BCD13, CBC11]). We formalise it as follows:

Definition 1.10 (static equivalence)

Two frames Φ and Ψ of same domain are *statically equivalent*, written $\Phi \sim \Psi$, when for all recipes ξ, ζ, $\xi\Phi =_E \zeta\Phi$ *iff* $\xi\Psi =_E \zeta\Psi$. This definition is lifted to extended processes by writing $A \sim B$ when $\Phi(A) \sim \Phi(B)$.

We recall that equality modulo theory $=_E$ only compares messages. In particular by taking $\xi = \zeta$ in the definition, we obtain that if $\Phi \sim \Psi$ then $\mathsf{msg}(\xi\Phi)$ *iff* $\mathsf{msg}(\xi\Psi)$. This formalises that the observation of destruction failures can be used to distinguish frames.

▶ *Example 1.6*

Let us give additional examples to emphasise some considerations related to our running example. The fact that the following two frames are statically equivalent

$$\Phi = \{\mathsf{ax}_1 \mapsto \mathsf{raenc}(m, r, \mathsf{pk}(k))\} \qquad \Psi = \{\mathsf{ax}_1 \mapsto k'\} \qquad \text{with } m \in \mathcal{F}_0 \text{ and } k, k' \in \mathcal{N}$$

intuitively model that encryption makes messages unintelligible (in that the attacker cannot distinguish a ciphertext from a fresh nonce). Naturally this cannot hold anymore once the decryption key is revealed, and indeed $\Phi \cup \{\mathsf{ax}_2 \mapsto k\} \not\sim \Psi \cup \{\mathsf{ax}_2 \mapsto k\}$ as witnessed by the recipe $\xi = \mathsf{radec}(\mathsf{ax}_1, \mathsf{ax}_2)$ whose computation succeeds in the first frame but triggers a decryption failure in the second. Without going to the extreme extent of revealing the key, the two situations are also distinguishable if we weaken the cryptographic assumptions on raenc. For example if we do not suppose the encryption scheme *key concealing* by adding the rule

$$\mathsf{getKey}(\mathsf{raenc}(x, y, \mathsf{pk}(z))) \to \mathsf{pk}(z)$$

then Φ and Ψ are distinguished by the recipe $\mathsf{getKey}(\mathsf{ax}_1)$ which yields a valid message in Φ but not in Ψ. The same holds with the weaker rule $\mathsf{testEnc}(\mathsf{raenc}(x, y, \mathsf{pk}(z))) \to \mathsf{ok}$ modelling that one can test whether a bitstring is a valid ciphertext. ◀

3.2 Against an active attacker: behavioural equivalences

Dynamic extensions of static equivalence consider distinguishability for an attacker interacting actively with the protocol. Consider for example a protocol modelled by a process P manipulating a nonce k. A possible model of the secrecy of k can be formalised by a non-interference statement: there is no observable difference in the behaviour of the protocol when k is replaced by an other term. As we will see in more details in Chapter 2 the same approach can be used to model various flavours of privacy-type properties. such as unlinkability, or ballot privacy in the context of electronic voting. There exist however several candidate equivalences for modelling this notion of indistinguishability.

The zoo of equivalences Indistinguishability can typically be modelled by *trace equivalence* as introduced in [BDNP01]: intuitively two processes are trace equivalent if for all traces of either of the two processes, there exists a trace in the other process that takes the same visible actions and produces a statically equivalent frame. This equivalence has been studied intensively in the context of security protocols [CCLD11, ACK16, CCD13, CKR18a]. However behavioural equivalences have been historically defined as *congruences*, that is, as the indistinguishability of two processes in any adversarial context. This is for example the case of the *testing equivalence* introduced in [AG99], sometimes called *may-testing equivalence* due to its strong inspiration from a well-established equivalence of this name [DNH84]. Informally two processes P and Q are equivalent if for all processes R and all public channels u, $P \mid R$ may input (resp. output) on u after some $\xrightarrow{\tau}$ steps *iff* $Q \mid R$ may as well. In some sense trace equivalence and may-testing equivalence capture a similar notion of security, modelling the actions of the adversary either with a labelled semantics and frames in the case of trace equivalence, or with explicit processes R in the case of may-testing equivalence. Actually it has been proved in [CCD13] that two trace equivalent processes are always may testing equivalent, and that the converse was true provided the processes were *image-finite* (a condition that we do not detail here but that is satisfied by bounded processes among others). This somewhat makes trace equivalence a convenient proof technique for may-testing in that it gets rid of the quantification over adversarial processes, often tedious to manage in proofs.

There also exist equivalences with bisimulation-based definitions. Several of them were introduced for example in [AG99] as proof techniques for may-testing equivalence, including one called *observational equivalence* in [ABF17] that is the targeted equivalence of the popular ProVerif tool [Bla16, CB13]. Intuitively if A_0 and A_1 are observationally equivalent then they should be able to emit on the same channels (just like may-testing equivalence), and for all sequences of $\xrightarrow{\tau}$ steps $A_i \overset{\varepsilon}{\Rightarrow} A'_i$ there should exist $A_{1-i} \overset{\varepsilon}{\Rightarrow} A'_{1-i}$ such that A'_0 and A'_1 are observationally equivalent as well. Similarly to the duo may-testing/trace equivalence, there exists an alternative definition observational equivalence replacing the adversarial processes by our labelled semantics, making proofs easier to manage. This variant is called *labelled bisimilarity* and is proved to coincide observational equivalence in [ABF17].

▶ *Remark: impact of the semantics*
As explained in Section 2.2, variations of the semantics of internal communications (classical, private, eavesdrop) may affect the resulting notion of equivalence. If we stricly stick to the cited references, all abovementioned results are stated in the classical semantics. How these claims can be generalised to the other two semantics is studied in [BCK20]. ◀

Formalisation in our model In this thesis we study trace equivalence and labelled bisimilarity. Since we mostly provide results for bounded processes, they therefore extend to may-testing equivalence and observational equivalence. We define the two equivalences below.

Definition 1.11 (trace equivalence)

If A and B are extended processes, we write $A \sqsubseteq_t B$ when for all traces $A \overset{\mathsf{tr}}{\Rightarrow} A'$, there exists a trace $B \overset{\mathsf{tr}}{\Rightarrow} B'$ such that $A' \sim B'$. We say that A and B are *trace equivalent*, written $A \approx_t B$, when $A \sqsubseteq_t B$ and $B \sqsubseteq_t A$.

Definition 1.12 (labelled bisimilarity)

Labelled bisimilarity $\approx_l$ is the largest symmetric relation such that for all extended processes A, B, $A \approx_l B$ entails 1 $A \sim B$, and 2 for all transitions $A \xrightarrow{\alpha} A'$, there exists a trace $B \overset{\bar{\alpha}}{\Rightarrow} B'$ such that $A' \approx_l B'$, where $\bar{\alpha} = \varepsilon$ if $\alpha = \tau$ and $\bar{\alpha} = \alpha$ otherwise.

In particular labelled bisimilarity is stronger than trace equivalence, i.e., two labelled bisimilar processes are always trace equivalent. Yet the converse is not true as witnessed by the following textbook example, with $a, b, c, 0 \in \mathcal{F}_0$ and using the choice operator introduced in Section 2.3:

$$P = \overline{a}\langle 0\rangle.(\overline{b}\langle 0\rangle + \overline{c}\langle 0\rangle) \qquad Q = (\overline{a}\langle 0\rangle.\overline{b}\langle 0\rangle) + (\overline{a}\langle 0\rangle.\overline{c}\langle 0\rangle)$$

These processes are trace equivalent because they are both able to perform an output on a followed by an output on b or c However they are not labelled bisimilar intuitively because when executing Q, one has to choose at the beginning of the trace whether b or c will be executed at the end (whereas after executing the first output of P it is still possible to output on both b and c).

▶ *Example 1.7*

We refer again to the processes modelling the Private Authentication protocol as described in Example 1.4. We let for instance

$$\bar{A} = \mathsf{Send}(sk_A, \mathsf{pk}(sk_B), N_A) \qquad \bar{B}_Y = \mathsf{Rcpt}(sk_B, \mathsf{pk}(sk_Y), N_B)$$

The process $\bar{A}$ models the role of A sending a connection to B, and $\bar{B}_Y$ models the role of B accepting connections from an agent $Y \in \{A, C\}$. The following equivalence statement models the security property that, when A attempts a connection with B, the adversary cannot infer whether B accepts connections from A or not:

$$(\{\!\{\bar{A} \mid \bar{B}_A\}\!\}, \Phi_0) \approx_t (\{\!\{\bar{A} \mid \bar{B}_C\}\!\}, \Phi_0)$$

The initial frame Φ_0 models that the attacker knows the public keys of all agents involved in the protocol, namely $\Phi_0 = \{\mathsf{ax}_1 \mapsto \mathsf{pk}(sk_A), \mathsf{ax}_2 \mapsto \mathsf{pk}(sk_B), \mathsf{ax}_3 \mapsto \mathsf{pk}(sk_C)\}$. Naturally, an extensive security modelling of private authentication would need to consider more than one session (ideally an unbounded number) of the protocol and more scenarios; typically the property should still hold if A initiates a connection with any agent besides B. We address these concerns in Chapter 2 by proposing a more complete model. Back to the simple equivalence

statement above, it appears to hold; the core argument is that for all messages u_1, u_2, r_1, r_2 and for all public keys $pk_1, pk_2 \in \{\mathsf{pk}(sk_A), \mathsf{pk}(sk_B), \mathsf{pk}(sk_C)\}$, the following frames are statically equivalent:

$$\Phi_0 \cup \{\mathsf{ax}_4 \mapsto \mathsf{raenc}(u_1, r_1, pk_1)\} \qquad \Phi_0 \cup \{\mathsf{ax}_4 \mapsto \mathsf{raenc}(u_2, r_2, pk_2)\}$$

In particular the conclusion still holds if we weaken the cryptographic assumptions on raenc by assuming that a ciphertext is distinguishable from an arbitrary term, which is modelled by adding the rewrite rule $\mathsf{testEnc}(\mathsf{raenc}(x, y, \mathsf{pk}(z))) \to \mathsf{ok}$. However trace equivalence is violated if we add the rule $\mathsf{getKey}(\mathsf{raenc}(x, y, \mathsf{pk}(z))) \to \mathsf{pk}(z)$, meaning that the encryption scheme is not assumed to be key-concealing. A possible attack consists of initiating a connection with B pretending to be A (i.e., using the recipe $\xi = \mathsf{raenc}(\langle N, \mathsf{ax}_1\rangle, r, \mathsf{ax}_2)$ for some $N, r \in \mathcal{F}_0$) and observing which key encrypts B's response. This is formally captured by the following trace:

$$\begin{array}{lll} (\{\!\{\bar{A}, \bar{B}_A\}\!\}, \Phi_0) & \xrightarrow{d(\xi)} (\{\!\{\bar{A}, \bar{d}\langle u\rangle\}\!\}, \Phi_0) & \text{with } u = \mathsf{raenc}(\langle N, N_B, \mathsf{pk}(sk_B)\rangle, r, \mathsf{pk}(sk_A)) \\ & \xrightarrow{\bar{d}\langle \mathsf{ax}_4\rangle} (\{\!\{\bar{A}, 0\}\!\}, \Phi) & \text{with } \Phi = \Phi_0 \cup \{\mathsf{ax}_4 \mapsto u\} \end{array}$$

There is only one trace in the other process taking the same actions, which is the following for some fresh name r':

$$\begin{array}{lll} (\{\!\{\bar{A}, \bar{B}_C\}\!\}, \Phi_0) & \xrightarrow{d(\xi)} (\{\!\{\bar{A}, \bar{d}\langle v\rangle\}\!\}, \Phi_0) & \text{with } v = \mathsf{raenc}(N_B, r', \mathsf{pk}(sk_B)) \\ & \xrightarrow{\bar{d}\langle \mathsf{ax}_4\rangle} (\{\!\{\bar{A}, 0\}\!\}, \Psi) & \text{with } \Psi = \Phi_0 \cup \{\mathsf{ax}_4 \mapsto v\} \end{array}$$

and $\Phi \not\sim \Psi$ because the recipe $\mathsf{getKey}(\mathsf{ax}_4)$ is evaluated to $\mathsf{pk}(sk_A)$ in Φ and to $\mathsf{pk}(sk_B)$ in Ψ. That is, the equality test "$\mathsf{getKey}(\mathsf{ax}_4) = \mathsf{ax}_1$" holds in Φ but not in Ψ. ◀

As a summary the symbolic framework we use allows to detect structural flaws in security protocols, assuming a perfect cryptography up to weakening assumptions that can be added into the model as in the example above. This permits to study the soundness of the protocol's logic and the necessity of specific cryptographic assumptions but is not intended to prove that a given set of assumptions is sufficient to achieve security.

4 Decision problems and complexity

> ***Section summary***
>
> Finally we recall some basic definitions and results of complexity theory that will be used across this thesis. We conclude by defining some decision problems related to security, including several variants appearing in the literature.

4.1 Computational complexity

About sizes Before anything we need to clarify the notion of *size* of the problem parameters since it plays a central role in complexity analyses. This is particularly important for our purpose since there exist several conventions for representing terms. The *tree size* of term t refers to its number of symbols and is written $|t|$. It corresponds to a classical representation

of a term as a ranked tree. But we may also represent terms as DAGs with maximal sharing (which may be exponentially more concise): the *DAG size* of t refers to the cardinality $|subterms(t)|$ and is written $|t|_{\mathsf{dag}}$. This definition is lifted to sets and sequences of terms with the sharing common to all elements of the structure. The size of a signature $\mathcal{F}$ is the sum of the arities of the symbols of $\mathcal{F}$ and the size of a rewrite system $\mathcal{R}$ is the sum of the sizes of the two sides of its rules. The size of a process is the sum of the number of its instructions and of the sizes of all terms appearing in the process. We emphasise that

1. A complexity upper bound stated w.r.t. the DAG size of the inputs is a stronger result than the same upper bound stated w.r.t. the tree size.
2. On the contrary a complexity lower bound stated in DAG size is a weaker result than the corresponding result in tree size.

In this thesis we always adress the strongest configurations: lower bounds are stated w.r.t. the tree representation of terms, and the upper bounds w.r.t. the DAG representation.

▶ *Remark: DAG size of processes*
We recall that our complexity upper bounds are to be understood w.r.t. the DAG size of processes $|P|_{\mathsf{dag}}$ which is here the *number of instructions* of P plus the DAG size of all terms appearing in P. In particular the DAG structure only applies to terms and *not* to the process structure itself. For example n parallel copies of P could technically be represented by an encoding of size $|P|_{\mathsf{dag}} + \mathsf{log}_2(n)$ whereas our notion of size grows linearly with n (which makes our complexity results weaker compared to potential results w.r.t. the logarithmic representation). In some sense we study the relation between the verificaton's hardness and the maximal number of executable instructions of the protocol, rather than its code's length. ◀

Common classes We now shortly remind some background on complexity, mainly introducing our notations. Given $f : \mathbb{N} \to \mathbb{N}$, we write $\mathsf{TIME}(f)$ (resp. $\mathsf{SPACE}(f)$) the class of problems decidable by a deterministic Turing machine running in time (resp. in space) at most $f(n)$ where n is the size of the inputs of the problem. We define the classes:

$$\mathsf{P} = \bigcup_{p\in\mathbb{N}} \mathsf{TIME}(n^p) \qquad \mathsf{LOGSPACE} = \bigcup_{p\in\mathbb{N}} \mathsf{SPACE}(\log(n^p)) \qquad \mathsf{PSPACE} = \bigcup_{p\in\mathbb{N}} \mathsf{SPACE}(n^p)$$

$$\mathsf{EXP} = \bigcup_{p\in\mathbb{N}} \mathsf{TIME}(2^{n^p}) \qquad 2\mathsf{EXP} = \bigcup_{p\in\mathbb{N}} \mathsf{TIME}(2^{2^{n^p}}) \qquad \mathsf{ELEM} = \mathsf{EXP} \cup 2\mathsf{EXP} \cup 3\mathsf{EXP} \cup \dots$$

The class ELEM ("*elementary functions*") therefore contains all problems decidable in time bounded by a tower of exponentials. We also anecdotically mention the class PRIMREC ("*primitive recursive functions*", not formalised here), which strictly contains ELEM and is used in one of the related works we survey later in the thesis. We also define the counterparts of some of these classes for non-deterministic Turing Machine: NLOGSPACE (NL for short), NP, NPSPACE and NEXP. Given a class $\mathcal{C}$, co-$\mathcal{C}$ is the class of problems whose negation is in $\mathcal{C}$.

Theorem 1.1 *(relation between the different classes [Pap03])*
$\mathsf{LOGSPACE} \subseteq \mathsf{NL} = \mathrm{co}\mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP}, \mathrm{co}\mathsf{NP} \subseteq \mathsf{PSPACE} = \mathsf{NPSPACE} \subseteq \mathsf{EXP}$
$\mathsf{EXP} \subseteq \mathsf{NEXP}, \mathrm{co}\mathsf{NEXP} \subseteq 2\mathsf{EXP} \subseteq 2\mathsf{NEXP}, \mathrm{co}2\mathsf{NEXP} \subseteq 3\mathsf{EXP} \subseteq \dots \subset \mathsf{ELEM} \subset \mathsf{PRIMREC}$

Polynomial hierarchy We also mention the notion of *oracle* classes, deciding a problem with a constant-time black box for an other problem: the class of problems decidable in $\mathcal{C}$ with an oracle for a problem Q is noted $\mathcal{C}^Q$. When Q is complete for a class $\mathcal{D}$ w.r.t. a notion of reduction executable in $\mathcal{C}$, we may write $\mathcal{C}^\mathcal{D}$ instead; in particular $\mathcal{C}^\mathcal{D} = \mathcal{C}^{\text{co}\mathcal{D}}$. This kind of reduction is needed to define the last complexity classes we will use in this paper: the *polynomial hierarchy*, which is a collection of complexity classes between P and PSPACE. Indeed the difference between NP and PSPACE lies in quantifier alternation; the usual complete problems considered for these two complexity classes are, given a boolean formula φ:

1 SAT (NP complete): does $\exists x_1, \ldots, x_n.\varphi(x_1, \ldots, x_n)$ hold?

2 QBF (PSPACE complete): does $\forall x_1, \exists y_1, \ldots, \forall x_n, \exists y_n.\varphi(x_1, y_1, \ldots, x_n, y_n)$ hold?

One can therefore consider intermediate versions of these problems, typically a weaker variant of QBF where the number of quantifier alternations is bounded by a constant i. All these intermediate problems are called Σ_n and Π_n and are part of the polynomial hierarchy.

Definition 1.13 (Polynomial hierarchy)

The *polynomial hierarchy* PH consists of the family of complexity classes Σ_n, Π_n and Δ_n defined inductively by

$$\Sigma_0 = \Pi_0 = \Delta_0 = \mathsf{P} \qquad \Sigma_{n+1} = \mathsf{NP}^{\Sigma_n} \qquad \Pi_{n+1} = \text{co}\mathsf{NP}^{\Sigma_n} \qquad \Delta_n = \mathsf{P}^{\Sigma_n}$$

4.2 Complete problems for each complexity class

A problem is said to be *complete* for a complexity class $\mathcal{C}$ if it is in $\mathcal{C}$ and the decision of any problem in $\mathcal{C}$ is reducible to it. For that we use the standard notion *many-to-one polytime reductions* (for the class NP and above) and *logspace reductions* for P. We present below the classical complete problems for some complexity classes, taken from [Pap03]. Typically:

Definition 1.14 (SAT formula)

A *literal* ℓ is either a (boolean) variable or its negation and a *clause* is a disjunction of literals. By convention $\bot$ is the empty clause and $\top$ denotes tautological clauses such as $x \vee \neg x$. A *SAT formula* is then a boolean formula of the form

$$\varphi = \bigwedge_{i=1}^{n} \ell_i^1 \vee \ell_i^2 \vee \ell_i^3 \qquad \ell_i^j \text{ literals}, i \in [\![1, n]\!], j \in [\![1, 3]\!],$$

A *valuation* v of φ is then a mapping from the variables of φ to $\mathbb{B} = \{\top, \bot\}$, and $v(\varphi) \in \mathbb{B}$ denotes the corresponding boolean evaluation of the formula.

Decision problem — SAT

Input: a SAT formula φ

Question: does there exist a valuation v of φ such that $v(\varphi) = \top$?

Theorem 1.2

SAT is NP complete.

It is also common to consider the following variant of satisfiability using a particular type of clauses that can be put under the form of an implication.

Definition 1.15 (Horn formula)

A *Horn clause* is a clause $C = \vee_{i=1}^{p} \ell_i$ with at most one positive literal, that is, there exists at most one $i \in [\![1, p]\!]$ such that ℓ_i is not a negation. A Horn clause $C = x \vee \bigvee_{i=1}^{p} \neg x_i$, x being either a boolean variable or $\bot$, is therefore often written under the more intuitive form

$$C = \left(\bigwedge_{i=1}^{p} x_i \right) \Rightarrow x$$

A *Horn formula* is a conjunction of Horn clauses.

Considering Horn clauses intuitively involves much simpler satisfiability mechanisms than arbitrary clauses and is known to be decidable in linear time in the size of the formula. It is even the canonical complete problem for P:

Decision problem — HornSAT

Input: a Horn formula φ

Question: does there exist a valuation v of φ such that $v(\varphi) = \top$?

Theorem 1.3

HornSAT is P complete.

Now we move on to high complexity classes. To obtain a complete problem for the PSPACE class we usually consider a generalisation of SAT with formula quantified by arbitrary number of operators $\forall$ and $\exists$.

Definition 1.16 (quantified satisfiability)

A *quantified formula* is either a boolean formula or of the form $\forall x, \varphi$ or $\exists x, \varphi$ for some quantified formula φ and variable x. We say that 1 φ holds if it is a boolean formula valued to $\top$, 2 $\forall x, \varphi$ holds if $\varphi\{x \mapsto 0\} \wedge \varphi\{x \mapsto 1\}$ holds, 3 $\exists x, \varphi$ holds if $\varphi\{x \mapsto 0\} \vee \varphi\{x \mapsto 1\}$.

Decision problem — QBF

Input: a quantified formula of the form $\varphi = \forall x_1, \exists y_1, \ldots, \forall x_n, \exists y_n, \psi$

Question: does φ hold?

Theorem 1.4

QBF is PSPACE complete.

As we mentioned earlier, the polynomial hierarchy can be seen as a collection of intermediary classes between NP and PSPACE. Unsurprisingly the complete problems for the hierarchy are also intermediary versions of SAT and QBF, using bounded quantifier alternation.

Decision problem — QBF_n

Input: a quantified formula of the form $\varphi = \forall x_1, \exists x_2, \forall x_3, \ldots, Q\, x_n, \phi$ where $Q = \forall$ if n is odd and $Q = \exists$ if n is even

Question: does φ hold?

Theorem 1.5

For all $n \in \mathbb{N}$, QBF_n is Π_n complete.

To finish we introduce a complete problem for the NEXP class, which is intuitively the counterpart of SAT with formulas of exponential size. The problem consists of considering a boolean circuit that represents an exponentially-bigger formula and to decide whether the formula in question is satisfiable.

Definition 1.17 (boolean circuit)

A *logical gate* is a boolean function with at most two inputs. We assume without loss of generality that the gate has at most two (identical) outputs, to be given as input to other gates. Logical gates range over $\bot$, $\top$, $\wedge$, $\vee$ and $\neg$ with the usual truth tables but we may use other common operators such as $=$. A *boolean circuit* is an acyclic graph of logical gates: each input (resp. output) of a gate is either isolated or connected to a unique output (resp. input) of an other gate, which defines the edges of this graph.
A boolean circuit Γ with m isolated inputs and p isolated outputs thus models a boolean function $\Gamma : \mathbb{B}^m \to \mathbb{B}^p$. We write $(c_1, c_2, g, c_3, c_4) \in \Gamma$ to state that $g : \mathbb{B}^2 \to \mathbb{B}$ is a gate of Γ whose inputs x and y are passed through edges c_1 and c_2, respectively, and whose output $g(x, y)$ is sent through edges c_3 and c_4. This notation is lifted to other in-outdegrees in the natural way.

Through binary representation of integers, a boolean circuit $\Gamma : \mathbb{B}^{m+2} \to \mathbb{B}^{n+1}$ can be interpreted as a function $[\![\Gamma]\!] : [\![0, 2^m - 1]\!] \times [\![1, 3]\!] \to \mathbb{B} \times [\![0, 2^n - 1]\!]$. This way, Γ encodes a formula $[\![\Gamma]\!]_\varphi$ with 2^n variables $\vec{x} = x_0, \cdots, x_{2^n-1}$ and 2^m clauses:

$$[\![\Gamma]\!]_\varphi(\vec{x}) = \bigwedge_{i=0}^{2^m-1} \ell_i^1 \vee \ell_i^2 \vee \ell_i^3 \qquad \text{where} \begin{cases} \ell_i^j = x_k & \text{if } [\![\Gamma]\!](i,j) = (0,k) \\ \ell_i^j = \neg x_k & \text{if } [\![\Gamma]\!](i,j) = (1,k) \end{cases}$$

Rephrasing, $[\![\Gamma]\!](i, j)$ returns a sign bit and the j^{th} variable of the i^{th} clause of $[\![\Gamma]\!]_\varphi$. This induces the following question decision problem:

Decision problem — SuccinctSAT

Input: A boolean circuit Γ

Question: Is the SAT formula $[\![\Gamma]\!]_\varphi$ satisfiable?

Theorem 1.6

SuccinctSAT is NEXP-complete.

4.3 Problems studied in this thesis and variations

Equivalence problems Our goal in this thesis is to study the decidability and complexity of static equivalence, trace equivalence and labelled bisimilarity. We thus study the following decision problems (for simplicity we make no difference between the plain process P and the extended process $(\{\!\{P\}\!\}, \varnothing)$):

Decision problem — StatEquiv
Input: A theory E, two frames Φ and Ψ
Question: Are Φ and Ψ statically equivalent for E?

Decision problem — TraceEquiv (resp. Bisimilarity)
Input: A theory E, two processes P and Q
Question: Are P and Q trace equivalent (resp. labelled bisimilar) for E?

These problems are undecidable in general due to the calculus being Turing-complete and we therefore often need to put restrictions on the inputs. For example, when we say that "StatEquiv is decidable for subterm convergent theories" it means that the following problem is decidable:

Decision problem — StatEquiv for subterm convergent theories
Input: A subterm convergent theory E, two frames Φ and Ψ
Question: Are Φ and Ψ statically equivalent for E?

The way we state the problem implies that a complexity analysis has to account for the sizes of all inputs, including the theory. However the treatment of this question is not uniform in the literature. For example some complexity analyses [AC06, Bau07, CCD13] consider the theory as a constant of the problem. Still considering static equivalence for example, it is proved in [AC06] that StatEquiv is decidable in polynomial time for all *fixed* subterm convergent theories; this means that for all such theories E the problem

Decision problem — E–StatEquiv
Input: Two frames Φ and Ψ
Question: Are Φ and Ψ statically equivalent for E?

is in P. This result provides an interesting insight on complexity but it does *not* imply that the earlier problem "StatEquiv for subterm convergent theories" is in P. This would indeed mean that the complexity is also polynomial in the size of the theory, which is not implied by the E-StatEquiv variant. Typically the decision procedure of [AC06] is polynomial in the sizes of the frames as expected but exponential in the size of E, which illustrates the difference between the two formulations. Going further we show in Chapter 7 that the first variant "StatEquiv for subterm convergent theories" is even coNP-hard.

It should also be noted that for the fixed-theory variant E-StatEquiv, P-hardness results do not make sense since the precise complexity may depend of the choice of E. We prove for

example in Chapter 7 that the problem is LOGSPACE for some instances of E and P-complete for others. Therefore a complexity analysis of E-StatEquiv should rather be seen as a family of complexity results—one for each instance of E. For all of these reasons our complexity analyses always make it clear whether they consider the theory as fixed. In this case we refer to the problem as *parametric equivalence* and say by opposition that *general equivalence* is the initial variant with the theory considered as part of the input. We argue that the latter is more relevant today as the theory can now be specified by the user in many automated tools: most of our results are therefore stated in the general-equivalence setting. Still, we prove and survey in Chapter 7 several results in the parametric-equivalence setting as well to highlight the impact on complexity of choosing one setting or the other.

Single-process problems Finally, although we study it only marginally in this thesis, we also mention the following decision problem

Decision problem — Deducibility

Input: a theory E, a frame Φ, a term u

Question: Is u deducible from Φ, i.e., does there exist a recipe ξ such that $\xi\Phi =_E u$?

This problem is reducible to non static-equivalence [AC06]: more precisely given a theory E, a frame Φ and a term u, then u is deducible from Φ in E *iff* the following frames are *not* statically equivalent in the extension of E with (constructor-destructor or equational) symmetric encryption:

$$\Phi_0 = \Phi \cup \{\mathsf{ax}_{|\Phi|+1} \mapsto \mathsf{senc}(0, u)\} \qquad \Phi_1 = \Phi \cup \{\mathsf{ax}_{|\Phi|+1} \mapsto \mathsf{senc}(1, u)\} \qquad 0, 1 \in \mathcal{F}_0$$

This transformation preserves subterm convergence and the constructor-destructor property. In particular, even under the assumption that the theory verifies these two properties, the decidability of StatEquiv implies that of Deducibility. A dynamic extension of the deducibility problem would be for example:

Decision problem — Weak Secrecy

Input: A theory E, a process P, a term u

Question: Does there exist a trace $P \overset{\mathsf{tr}}{\Rightarrow} (\mathcal{P}, \Phi)$ such that u is deducible from Φ?

When the answer to this question is negative, we consider that u verifies a form of secrecy in the sense that the adversary cannot entirely recover it after interacting with the protocol. This secrecy is however said to be "weak" in the sense that it does not capture partial information leaks: for example if u is of the form $u = \langle u_1, \ldots, u_{100} \rangle$ with $u_1, \ldots, u_{99}$ public terms but u_{100} is weakly secret, then u is also weakly secret although the adversary can reconstruct most of it. Stronger notions of secrecy are formalised using equivalence properties, under the form of *non-interference* statements as in Chapter 2. In particular Weak Secrecy can typically be reduced to non TraceEquiv, for example by observing that the following points are equivalent:

1. there exists a trace $P \overset{\mathsf{tr}}{\Rightarrow} (\mathcal{P}, \Phi)$ such that u is deducible from Φ
2. $Q(0) \not\approx_t Q(1)$ where $Q(v) = P \mid c(x).\,\mathsf{if}\ x = v\ \mathsf{then}\ \overline{c}\langle 0 \rangle$ for some fresh constants $0, 1, c \in \mathcal{F}_0$

Chapter 2:

Formalisations of privacy using equivalences

Summary.
In this chapter we describe, in the applied pi-calculus, several cryptographic protocols and privacy-type properties. When we present the property for an unbounded number of sessions we explain how to restrict the model to the bounded case. The restriction is sometimes more subtle than simply replacing replications by n parallel copies of the processes as doing so tends to fix the *scenario* (which participant executes which session, which one has been compromised by the adversary...) in a too restrictive way. Our models aim at being parametric in the scenario and we achieve this by involving the adversary in its construction.

NB. *These models are used throughout this thesis to benchmark the scalability of our verification techniques. Our benchmarks also include other protocols modelled along the same lines and all are publicly available at [Rak20].*

1 The private authentication protocol

> ***Section summary***
> We formalise common privacy goals in authentication protocols by *non-interference* statements, continuing the example of the Private Authentication protocol of Chapter 1. This includes all scenarios of *anonymity* and *strong secrecy* of the messages under various assumptions.

We illustrate how to model security properties in typical authentication protocols by continuing the example of the Private Authentication Protocol described in the previous chapter. Using the extensions of the calculus described in Chapter 1, Section 2.3, we let a private (constructor) function symbol $\mathsf{sk}/1$, the term $\mathsf{sk}(X)$ modelling the key of the agent X. For convenience we also include in the theory the following rewrite rule that allows to recover the identity of an agent from its public key:

$$\mathsf{getID}(\mathsf{pk}(\mathsf{sk}(x))) \to x$$

Then using the notations of Chapter 1, Example 1.4, we consider again the processes

$$\mathsf{Send}(sk_A, pk_C, N_A) \qquad\qquad \mathsf{Rcpt}(sk_B, pk_D, N_B)$$

that model one session of the protocol, i.e., the roles of 1 A initiating a communication with C with nonce N_A; and 2 B accepting a connection from a unique identity D with nonce N_B. Several sessions of the protocol can be obtained by putting in parallel several copies of P.

1.1 Authentication and reachability properties

Although this is not a privacy-type property, we present briefly and informally the first security goal of the protocol to illustrate how behavioural equivalences can capture other kinds of security properties. The property in question is that, after one successful session of the protocol with $C = B$ and $D = A$, the two agents effectively agree on the nonces N_A and N_B despite the potential adversarial interferences. If we write $Q = \mathsf{Send}(sk_A, pk_B, N_A) \mid \mathsf{Rcpt}(sk_B, pk_A, N_B)$ then this authentication property states that for all traces of Q such that A accepts a nonce N, B has accepted the nonce N_A earlier and $N = N_B$. This is a typical example of a *reachability* property and they can be encoded as equivalences.

However this is a rather inefficient approach for several reasons. First of all the complexity of deciding equivalence is significantly (often exponentially) higher than deciding reachability; see the results of Part III, Chapter 7 for more details on that fact. Second this makes the modelling unnatural and error-prone compared to frameworks specialised in reachability properties. For the sake of completeness we wrote a model of this property but a more scalable approach would rely on frameworks and tools natively supporting rechability properties like ProVerif [BSCS20] or Tamarin [SMCB13]. Our encoding of authentication consists of verifying the equivalence of $\bar{Q}$ and $\bar{Q}_{\mathsf{check}}$, where:

1. $\bar{Q}$ is a process executing Q and then outputting publicly on a dedicated channel a summary of the communication (i.e., the public keys and the nonces of the two participants)
2. $\bar{Q}_{\mathsf{check}}$ does the same thing but outputs the summary only after a synchronisation with B, using a private channel, checking that the nonce accepted by A is the one sent by B.

All in all $\bar{Q}$ and $\bar{Q}_{\mathsf{check}}$ are trace equivalent intuitively if they perform the final summary output in the same situations. That is, A can accept a nonce N only if N has indeed be sent by B (after accepting the nonce of A itself), which is the expected property.

The property can be verified for several sessions of the protocol by putting instances of P in parallel to $\bar{Q}$ and $\bar{Q}_{\mathsf{check}}$. Note however that upon completion of a session between A and B, A only has the guarantee of an authentication with one instance of B (but this could be any of the sessions initiated by B with A). This translates as the fact that

$$\bar{Q} \mid R \not\approx_t \bar{Q}_{\mathsf{check}} \mid R \qquad \text{with } R = \mathsf{new}\ N_A.\, \mathsf{new}\ N_B.\, Q$$

These limitations are already discussed in the paper describing the protocol [AF04] and have to be mitigated by other mechanisms in the communication following the authentication. For this reason we aim at verifying a property that does not consider this as a security issue. A simple workaround consists of only considering parallel sessions R that may involve A or B but not both in the same session. We refer to frameworks specialised in reachability properties to express more subtle formalisations of the property.

1.2 Nonce secrecy and anonymity

Non interference We now model the remaining security objectives of the protocol, namely that it should conceal the identities of the participants (including C the recipient of A and D the connection accepted by B) and the values of the exchanged nonces. A possible formalisation is that there should not be any observable difference in $\mathsf{Send}(A, C, N_A)$ and $\mathsf{Rcpt}(B, D, N_B)$ when replacing the identities by others and the nonces N_A, N_B by any other

(public) value. For example in a simplistic scenario involving only the role of the sender this would mean that,

$$\forall u_1, \ldots, u_6 \in \mathcal{T}(\mathcal{F}_{\mathsf{c}} \cup \mathcal{F}_0),\ (\{\!\{\mathsf{Send}(sk_{u_1}, pk_{u_2}, u_3)\}\!\}, \Phi_0) \approx (\{\!\{\mathsf{Send}(sk_{u_4}, pk_{u_5}, u_6)\}\!\}, \Phi_0)$$

where $\approx$ is either $\approx_t$ or $\approx_l$ and Φ_0 is a frame containing the public keys of all identities involved. The property can then naturally be extended by adding Rcpt in the scenario (and extending the universal quantification over its argument), and even considering several copies of Send and Rcpt with their own arguments to represent several parallel sessions of the protocol. This models a form of non-interference property and has been called *strong secrecy* in [Bla04] which corresponds to the query noninterf of the ProVerif tool [BSCS20]. Verifying this property for some fixed values of the u_i's can be seen as a privacy analysis in a particular scenario: this is typically what has been done in Chapter 1, Example 1.7 which models the anonymity of D assuming A wants to communicate B. In some sense, the strong secrecy based modelling captures privacy in all scenarios.

It is possible to express strong secrecy as a compact equivalence statement. This relies on the fact that the following points are equivalent for all processes $Q(x)$, for some fixed $a \in \mathcal{F}_0$:

(i) for all $u, v \in \mathcal{T}(\mathcal{F}_{\mathsf{c}} \cup \mathcal{F}_0)$, $Q(u) \approx Q(v)$
(ii) for all $u \in \mathcal{T}(\mathcal{F}_{\mathsf{c}} \cup \mathcal{F}_0)$, $Q(u) \approx Q(a)$
(iii) $c(x).Q(x) \approx c(x).Q(a)$

That is, we model strong secrecy (i) as the indistinguishability of the situations where the secret value is either replaced by a constant or an attacker-chosen term (iii). The intermediary step (ii) uses the transitivity of $\approx$ to simplify the definition and make a clearer link between (i) and (iii). A more classical encoding is the equivalence

$$c(x).c(y).Q(x) \approx c(x).c(y).Q(y)$$

for example mentioned in the ProVerif manual [BSCS20]. However our equivalent encoding, by requiring only one input on c for each secret parameter instead of two, is more compact and thus easier to analyse by automated tools.

Security analysis We thus propose the following equivalence as a model of privacy in the protocol (using the patterned inputs as introduced in Chapter 1, Section 2.3):

$$(\{\!\{!\,\mathsf{KeyGen} \mid !\,\mathsf{AdvRole}\}\!\}, \Phi_0) \quad \approx \quad (\{\!\{!\,\mathsf{KeyGen} \mid !\,\mathsf{CstRole}\}\!\}, \Phi_0) \tag{SS}$$

$$\begin{aligned}
\text{where} \quad \mathsf{KeyGen} &= \mathsf{new}\ id.\, \overline{g}\langle \mathsf{pk}(\mathsf{sk}(id))\rangle \quad \text{and} \quad \Phi_0 = \{\mathsf{ax}_1 \mapsto \mathsf{pk}(\mathsf{sk}(a))\} \\
\mathsf{AdvRole} &= d(\langle role, x_1, x_2, x_3\rangle).\ \mathsf{Role}(role, \mathsf{sk}(x_1), \mathsf{pk}(\mathsf{sk}(x_2)), x_3) \\
\mathsf{CstRole} &= d(\langle role, x_1, x_2, x_3\rangle).\ \mathsf{Role}(role, \mathsf{sk}(a), \mathsf{pk}(\mathsf{sk}(a)), a) \\
\mathsf{Role}(sk, pk, N) &= \mathsf{if}\ role = \mathsf{send}\ \mathsf{then}\ \mathsf{Send}(sk, pk, N) \\
&\quad\ \mathsf{else\ if}\ role = \mathsf{rcpt}\ \mathsf{then}\ \mathsf{Rcpt}(sk, pk, N)
\end{aligned}$$

with $g, d, a, \mathsf{send}, \mathsf{rcpt} \in \mathcal{F}_0$. Intuitively the process KeyGen generates a fresh identity id and reveals its public key to the attacker (who can retrieve id by using the function symbol getID). When replicated, this process therefore allows the attacker to involve an arbitrary number of agents in the protocol. Then the processes AdvRole and CstRole both receive session data from

the attacker on a channel d, and then either execute a role of the protocol with this data or a constant value a, respectively. We can also consider restricted models where the number of protocol participants is bounded by an integer n. This leads to the following equivalence

$$(\{\!|!^{2n}\,\mathsf{KeyGen} \mid !^{n}\,\mathsf{AdvRole}|\!\}, \Phi_0) \quad \approx \quad (\{\!|!^{2n}\,\mathsf{KeyGen} \mid !^{n}\,\mathsf{CstRole}|\!\}, \Phi_0) \qquad (\mathsf{SS}_n)$$

where $!^n P$ is syntactic sugar for n identical parallel copies of P. Note that each role involves 2 identities which is why n protocol participants may involve $2n$ identities in total.

Proposition 2.1 (bounded model of strong secrecy)
If (SS_n) holds for all $n \in \mathbb{N}$ then (SS) holds as well.

This proposition essentially follows from the observation that for any processes P, Q, if $!^n P \approx !^n Q$ for all $n \in \mathbb{N}$ then $!\,P \approx !\,Q$. This is a proof technique that proves equivalences session by session. However the converse does not hold a priori: consider for example, for $c, \mathsf{0}, \mathsf{1} \in \mathcal{F}_0$, the processes

$$P = \overline{c}\langle \mathsf{0} \rangle + \overline{c}\langle \mathsf{1} \rangle \qquad\qquad Q = \overline{c}\langle \mathsf{0} \rangle \mid \overline{c}\langle \mathsf{1} \rangle$$

We have $!\,P \approx !\,Q$ but $P \not\approx Q$. Using the modelling (SS_n) we can then draw several conclusions when experimenting with this security model under different cryptographic assumptions:

1. *The encryption needs to be randomised* (experiment: the strong secrecy of the nonces is violated when using aenc instead of raenc). This is however not surprising and this assumption is completely standard.
2. *A ciphertext does not need to be indistinguishable from an arbitrary bitstring* (experiment: equivalence still holds with the additional rewriting rule $\mathsf{testEnc}(\mathsf{raenc}(x, y, \mathsf{pk}(z))) \to \mathsf{ok}$). This somewhat strengthens the security proof by weakening the cryptographic assumptions.
3. *The encryption function needs to be key concealing* (experiment: trace equivalence is violated with the rewriting rule $\mathsf{getKey}(\mathsf{raenc}(x, y, \mathsf{pk}(z))) \to \mathsf{pk}(z)$). Otherwise there is an observable difference between the responses of B upon accepting or refusing a communication. We refer to the Chapter 1, Example 1.7 for an example of an attack trace.
4. *The decoy message of B is necessary* (experiment: trace equivalence is violated when the decoy output $\overline{d}\langle \mathsf{raenc}(n, r, \mathsf{pk}(s)) \rangle$ is omitted). This also holds if B only emits a decoy message when refusing a connection but not when failing to decrypt the received ciphertext. The attack traces show that observing whether B responds to particular messages breaks the anonymity of B itself, among others.

1.3 Privacy in presence of compromised sessions

Context and model The above equivalences (SS) and (SS_n) only formalise privacy in a context where no agents are compromised, that is, we operate under the trust assumption that the adversary controls the communication network but does not engage in a session of the protocol themselves. This weakens the security guarantees offered by the analysis: for example, although A's anonymity cannot be preserved during a session with a compromised agent C (because the attacker can decrypt all messages sent to C), one would expect that the protocol still preserves the anonymity other honest agents subsequently communicating

with A. Although compromised roles are already impliticly modelled within the attacker capabilities, the current formalisation does not permit honest agents to engage a session with them. In a recent work [GHS+20] a similar approach typically uncovered a flaw on some Noise protocols where the adversary had to engage as a protocol participant to break the anonymity of another, honest agent. As we will see, using this refined attacker model unveils a violation of strong secrecy in the Private Authentication Protocol too.

To capture this kind of scenaarios, it sufficies to allow the adversary to directly choose the public key of the recipient (instead of its identity): the key may then be a honest key $\mathsf{pk}(\mathsf{sk}(id))$ forwarded from KeyGen or a key of the form $\mathsf{pk}(sk)$ where $sk \in \mathcal{F}_0$ is generated by the adversary. To avoid trivial security violations, the identity and the nonce of a role communicating with a compromised recipient have to be discarded from the security analysis, that is, non-interference is not applied to these parameters. For that we use an additional rewrite rule, purely for modelling purposes, to test compromised keys:

$$\mathsf{isHonest}(\mathsf{pk}(\mathsf{sk}(x))) \to \mathsf{ok}$$

Note that the test "$\mathsf{isHonest}(k) = \mathsf{ok}$" can be replaced by "$\mathsf{getID}(k) = \mathsf{getID}(k)$" if the theory is constructor-destructor, although the first one is arguably more readable. This eventually leads to the following model of privacy with compromised sessions:

$$(\{\!|\,!\,\mathsf{KeyGen} \mid !\,\mathsf{AdvRole}_c\,|\!\}, \Phi_0) \quad \approx \quad (\{\!|\,!\,\mathsf{KeyGen} \mid !\,\mathsf{CstRole}_c\,|\!\}, \Phi_0) \qquad (\mathsf{SS}_c)$$

where $\mathsf{AdvRole}_c = d(\langle role, x_1, x_2, x_3\rangle).\ \mathsf{Role}(role, \mathsf{sk}(x_1), x_2, x_3)$

$\mathsf{CstRole}_c = d(\langle role, x_1, x_2, x_3\rangle).$
$\mathsf{if}\ \mathsf{isHonest}(x_2) = \mathsf{ok}\ \mathsf{then}\ \mathsf{Role}(role, \mathsf{sk}(a), \mathsf{pk}(\mathsf{sk}(a)), a)$
$\mathsf{else}\ \mathsf{Role}(role, \mathsf{sk}(x_1), x_2, x_3)$

As before a version for a bounded number of sessions can be easily derived.

▶ *Remark: case of several accepted connections*
In our modelling of the protocol the recipient B only accepts connections from a unique identity A. A more general model could let B accept identities from a list L of registered agents. When modelling privacy in this context, the discussion above is adapted as follows:

1. to avoid trivial violations, privacy should not apply to the identity and the nonce of B if there is *at least one* compromised identity in L
2. even if an identity of L is compromised, we expect that the protocol preserves the anonymity of the other honest identities of L, that is, non-interference should apply to them. ◀

Security implications Using this more general model uncovered a violation of anonymity not captured by (SS); more precisely, the attack showed that there were some (public) values of the nonce N_B that could be used to break the anonymity of the agent that generated it. Although not translating into a real attack if N_B is effectively a "fresh unpredicable nonce" as required by the specification of the protocol [AF04], we describe the attack scenario below to highlight what kinds of issues arise from using nonces that are not purely-random bitstrings. For that we recall the definition of the recipient process proposed in Chapter 1:

$$\begin{aligned}\mathsf{Rcpt}(s,p,n) = d(x).\ &\mathsf{new}\ r.\\ &\mathsf{let}\ \langle y,p\rangle = \mathsf{radec}(x,s)\ \mathsf{in}\ \overline{d}\langle\mathsf{raenc}(\langle y,n,\mathsf{pk}(s)\rangle,r,p)\rangle\\ &\mathsf{else}\ \overline{d}\langle\mathsf{raenc}(n,r,\mathsf{pk}(s))\rangle\end{aligned}$$

The security violation relies on the fact that, for some instances of n, the decoy message $\mathsf{raenc}(n, r, \mathsf{pk}(s))$ can be forwarded to another recipient as the initial message x. Consider for example a scenario where, under some circumstances, an honest recipient B ends up using a nonce $N_B = \langle u, pk_C \rangle$ for some term u and pk_C the public key of a compromised agent. Say that B outputs the decoy message $m = \mathsf{raenc}(N_B, r, \mathsf{pk}(s))$ after refusing a connection from an honest agent A; then for any identity id willing to receive communications from C, the attacker can check whether $B = id$ by sending m to id to initiate a session:
1 if $B = id$ then id will accept and respond with a message encrypted with pk_C
2 if $B \neq id$ then id will refuse and respond with a decoy message encrypted with pk_{id}
In particular the attacker can check whether $B = id$ by testing whether decrypting the response of id with sk_C succeeds or not, which breaks B's anonymity. This attack can be discarded from the model by considering that the strong secrecy of the parameters should not extend to the decoy nonces, which means replacing the else branch of $\mathsf{Rcpt}(s, p, n)$ by

$$\ldots \text{else new } n'.\, \overline{d}\langle \mathsf{raenc}(n', r, \mathsf{pk}(s)) \rangle$$

We study this patched property in Chapters 3 and 5 using automated tools, proving it to hold for up to 5 honest participants.

2 The Basic Access Control protocol

> ***Section summary***
>
> We study *unlinkability*, namely the impossibility to observe that two protocol sessions have been executed by the same person. This captures a form of non-traceability of the users. We illustrate it with the Basic Access Control protocol of the *European e-passport* (simplified here for readability).

2.1 Description of the passport and reader processes

We describe in this section a simplified control flow of the Basic Access Control protocol (BAC) implemented in the European electronic passport. The experiments conducted in later chapters use a more accurate model, closer to the original specification [For04]. The protocol consists of a communication between a passport and a reader using the RFID tag of the passport. In a preliminary phase, a reader obtains the private key k of a passport through a trusted channel (in practice the key is printed on the passport and is obtained by the reader using an optical scanner). In the protocol's graphical description below this is represented using dotted arrows. This is then followed by a public challenge using the shared key k.

Passport	$\dashrightarrow$	Reader	:	k	
	$\longleftarrow$		:	getChallenge	
	$\longrightarrow$		:	n	
	$\longleftarrow$		:	$\mathsf{rsenc}(n, r, k)$	bound as x
	$\longrightarrow$		:	ok	if $\mathsf{rsdec}(x, k) = n$
				error	otherwise

where $n, r \in \mathcal{N}$ are fresh and $\mathsf{getChallenge}, \mathsf{ok}, \mathsf{error} \in \mathcal{F}_0$. In particular, the passport triggers an error when it receives a communication originated from a reader that does not use the expected key k, i.e., a reader that has been paired with an other passport during the preliminary

phase. A typical security concern would be whether this could help linking different sessions of the protocol or not. We model this as follows, with $c, d \in \mathcal{F}_0$ and $distr \in \mathcal{N}$:

$$\begin{aligned}
\mathsf{Passport}(k) = \; & \overline{distr}\langle k\rangle \\
& c(x_0).\, \mathsf{if}\ x_0 = \mathsf{getChallenge}\ \mathsf{then} \\
& \mathsf{new}\ n.\, \overline{c}\langle n\rangle.\, c(x). \\
& \mathsf{if}\ \mathsf{rsdec}(x,k) = n\ \mathsf{then}\ \overline{c}\langle \mathsf{ok}\rangle \\
& \mathsf{else}\ \overline{c}\langle \mathsf{error}\rangle
\end{aligned}
\qquad
\begin{aligned}
\mathsf{Reader} = \; & distr(x_k) \\
& \overline{d}\langle \mathsf{getChallenge}\rangle. \\
& d(x_n). \\
& \mathsf{new}\ r.\, \overline{d}\langle \mathsf{rsenc}(x_n, r, x_k)\rangle
\end{aligned}$$

We use two different channels c and d to model that the adversary can distinguish between the messages sent by the passport from those sent by the reader. Since these channels are public, this does not affect the executability of the protocol in the private semantics.

2.2 Modelling unlinkability

According to the ISO/IEC 15408-2 standard a protocol provides unlinkability if it "*ensures that a user may make multiple uses of [the protocol] without others being able to link these uses together*". This therefore models a property of non-traceability of the users and is often modelled as the indistinguishability of two sets of protocol sessions involving several times the same user in one case and not in the other. However as discussed in [BCEDH12, FHMS19] there exist several formalisations unlinkability. Referring to the terminology of [ACRR10] we can mention *weak unlinkability* and *strong unlinkability*, depending on whether trace equivalence or bisimilarity is used to model indistinguishability. Since our objective is mostly to benchmark the scalability of our verification techniques, we sticked to weak unlinkability as it fits our implementations (that target trace equivalence, see Chapters 3 and 5) and permitted to develop more succinct models. Concretely we model unlinkability by the following equivalence statement:

$$!\,\mathsf{Reader} \mid !\,\mathsf{new}\ k.\, !\,\mathsf{Passport}(k) \quad \approx_t \quad !\,\mathsf{Reader} \mid !\,\mathsf{new}\ k.\, \mathsf{Passport}(k) \qquad (\mathsf{UL}^{\approx_t})$$

The two sides of the equivalence put in parallel an arbitrary number of readers and passports, the only difference being that several identical passports are allowed to occur on the left side (whereas all passports are fresh on the right side). In particular the trace inclusion from right to left is trivially verified and the property is therefore equivalent to

$$!\,\mathsf{Reader} \mid !\,\mathsf{new}\ k.\, !\,\mathsf{Passport}(k) \quad \sqsubseteq_t \quad !\,\mathsf{Reader} \mid !\,\mathsf{new}\ k.\, \mathsf{Passport}(k) \qquad (\mathsf{UL}^{\sqsubseteq_t})$$

Let us then study how this property can be adapted for a bounded number of sessions, say, for n passport and readers. For that we consider a fixed set K of n distinct names; the two inclusions of $(\mathsf{UL}^{\approx_t})$ can then be rephrased respectively as

$$\begin{cases} \forall k_1, \ldots, k_n \in K, \mathsf{Dupl}(k_1, \ldots, k_n) \sqsubseteq_t \mathsf{Fresh} \\ \exists k_1, \ldots, k_n \in K, \mathsf{Fresh} \sqsubseteq_t \mathsf{Dupl}(k_1, \ldots, k_n) \end{cases}$$

$$\begin{aligned}
\text{with} \quad & \mathsf{Dupl}(x_1, \ldots, x_n) = !^n\,\mathsf{Reader} \mid \; \mathsf{Passport}(x_1) \mid \cdots \mid \mathsf{Passport}(x_n) \\
\text{and} \quad & \mathsf{Fresh} = !^n\,\mathsf{Reader} \mid !^n\,\mathsf{new}\ k.\, \mathsf{Passport}(k)
\end{aligned}$$

This time again the second inclusion is trivial (it sufficies to choose $k_1, \ldots, k_n$ pairwise distinct) and only the first inclusion matters. We know show how to remove the universal quantification on K by using a similar trick as in the Private Authentication protocol (Section 1), namely involving the adversary in the model to avoid an explicit enumeration of all cases. For that we let a private function symbol $\mathsf{sk}/1$ that takes the identity of the passport's owner (chosen by the adversary) as an argument and returns its private key. This leads to the following model of weak unlinkability for n sessions of the protocol:

$$c(id_1)\ldots c(id_n).\,\mathsf{Dupl}(\mathsf{sk}(id_1),\ldots,\mathsf{sk}(id_n)) \sqsubseteq_t c(id_1)\ldots c(id_n).\,\mathsf{Fresh} \qquad (\mathsf{UL}_n^{\sqsubseteq_t})$$

Similarly to our model of strong secrecy proving this bounded model for all n is a sufficient condition to obtain security in the unbounded case but not a necessary condition a priori. It is also possible to consider another model of unlinkability for a bounded number of sessions capturing the stronger requirement that the adversary should not be able to distinguish between n different passports and any fixed set of (not necessarily identical) n passports. This kind of models are for instance investigated in [CCLD11, CKR18a, CKR19] and are obtained by replacing inclusion with equivalence in the definition of $(\mathsf{UL}_n^{\sqsubseteq_t})$. Formally:

$$c(id_1)\ldots c(id_n).\,\mathsf{Dupl}(\mathsf{sk}(id_1),\ldots,\mathsf{sk}(id_n)) \approx_t c(id_1)\ldots c(id_n).\,\mathsf{Fresh} \qquad (\mathsf{UL}_n^{\approx_t})$$

Proposition 2.2 (relation between the four models of unlinkability)

$$\forall n \in \mathbb{N}, (\mathsf{UL}_n^{\approx_t}) \implies \forall n \in \mathbb{N}, (\mathsf{UL}_n^{\sqsubseteq_t}) \implies (\mathsf{UL}^{\sqsubseteq_t}) \iff (\mathsf{UL}^{\approx_t})$$

The following security conclusions can then be drawn with respect our different models of unlinkability in the BAC protocol:

1. *How error messages are handled may break unlinkability* (experiment: $(\mathsf{UL}^{\approx_t})$ is violated when using an older version of the BAC protocol, not detailed here, using different error messages for different authentication errors). This security violation is mostly known from [CS10, ACRR10]. The flawed protocol was implemented in the French version of the e-passport and has been patched since then leading to the standard we analyse here.
2. *The current BAC standard verifies (weak) unlinkability* (experiment: $(\mathsf{UL}^{\approx_t})$ holds). This fact has been proved in [HBD16] using a frontend to the ProVerif tool. This conclusion can be hinted by the results in the bounded case by Proposition 2.2 as we observed during our own experiments that $(\mathsf{UL}_n^{\sqsubseteq_t})$ held for $n \leqslant 5$; see Chapters 3 and 5 for more details.
3. *In some cases the adversary can non-trivially lower bound the number of distinct passports involved in the communication* (experiment: $(\mathsf{UL}_n^{\approx_t})$ does not hold for any $n \leqslant 2$). This equivalence violation is for example pointed out using automated tools in Chapter 3 or in [CKR18a, CKR19]: for $n = 2$ for example, since the inclusion $(\mathsf{UL}_n^{\sqsubseteq_t})$ holds, this means that after observing two sessions of the protocol, if they were executed by two different passports then the adversary may be able to know it. This is however not standardly considered as a practical security issue in the sense that it is not in contradiction with $(\mathsf{UL}^{\approx_t})$ holding.

2.3 Unlinkability in presence of compromised sessions

As in Section 1.3 we may observe that the security definition $(\mathsf{UL}^{\approx_t})$ implicitly assumes that there are no compromised protocol sessions, that is, that the adversary never initiates a session

with their own passports. As before a compromised key can be modelled by an attacker-generated term and the definition of unlinkability can be refined as follows:

$$\begin{array}{rl} & !\,\mathsf{Reader} \mid !\,a(x).\,\mathsf{Passport}(x) \mid !\,\mathsf{new}\ k.\,!\,\mathsf{Passport}(k) \\ \approx_t & !\,\mathsf{Reader} \mid !\,a(x).\,\mathsf{Passport}(x) \mid !\,\mathsf{new}\ k.\,\mathsf{Passport}(k) \end{array} \qquad (\mathsf{UL}_c^{\approx_t})$$

for some channel $a \in \mathcal{F}_0$. We naturally have $(\mathsf{UL}_c^{\approx_t}) \Rightarrow (\mathsf{UL}^{\approx_t})$. Variants for a bounded number of sessions can also be defined along the same lines as before. Unlike the Private Authentication protocol, we did not identify additional security violations by extending the attacker model this way, although this naturally enriched the set of considered scenarios significantly.

3 E-voting protocols

Section summary

We study *ballot privacy* for a fixed number of voters in a mixnet variant of the Helios e-voting protocol with some modifications inspired by Belenios [CGG19] (stronger zero-knowledge proofs) added to thwart several security issues. We study several formalisations of privacy: one common model in symbolic approaches using *vote swapping* and another model inspired by the *BPRIV computational definition* that permits to abstract the revoting scenario from the process.

3.1 Symbolic and cryptographic definitions of ballot privacy

We study here protocols used to conduct elections remotely. Many security properties may be expected from them, a typical one being *ballot privacy*: it should not be possible to learn more information on a vote than what is revealed by the result of the election. This property has been formalised using several approaches, including symbolic models using trace equivalence [DKR09] but also *computational* (or *cryptographic*) models [BY86, Ben87, BCG$^+$15]. In the latter ballot privacy is formalised as the impossibility, up to negligible probability, for an attacker to guess which of two systems they are interacting with. The two systems typically differ in a parameter related to how participants have voted, which makes this approach similar in spirit to our modelling of privacy in the applied pi calculus with behavioural equivalences. Pushing the analogy further, the two symbolic models of ballot privacy we present later in the section can be seen as the symbolic analogue of two existing computational definitions.

To highlight the link between the two approaches we give below an intuition of the two cryptographic definitions in question and will later formalise our symbolic models using a similar terminology. In computational models the adversary interacts with so-called *oracles* in a black-box manner. They are usually of the following kinds in definitions of ballot privacy:

1. A *voter oracle* $\mathcal{O}_{\mathsf{voter}}$ that simulates the behaviour of the honest voters during the election, that is, the voters that have not been compromised by the adversary.
2. A *cast oracle* $\mathcal{O}_{\mathsf{cast}}$ that simulates the behaviour of dishonest voters, that is, voters that follow the voting instructions of the adversary.
3. A *public bulletin board oracle* that provides the current (public) state of the election. It can typically return, for example, the set of all ballots cast so far.
4. The *tally* that computes the final result of the election that the attacker simulated by its successive calls of the above oracles.

Many formalisations of ballot privacy rely on a secret parameter $\beta \in \{0,1\}$ that affects the $\mathcal{O}_{\mathsf{voter}}$ oracle, that is, how honest voters vote. Privacy is then considered broken if, after interacting with the oracles for a random value of β, the adversary can guess the value of β with probability significantly better than $\frac{1}{2}$. For example an early definition of ballot privacy relies on permutations of honest voters [BY86]: in this definition the voter oracle receives the identity of two honest voters id_1, id_2, two votes v_0, v_1, and casts two ballots for v_0 and v_1 in the name of id_β and $id_{1-\beta}$, respectively. The inability for the attacker to guess β with significant probability intuitively models that a coalition of voters cannot isolate the vote of a specific honest voter. In a more recent definition called BPRIV [BCG+15], the voter oracle receives one honest identity id, two votes v_0, v_1 and casts a ballot for v_β in the name of id. In some sense the attacker therefore constructs two elections in parallel, one for each value of β, but only has access to the public bulletin board corresponding to the value of β they are effectively interacting with. The result announced when tallying is then always the one corresponding to the case $\beta = 0$: this way, the inability for the attacker to guess β with significant probability models that one cannot distinguish the ballots effectively used during an election (case $\beta = 0$) from irrelevant ballots that did not affect the result (case $\beta = 1$).

Our two symbolic definitions of ballot privacy recall the two cryptographic ones above, replacing probabilistic indistinguishability by trace equivalence. More precisely we will study:

1. A well-established model by *vote swap* [DKR09, ACK16] that states that the two voting scenarios obtained by swapping the votes of two honest voters are trace equivalent. This reminds of the cryptographic model by vote permutation mentioned above.
2. A model directly inspired from BPRIV, simulating the interaction adversary-oracles with processes of the applied pi calculus. We are not aware of existing formalisations of BPRIV in a symbolic setting.

We present these definitions in the case of a variant of the Helios protocol [Adi08].

3.2 The Helios protocol

Modelling voting protocols We first introduce some notations for formalising voting systems. We assume a set of votes $\mathbb{V}$ that contains the values the voters vote from and the set of their identities $\mathbb{I}$. We also let R be the *result space* that contains all the possible outcomes of the election; in a majority vote for example we would have $\mathbb{V} = \mathsf{R}$ but the result can also include the vote count, or even the multiset of all ballots for verifiability purpose. The voting system is then split in several components:

1. The *voter* process Voter that simulates the honest voters. A process DVoter is needed as well depending on the model to simulate dishonest voters explicitly.
2. The *ballot box* process $\mathsf{BallotBox}$ that receives the ballots from the voters and stores them into a state BB. Upon receiving a new ballot b, it performs a *validity check* $\mathsf{Valid}(b, \mathsf{BB})$ and only adds b to the ballot box if it succeeds. This test can for instance be a verification of a zero knowledge proof or that an identical ballot is not already in the box.
3. We choose not to model the *public bulletin board* by an explicit process; instead, all honest ballots are revealed to the adversary before being effectively cast.
4. The *tally* process Tally that takes the ballot box BB, the secret key of the election sk, and publishes the result of the election.

The case of Helios We now illustrate this framework by presenting a mixnet variant of the Helios protocol [Adi08]. We also discuss a security issue that leads us to strengthen the zero knowledge proofs in the model. A schematic description is provided in Figure 2.1.

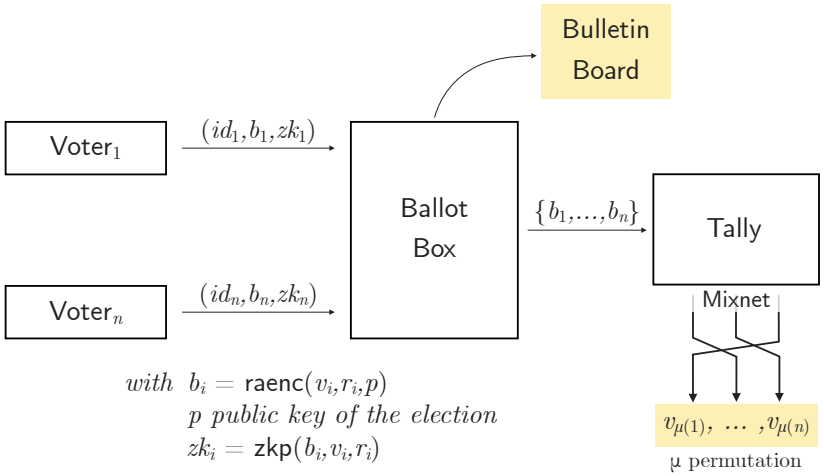


Figure 2.1 Control flow of the Helios protocol ("vanilla")

The voter casts their vote by connecting to a server using a password-based authentication and sending the corresponding ballot from there (which is at the same time published on the public bulletin board). Although the communication between the voters and the ballot box is not private (because its content is published on the bulletin board), this channel is often assumed authenticated due to the password authentication. That is, we assume that the ballot box effectively stores the ballot sent by the voter. Naturally making this assumption implies that the ballot box itself has not been compromised. After the voting phase is closed the ballot box sends all ballots to the tally that decrypts them, shuffles the votes using a mixnet, and eventually outputs all votes in clear.

In Helios a ballot contains the vote encrypted with the public key of the election and a zero-knowledge proof justifying that the ballot is well formed, checked by the ballot box before storing the encrypted vote. This zero-knowledge proof essentially corresponds to the theory described in Chapter 1, Example 1.2.

Trust assumptions We now discuss which of the above components may be trusted or not and discuss, informally at first, the implications in terms of security and modelling. Mainly three entities may be compromised in the system:

1. The *voters*: a compromised voter reveals its secret credentials (such as signing keys, if any) to the adversary and receives its voting instructions from them.
2. The *tally*: a compromised tally reveals the secret key of the election to the adversary, making it critical for security.
3. The *ballot box*: if compromised the ballot box may remove some ballots or modify them before submitting them to the tally.

The case of potential dishonest voters is already explicitly considered by our definitions. In the applied pi calculus, they can be modelled by a dedicated process or by allowing the

ballot box to receive ballots directly from the adversary. Regarding the tally, it needs to be assumed honest to expect privacy. This assumption is backed up by the common practice to split the tally into several independent entities, each possessing a part of the secret key of the election. This way compromising the tally intuitively requires to take control over all of its parts, or a given threshold. Finally we discuss the case of a compromised ballot box, which is notorious for posing many modelling issues [CLW20]. On the one hand assuming an honest ballot box weakens security and is not aligned with practical concerns (in particular the fact that many e-voting systems include verifiability measures precisely so that voters can check that their vote has been correctly included in the ballot box). But on the other hand a completely dishonest ballot box is incompatible with privacy: the attacker could drop all ballots except one and the result of the election would reveal the underlying vote.

Therefore attempts to model a dishonest ballot box usually parametrise their definition with the set of adversarial actions that are not tolerated [CLW20]. For example dropping all ballots but one seems to be an unavoidable privacy loss and this should not be considered as a violation of ballot privacy. In vote-swapping models one usually gets around this issue by a modelling trick allowing the tally to operate only if each honest voter has a ballot in their name in the ballot box [ACK16, CKR18a]. In addition our processes all assume that the channel between the voter and the ballot box is trusted, at least to the extent that an honest vote may not be modified. All in all this essentially limits our dishonest-ballot-box models to dropping a strict subset of the ballots of each voter, and possibly reordering them before they reach the ballot box. As our primary focus is benchmarking we leave open the problem of designing a more realistic symbolic model of a dishonest ballot box.

3.3 Two definitions of ballot privacy

Helios in the applied pi calculus Let us now explain how to model the protocol in the applied pi calculus. First of all the honest-voter process can be defined as follows, where $c \in \mathcal{F}_0$ is a public channel used to publish ballots to the bulletin board, $bb \in \mathcal{N}$ is a private channel between the voter and the ballot box and pk is the public key of the election:

$$\begin{aligned}\mathsf{Voter}(v, id) = \mathsf{new}\ r.\, \mathsf{let}\ ballot = \mathsf{raenc}(v, r, pk)\ \mathsf{in}\\ \mathsf{let}\ proof = \mathsf{zkp}(ballot, v, r)\ \mathsf{in}\\ \overline{c}\langle\langle id, ballot, proof\rangle\rangle.\\ \overline{bb}\langle\langle id, ballot, proof\rangle\rangle\end{aligned}$$

If the voter is dishonest then arbitrary ballots b may be sent. The process is therefore simply $\mathsf{DVoter}(b) = \overline{bb}\langle b\rangle$. Let us now model the ballot box. We represent its internal state by a tuple of n ballots, where n is the fixed number of registered participants $id_1, \ldots, id_n$. We present here the case $n = 2$ for simplicity. This internal state is passed through different parallel copies of the BallotBox process below using a private channel $s \in \mathcal{N}$:

$$\begin{aligned}\mathsf{BallotBox} = s(\langle b_1, b_2\rangle).\ bb(\langle id, b, proof\rangle).\\ \mathsf{if}\ \mathsf{checkzkp}(proof, b) = \mathsf{ok}\ \mathsf{then}\\ \quad \mathsf{if}\ id = id_1\ \mathsf{then}\ \overline{s}\langle\langle b, b_2\rangle\rangle\\ \quad \mathsf{else\ if}\ id = id_2\ \mathsf{then}\ \overline{s}\langle\langle b_1, b\rangle\rangle\\ \quad \mathsf{else}\ \overline{s}\langle\langle b_1, b_2\rangle\rangle\\ \mathsf{else}\ \overline{s}\langle\langle b_1, b_2\rangle\rangle\end{aligned}$$

A ballot box accepting at most p votes will thus be modelled by the process $!^p\,\mathsf{BallotBox}$. Here the channel bb between the voter and the ballot box is authenticated but nothing forces to execute communications on bb after a honest vote has been generated. That is, the ballot box may drop or reorder ballots but a priori nothing more. Another possible model of this (limited) form of non-trusted ballot box is to use a public channel instead of bb but compensating by signing messages (see the theory of signatures of Chapter 1, Example 1.2). This alternative modelling makes the theory more complex but has the advantage of rendering the process DVoter superfluous. On the contrary a model of a completely honest ballot box would have to ensure that the ballot has been cast before carrying the internal state on channel s. This can be done by merging the BallotBox process into the voter processes:

$$
\begin{array}{l}
\mathsf{Voter}^h(v, id) = \\
\quad s(\langle b_1, b_2\rangle).\\
\quad \mathsf{new}\ r.\,\mathsf{let}\ \mathit{ballot} = \mathsf{raenc}(v, r, pk)\ \mathsf{in}\\
\quad \mathsf{let}\ \mathit{proof} = \mathsf{zkp}(\mathit{ballot}, v, r)\ \mathsf{in}\\
\quad \overline{c}\langle\langle id, \mathit{ballot}, \mathit{proof}\rangle\rangle.\\
\quad \mathsf{if}\ id = id_1\ \mathsf{then}\ \overline{s}\langle\langle \mathit{ballot}, b_2\rangle\rangle\\
\quad \mathsf{else\ if}\ id = id_2\ \mathsf{then}\ \overline{s}\langle\langle b_1, \mathit{ballot}\rangle\rangle\\
\quad \mathsf{else}\ \overline{s}\langle\langle b_1, b_2\rangle\rangle
\end{array}
\qquad
\begin{array}{l}
\mathsf{DVoter}^h(x) = \\
\quad s(\langle b_1, b_2\rangle).\\
\quad \mathsf{let}\ \langle id, \mathit{ballot}, \mathit{proof}\rangle = x\ \mathsf{in}\\
\quad \mathsf{if}\ \mathsf{checkzkp}(\mathit{proof}, \mathit{ballot}) = \mathsf{ok}\ \mathsf{then}\\
\quad\quad \mathsf{if}\ id = id_1\ \mathsf{then}\ \overline{s}\langle\langle \mathit{ballot}, b_2\rangle\rangle\\
\quad\quad \mathsf{else\ if}\ id = id_2\ \mathsf{then}\ \overline{s}\langle\langle b_1, \mathit{ballot}\rangle\rangle\\
\quad\quad \mathsf{else}\ \overline{s}\langle\langle b_1, b_2\rangle\rangle\\
\quad \mathsf{else}\ \overline{s}\langle\langle b_1, b_2\rangle\rangle
\end{array}
$$

In this case the ballot box process is not necessary anymore. Finally we define the tallying process that receives the internal state through the channel s and publishes the result of the election. If sk is the secret key of the election:

$$
\begin{array}{rl}
\mathsf{Tally} = & s(\langle b_1, b_2\rangle).\\
& \mathsf{let}\ v_1 = \mathsf{rsdec}(b_1, sk)\ \mathsf{in}\\
& \mathsf{let}\ v_2 = \mathsf{rsdec}(b_2, sk)\ \mathsf{in}\\
& (\overline{c}\langle v_1\rangle \mid \overline{c}\langle v_2\rangle)
\end{array}
$$

The mixnet is modelled by outputting the final votes v_1, v_2 on parallel processes: the vote shuffling is thus represented by the inherent non-determinism of parallel operators. Note also that this model of the tally assumes the ballot box honest to some extent since the final result is output only if a valid vote has been received for each voter. This assumption can be lifted—although this will break ballot privacy in the case of the vote swap definition—by removing the let ... in instructions and directly outputting $\mathsf{rsdec}(b_1, sk)$ and $\mathsf{rsdec}(b_2, sk)$. This way only valid votes (i.e., messages) are tallied but nothing forces all votes to be valid to output a partial result.

Definition by vote swap We formalise ballot privacy in the applied pi calculus as the indistinguishability of two processes where the votes of two honest voters are swapped [DKR09, ACK16]. Let us thus assume that $\mathbb{V} = \{0, 1\}$ and consider the identities of two honest voters $id_1, id_2 \in \mathbb{I}$ and a dishonest voter id_3. We then consider the process modelling the voting system for three voters:

$$\mathsf{VotingSystem}(v_1, v_2) = \mathcal{O}_{\mathsf{voter}}(v_1, v_2, id_1, id_2) \mid !^n\,\mathcal{O}_{\mathsf{cast}} \mid !^p\,\mathsf{BallotBox} \mid \mathsf{Tally}$$

$$\begin{array}{ll}
\text{with} & \mathcal{O}_{\mathsf{voter}}(v_1, v_2, id_1, id_2) = \mathsf{Voter}(v_1, id_1) \mid \mathsf{Voter}(v_2, id_2)\\
\text{and} & \mathcal{O}_{\mathsf{cast}} = c(\langle \mathit{ballot}, \mathit{proof}\rangle).\ \mathsf{DVoter}(\langle id_2, \mathit{ballot}, \mathit{proof}\rangle)
\end{array}$$

This models a system with three voters emitting $n+2$ ballots including n dishonest ones and a ballot box accepting p ballots maximum. Ballot privacy is then modelled by the equivalence:

$$\mathsf{VotingSystem}(0,1) \approx_t \mathsf{VotingSystem}(1,0)$$

There exists a reduction result justifying that it is sufficient to consider two honest voters and a dishonest one to prove ballot privacy for an arbitrary number of voters w.r.t. formalisations of ballot privacy by vote swap [ACK16]. The reference [ACK16] also provides a bound on p (7 if revote is allowed). However there are a priori no bounds on the number of times each voter may revote, which is therefore the only variable parameter of our bounded model. Yet note that, although dishonest revotes are modelled by the n copies of the cast oracle, it is not clear how to model honest revotes with this definition. A simple revoting scenario can still be captured by putting several parallel copies of $\mathcal{O}_{\mathsf{voter}}(v_1, v_2, id_1, id_2)$, thus modelling honest voters that generate several ballots for their fixed candidate. We leave open the modelling of a more general revote scenario w.r.t. this definition.

We finally discuss briefly some of the known security issues related to Helios and how they are reflected in this model.

1 *Helios is not ballot private* (experiment: $\mathsf{VotingSystem}(0,1) \not\approx_t \mathsf{VotingSystem}(1,0)$). Helios is known to be vulnerable to *ballot copy*, a technique introducing a bias in the result of the election to break privacy [CS13]. Looking at a trace violating trace equivalence we obtain the following attack scenario in our context of two honest voters id_1, id_2 and one dishonest voter id_3. The two honest voters vote for v_1 and v_2, respectively. The adversary then copies, from the bulletin board, the encrypted vote of id_2 and the corresponding zero-knowledge proof and asks id_3 to cast them as its own ballot. After tallying the attacker knows that the vote that appears at leat twice in the result is the vote of id_2. A graphical display of the attack can be found in Figure 2.2.

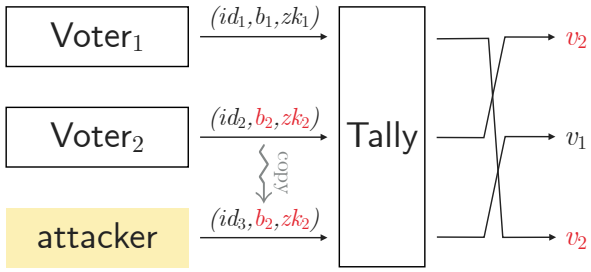


Figure 2.2 Ballot-copy attack

2 *Ballot copy can be mitigated using weeding, but this countermeasure is inefficient if honest voters can generate more than one vote.* The discovery of this issue has been pointed to us by Peter Rønne (private communication, 2016). A natural countermeasure usually considered in Helios against ballot copy is to use a *weeding* procedure, that is, to remove duplicated ballots from the ballot box. Trace equivalence can be proved with this new protocol. However a variant of the ballot-copy attack can be mounted if a honest voter generates several ballots (for example if the first ballot did not reach the ballot box due to adversarial interferences, forcing the voter to generate a new one). Indeed, assuming the honest voter generates two ballots for $v \in \mathbb{V}$ and, say, only the second one reaches the

ballot box, it sufficies to copy and cast in one's name the first ballot to mount a ballot-copy attack despite weeding. This is reflected in the model as the equivalence violation in the simple revote scenario described earlier, i.e., $\mathsf{VotingSystem}^w(0,1) \not\approx_t \mathsf{VotingSystem}^w(1,0)$ where

$$\mathsf{VotingSystem}^w(v_1,v_2) = !^2\,\mathcal{O}_{\mathsf{voter}}(v_1,v_2,id_1,id_2) \mid \mathcal{O}_{\mathsf{cast}} \mid !^2\,\mathsf{BallotBox}^w \mid \mathsf{Tally}$$

where $\mathsf{BallotBox}^w$ is the variant of BallotBox that only accepts a ballot if it passes the zero-knowledge proof and no identical encrypted vote have been received before.

3 *The ballot-copy attack is thwarted by including the identity in the zero-knowledge proofs* (experiment: trace equivalence holds with this new theory). Another possible countermeasure to ballot copy is to streghten the zero-knowledge proofs by including the identity of the voter inside. That is, we add an additional argument to the symbols zkp and checkzkp compared to the theory of Chapter 1, Example 1.2 and use the new rewrite rule:

$$\mathsf{checkzkp}(\mathsf{zkp}(\mathsf{raenc}(m,r,\mathsf{pk}(k)),m,r,id),\mathsf{raenc}(m,r,\mathsf{pk}(k)),id) \to \mathsf{ok}$$

This way it is not possible anymore to cast someone else's ballot in one's own name. Our experiments in Chapters 3 and 5 include the verification of trace equivalence for this variant for up to a total of 10 ballots emitted by two honest voters.

Definition inspired by BPRIV We now explain how to model the BPRIV definition outlined at the beginning of this chapter. We are not aware of existing formalisations of this definition in the applied pi calculus or symbolic models in general. To stick to the formalism of BPRIV [BCG+15] we define

1 the voting oracle $\mathcal{O}^{\beta}_{\mathsf{voter}}$, $\beta \in \{0,1\}$, that generates a vote for v_0 and v_1, casts them in two different ballot boxes, and publish the ballot for v_β on the bulletin board but effectively tallies v_0.

2 the cast oracle $\mathcal{O}_{\mathsf{cast}}$ that casts a ballot b in the name of the dishonest voter id.

We present here the definition of ballot privacy in the case of one honest voter id_1 and one dishonest voter id_2. We first define the two voting oracles:

$$\begin{aligned}\mathcal{O}^{\beta}_{\mathsf{voter}} = c(\langle v_0,v_1\rangle).\\ &\mathsf{new}\ r_0.\,\mathsf{let}\ ballot_0 = \mathsf{raenc}(v_0,r_0,pk)\ \mathsf{in}\\ &\mathsf{new}\ r_1.\,\mathsf{let}\ ballot_1 = \mathsf{raenc}(v_1,r_1,pk)\ \mathsf{in}\\ &\mathsf{let}\ proof_0 = \mathsf{zkp}(ballot_0,v_0,r_0)\ \mathsf{in}\\ &\mathsf{let}\ proof_1 = \mathsf{zkp}(ballot_1,v_1,r_1)\ \mathsf{in}\\ &\overline{c}\langle\langle id_1,ballot_\beta,proof_\beta\rangle\rangle.\ \overline{bb}\langle\langle id_1,ballot_0,proof_0\rangle\rangle\end{aligned}$$

Then the cast oracle is essentially the same process as DVoter:

$$\mathcal{O}_{\mathsf{cast}} = c(\langle ballot,proof\rangle).\ \mathsf{DVoter}(\langle id_2,ballot,proof\rangle)$$

Note that in general the process BallotBox also has β as a parameter since the validity test Valid depends on the state of the bulletin board. However the situation is simpler for zero-knowledge-proof-based validity tests since they only depend on the submitted ballot. Still note that this is not the case for the weeding-based variant of the protocol which checks that

no identical ballots have previously been cast. Here ballot privacy is thus simply defined as $\mathsf{VotingSystem}^0 \approx_t \mathsf{VotingSystem}^1$ where

$$\mathsf{VotingSystem}^\beta = !^m \mathcal{O}^\beta_{\mathsf{voter}} \mid !^n \mathcal{O}_{\mathsf{cast}} \mid !^p \mathsf{BallotBox} \mid \mathsf{Tally}$$

This models a scenario with m calls to the voting oracle, n calls to the cast oracle and p ballots accepted by the ballot box. This covers arbitrary revote scenarios and all security issues witnessed using the vote-swapping model are reflected in this model as well.

▶ *Remark: reduction result for BPRIV*
We mentioned a reduction result for the vote-swap model proving that ballot privacy for an arbitrary number of voters was equivalent to ballot privacy for three voters and a bounded number of ballots accepted by the ballot box [ACK16]. One may conjecture that a similar reduction result could be proved for the BPRIV definition but leave an investigation of this problem to future work. ◀

Part II

DEciding Equivalence Properties in SECurity protocols

Introduction

Automated verification of equivalence properties

In Part I we introduced the applied pi calculus and two notions of behavioural equivalence in this model, trace equivalence and labelled bisimilarity, that can be used to model various forms of privacy in security protocols. In this part we will be more interested in the decidability of these equivalences. We detail the related work and our contributions below.

Decidability results The problem of analysing security protocols is undecidable in general but several decidable subclasses have been identified. While complexity results are known for reachability properties [DLM04, RT03] the case of behavioural equivalences remains mostly open. For active attackers, bounding the number of protocol sessions is often sufficient to obtain decidability results [RT03] and is of practical interest: most real-life attacks indeed only require a small number of sessions. In this context Baudet [Bau07], and later Chevalier and Rusinowtich [CR12], showed that real-or-random secrecy was coNP for subterm convergent theories, by checking whether two constraint systems admit the same set of solutions. These procedures however require some restrictions: they do not allow for else branches, nor do they verify trace equivalence in full generality. In [CCD13], Cheval et al. have used Baudet's procedure as a black box to verify trace equivalence of *determinate* processes. This class of processes (to be defined in Chapter 5) is of practical interest but insufficient for many anonymity properties. Finally, decidability results for an unbounded number of sessions were proposed in [CCD15b, CCD15a], but with severe restrictions on the processes and on the theory (we detail these results more in a survey in Part III, Chapter 7).

Tool support (bounded) There are already mature tools for reachability properties, see ProVerif [Bla16] and Tamarin [SMCB13] for example, but some prototypes also exist for verifying equivalence properties. We start discussing tools that are limited to a bounded number of sessions. The SPEC tool [TD10, TNH16] verifies a sound symbolic bisimulation, but is restricted to particular cryptographic primitives (pairing, encryption, signatures and hash functions) and does not allow for else branches. The APTE tool [CCLD11, Che14] also covers fixed primitives but allows else branches and decides trace equivalence exactly, using a constraint-solving approach similar to ours. On the contrary, the Akiss tool [CCCK16] allows for user-defined cryptographic primitives. The tool is based on Horn clause resolution. Partial correctness of Akiss is shown for primitives modelled by arbitrary theories that have the finite variant property [CLD05]. Termination is additionally shown for subterm convergent rewrite systems. However, Akiss only decide trace equivalence of determinate processes; for other

processes trace equivalence can be both over- and under-approximated which still proves to be sufficient on many examples. The recent SatEquiv tool [CDD17, CDD18] uses a different approach: it relies on Graph Planning and SAT solving to verify trace equivalence, rather than a dedicated procedure. The tool is extremely efficient and several orders of magnitude faster than other tools, but can only handle a fixed set of cryptographic primitives, does not support else branches, and only considers a class of simple processes (a subclass of determinate processes) that satisfy a type-compliance condition. These restrictions severely limit its scope.

Tool support (unbounded) Other tools support verification of equivalence properties, even for an unbounded number of sessions. This is the case of ProVerif [Bla16], Tamarin [SMCB13], Maude-NPA [SEMM14] which all allow for user-defined cryptographic primitives. However, given that the underlying problem is undecidable, these tools may not terminate. Moreover, they only approximate trace equivalence by verifying the more fine-grained *diff-equivalence*. We study this equivalence more precisely in Chapter 5 but we can mention now that it is too fine-grained on many examples. While some recent improvements of ProVerif [CB13, BS18] helps covering more protocols, general verification of trace equivalence is still out of scope. For instance, the verification by Arapinis et al. [AMR+12] of unlinkability in the 3G mobile phone protocols required some "tricks" and approximations of the protocol to avoid false attacks. In [CGLM17, CGLM18], Cortier et al. develop a type system and automated type checker for verifying equivalences. While extremely efficient, this tool only covers a fixed set of cryptographic primitives and does not capture arbitrary proofs of trace equivalence. A different approach has been taken by Hirschi et al. [HBD16], identifying sufficient conditions provable by ProVerif for verifying unlinkability properties, implemented in the tool Ukano, a front-end to the ProVerif tool. Ukanodoes however not verify equivalence properties in general.

The DeepSec prover In Part II we contribute to the practical verification of equivalence properties when the number of sessions is *bounded*. We emphasise again that even in this setting, the system under study has infinite state space due to active attacker controlling the network who can forge messages and use them to interact with the protocol. Our work targets the wide class of *constructor-destructor subterm convergent theories*.

In Chapter 3 we show how the decision of trace equivalence and labelled bisimilarity can be reduced to a form of *constraint solving*, using a novel notion of *partition tree*. This type of approach is rather common for the decidability of equivalences [Bau07, DKR07, LL10, CCD13]. This results in an high-level procedure that takes the form of a tree generation, assuming an oracle to a constraint solver. Before going into the technical details of the solver, we present the implementation of the procedure for trace equivalence, the DeepSec prover, and compare its performances against some of the previously-mentioned tools for equivalence verification in a bounded number of sessions. Chapter 4 is then a more technical chapter that describes in details the constraint solving procedure used in an abstract way in Chapter 3 and give correctness arguments. Termination and complexity will be studied later in Part III.

Proof symmetries Finally in Chapter 5 we develop a new optimisation technique for the decision of trace equivalence that takes the form of a *a sound refinement* of trace equivalence that we call *equivalence by session*. This new equivalence exploits better the process structure

and symmetries that arise during practical verification, to the cost of some occasional *false attacks*. This enables us to handle all examples presented in the experiments of Chapter 3 with a verification time reduced by orders of magnitude by the use of *partial order reductions* (por) and *reductions by symmetry* that we prove correct.

In terms of related work, por techniques for the verification of cryptographic protocols were first introduced by Clark et al. [CJM03]: while well developed in verification of reactive systems these existing techniques do not easily carry over to security protocols, mainly due to the symbolic treatment of attacker knowledge. Mödersheim et al. [MVB10] proposed por techniques that are suitable for symbolic methods based on constraint solving. However, both the techniques of [CJM03] and [MVB10] are only correct for reachability properties. Partial order reduction techniques for equivalence properties were only introduced more recently by Baelde et al. [BDH14, BDH15]: implementing these techniques in the APTE tool resulted in spectacular speed-ups. Other state-of-the-art tools for a bounded number of sessions such as Akiss and even the implementation of DeepSec presented in Chapter 3, integrated these techniques as well. However, these existing techniques are limited in scope as they require protocols to be *determinate*. Examples of protocols that are typically not modelled as determinate processes are the BAC protocol, and the Helios e-voting protocol mentioned in Chapter 2. In recent work, Baelde et al. [BDH18a] propose por techniques that also apply to non-determinate processes (but do not support private channels) and implement these techniques in the DeepSec tool. Unfortunately, these techniques introduce a computational overhead, that limits the efficiency gain. As our experiments will show, our techniques, although including some approximations (potential false attacks), significantly improve efficiency. We also mention the less recent work including some symmetries (but no partial-order) reductions [CDSV04] to a tool, $\mathsf{S^3A}$ [DSV03], that verifies testing equivalence in the spi calculus. The tool however only supports a fixed equational theory and no else branches, and we are not aware of a publicly available implementation. In the experiments of Chapter 5 we will therefore focus on the performance comparison against the procedure evaluated in Chapter 3.

Finally, our approach can also be compared to tools for an unbounded number of sessions—or more precisely to tools that verify *diff equivalence* which takes a similar approach as us in the sense that it is a sound refinement of trace equivalence like equivalence by session. In particular both techniques may fail to prove equivalence of processes that are trace equivalent. However equivalence by session is less fine-grained, for example capturing equivalence proofs for the BAC protocol. A detailed comparison between these two equivalences is given in Chapter 5. Moreover, the restriction to a bounded number of sessions allows us to decide equivalence by session, while termination is not guaranteed for tools in the unbounded case.

Chapter 3:

Automated analysis for bounded processes

Summary.
In this chapter we present an algorithm for proving *equivalence of bounded processes*. The approach is based on a symbolic semantics representing adversarial inputs by *constraints* expressing which values these inputs may take. The overall procedure then relies on a constraint-solving procedure (used in a black box manner here and detailed in Chapter 4 for constructor-destructor subterm convergent theories). This generates a so-called *partition tree* that can be seen, intuitively, as a proof tree that the processes are equivalent. We also discuss implementation choices and evaluate the performances of the resulting automated tool, DeepSec.

1 The symbolic approach for decidability

> ***Section summary***
> We introduce another process semantics replacing adversarial inputs by *constraint systems* and a novel notion of *most general solutions*. This makes the resulting transition system finitely branching (unlike the standard semantics), thus reducing protocol analysis to *constraint solving*.

1.1 Constraint systems

Our decision procedures for process equivalences rely on a symbolic semantics, by opposition to the usual semantics of the calculus that we will call the *concrete* semantics from now. Rather than requiring the attacker to provide concrete recipes, we record the constraints they should satify to reach a certain point in the protocol, thus providing a finite representation of the infinite set of actions available to the attacker. For example let $c \in \mathcal{F}_0$, $\mathsf{h}/1 \in \mathcal{F}_\mathsf{c}$ and

$$P = \mathsf{new}\ k.\, \overline{c}\langle k\rangle.\, c(x).\, \mathsf{if}\ \mathsf{fst}(x) = k\ \mathsf{then}\ \overline{c}\langle \mathsf{h}(x)\rangle$$

The trace executing the output $\mathsf{h}(x)$ will gather constraints that intuitively indicate that: 1 x is a term deducible by the attacker from the frame $\{\mathsf{ax}_1 \mapsto k\}$; and 2 $x = \langle k, y\rangle$ for some term y. A constraint solving algorithm, detailed in Chapter 4, can then be used to show that these constraints have a *solution*: the recipe $\xi = \langle \mathsf{ax}_1, a\rangle$, $a \in \mathcal{F}_0$ can be used to deduce the input term x and satisfy the constraints, thus proving the output of $\mathsf{h}(x)$ to be reachable. Similar approaches are common to decide reachability or equivalence properties [Bau07, CCD13]; our approach has however a larger scope.

Formalising symbolic constraints In order to store the order of output terms in a frame Φ, we assume that $dom(\Phi)$ is always of the form $\{\mathsf{ax}_1, \mathsf{ax}_2, \ldots, \mathsf{ax}_n\}$, that is, axioms are introduced by increasing index order. In order to define constraints we also introduce a new type of variables for recipes:

Definition 3.1 (second-order terms)

From now on we consider a partition of the set of variables $\mathcal{X} = \mathcal{X}^1 \uplus \mathcal{X}^2$. The elements of $\mathcal{X}^1$ are called *first-order variables* and correspond to the variables we used so far in terms (appearing in processes, frames, rewrite rules). Those of $\mathcal{X}^2$ are called *second-order variables* and are used to represent an undefined recipe. A *first-order term* is an element of $\mathcal{T}(\mathcal{F} \cup \mathcal{F}_0 \cup \mathcal{N} \cup \mathcal{X}^1)$ and a *second-order term* is an element of $\mathcal{T}(\mathcal{F} \cup \mathcal{F}_0 \cup \mathcal{A}\mathcal{X} \cup \mathcal{X}^2)$.

We now distinguish $vars^1(u) = vars(u) \cap \mathcal{X}^1$ and $vars^2(u) = vars(u) \cap \mathcal{X}^2$. Note that we say that a second-order term t is ground if $vars^2(t) = \varnothing$, i.e., t may contain axioms. Recalling Definition 1.8, a recipe is therefore a ground second-order term. We also adapt the other notations of the term algebra to reflect the separation: $subterms^1$, $subterms^2$...

When executing an input x in the symbolic semantics, x will be associated to a fresh second-order variable X that will serve as a placeholder for the underlying recipe.

Definition 3.2 (second-order substitutions)

We suppose a partition $\mathcal{X}^2 = \biguplus_{n\in\mathbb{N}} \mathcal{X}^2_{=i}$ where each class $\mathcal{X}^2_{=i}$ is infinite. We also write $\mathcal{X}^2_{\leqslant i} = \bigcup_{j=0}^{i} \mathcal{X}^2_{=j}$. If X is a second-order variable we may write $X : i$ to emphasise that $X \in \mathcal{X}^2_{=i}$ and say in this case that X *is of type* i. A *second-order substitution* is then a substitution Σ of domain $dom(\Sigma) \subseteq \mathcal{X}^2$ that respects the types, that is,

$$\forall X{:}i \in dom(\Sigma),\ X\Sigma \in \mathcal{T}_i^2 \qquad \text{where } \mathcal{T}_i^2 = \mathcal{T}(\mathcal{F} \cup \mathcal{F}_0 \cup \mathcal{X}^2_{\leqslant i} \cup \{\mathsf{ax}_1, \ldots, \mathsf{ax}_i\})$$

Intuitively, if X is a second-order variable associated to an input variable x, the type of X carries the information of which axioms were available at the time x was performed. The restriction that second-order substitutions respect types thus ensures that X will be instantiated by a recipe that is consistent with the frame x is constructed from. We can then define the constraints that we use to characterise the possible values x may take:

Definition 3.3 (constraint)

We consider the following three kinds of atomic formulas:

1. *deduction fact* $\xi \vdash^? u$ where u is a message in normal form and ξ is a second-order term such that $root(\xi) \notin \mathcal{F}_\mathsf{c}$;
2. *second-order equations* $\xi =^? \zeta$ where ξ and ζ are two second-order terms;
3. *(first-order) equations* $u =^? v$ where u and v are two messages in normal form.

A *constraint* is then a first-order formula over these atoms, that is, either $\top$, $\bot$, one of the three atoms above, or of the form $\varphi \wedge \psi$, $\varphi \vee \psi$, $\neg\varphi$, $\forall x.\varphi$, or $\forall X{:}n.\varphi$ for φ, ψ constraints. Note that $vars(\varphi)$ then refers to the *free* variables of the constraint φ. The negation $\neg(\alpha =^? \beta)$ of an equation is written $\alpha \neq^? \beta$ and called a *disequation*.

A deduction fact $\xi \vdash^? u$ indicates that term u is deducible by the recipe ξ and second-order equations $\xi =^? \zeta$ are used to put restrictions on which recipes ξ may be used to do so. For

example $X : i \vdash^? x$ states that the variable x is to be replaced by a term deducible by the attacker using at most the i first outputs of the frame; a constraint solving procedure may then impose that $\exists Y{:}i.\, X =^? \mathsf{f}(Y)$, i.e., that the underlying recipe should have a f symbol at its root. Equations reflect the syntactic equalities that the first-order terms verify. Typically when executing $\mathsf{if}\ \mathsf{fst}(x) = t\ \mathsf{then}\ P\ \mathsf{else}\ Q$, the positive branch will intutively lead to the constraint $\exists y.\, x =^? \langle t, y\rangle$ and the negative branch to $\forall y.\, x \neq^? \langle t, y\rangle$.

Constraint systems Finally we define and give some properties of *constraint systems* that are used to collect the first-order constraints induced by a given execution of a process.

Definition 3.4 (constraint system)

A *constraint system* is a triple $\mathcal{C} = (\Phi, \mathsf{D}, \mathsf{E}^1)$ whose elements are of the following form:

1. $\Phi = \{\mathsf{ax}_1 \mapsto t_1, \ldots, \mathsf{ax}_n \mapsto t_n\}$ is a frame (non necessarily ground)
2. D is a set of constraints of the form $X \vdash^? x$, with $X \in \mathcal{X}^2_{\leqslant n}$ and $x \in \mathcal{X}^1$, or $\forall X{:}i.\, X \nvdash^? x$ for some $i \leqslant n$ and $x \in \mathcal{X}^1$. We also require the *origination property*: for all $i \in [\![1, n]\!]$, for all $x \in vars(t_i)$, there exists $X \in \mathcal{X}^2_{\leqslant i-1}$ such that $(X \vdash^? x) \in \mathsf{D}$.
3. E^1 is a set of constraints of the form $u =^? v$ or $\forall z_1 \ldots \forall z_k.\, \bigvee_{j=1}^{r} u_j \neq^? v_j$.

The components of $\mathcal{C}$ are also written $\Phi(\mathcal{C})$, $\mathsf{D}(\mathcal{C})$ and $\mathsf{E}^1(\mathcal{C})$. The set D contains all input binders x that have been executed, each mapped to a second-order variable X that will serve as a placeholder for the corresponding recipe. The constraints $\forall X{:}i.\, X \nvdash^? x$ indicate that a channel x has been used in an internal communication and should therefore not be deducible by the adversary (which, we recall, is a requirement of the concrete semantics we use, the private semantics). Next the origination property expresses that when reference is made to an input x in an output t_i, this input should be computed only from the previous outputs $t_1, \ldots, t_{i-1}$. This is a natural property preventing cyclic input-output dependencies. Finally E^1 is a set of (dis)equalities imposed on the protocol messages by conditionals, among others. We will formalise in Section 1.3 the semantics of these constraints through a notion of *solution*.

▶ *Remark: notational conventions*
We use several convenient notations throughout the paper to lighten the presentation of constraints. First of all we do not make a difference between sets and conjunctions of constraints: for instance we may write $\mathsf{E}^1 = \bigwedge_{i=1}^{n} \varphi_i$ instead of $\mathsf{E}^1 = \{\varphi_i\}_{i=1}^{n}$ and conversely. We also interpret a substitution σ as the set of equations $\mathcal{E} = \{x =^? x\sigma \mid x \in dom(\sigma)\}$. ◀

1.2 (Most general) unifiers

We now recall some basics on term unification, a key concept in symbolic models that has some specificities in our context, in particular regarding second-order terms.

Unification of first-order terms Two first-order terms u and v are *unifiable* if there exists a substitution σ, called a *unifier*, such that $u\sigma = v\sigma$. For example the terms $u = \langle \mathsf{sdec}(x, y), z\rangle$ and $v = \langle z_1, z_2\rangle$ are unified by $\sigma = \{z_1 \mapsto \mathsf{sdec}(x, y), z_2 \mapsto z\}$. The terms u and z' are unifiable as well using $\sigma = \{z' \mapsto u\}$, but the terms u and z are not. More generally a unifier σ of a set of equations $\mathcal{E} = \{u_i =^? v_i\}_{i=1}^{n}$ is a unifier of u_i and v_i for all i. A classical characterisation of the set of unifiers of two terms is based on *most general unifiers*:

Definition 3.5 (most general unifier)

A substitution σ is called a *most general unifier* of $\mathcal{E}$ if for any θ unifier of $\mathcal{E}$, there exists τ such that $\theta = \sigma\tau$. In this case we write $\sigma = mgu(\mathcal{E})$. On the contrary we write $mgu(\mathcal{E}) = \bot$ when no such substitutions exist.

A straightforward inductive procedure allows to decide whether $\mathcal{E}$ is unifiable and, if it is, to compute $mgu(\mathcal{E})$ We assume that this computation does not introduce variables, that is, if $\sigma = mgu(\mathcal{E})$ then $dom(\sigma) \cup vars(img(\sigma)) \subseteq vars(\mathcal{E})$. We also require that $dom(\sigma) \cap vars(img(\sigma)) = \varnothing$, that is, applying a mgu twice has not more effect than applying it once. Note as well that all unifiers are instances of the mgu but the converse is also true, that is, all instances of a mgu are unifiers. By convenience we also write $mgu(\mathcal{E})$ in the case where $\mathcal{E}$ contains disequations (typically when writing $mgu(\mathsf{E}^1(\mathcal{C}))$): in this case only equations are taken into account and nothing ensures that the mgu satisfies the disequations of $\mathcal{E}$.

However mgu's are only syntactic: when taking the theory E into account we say that σ is a *unifier modulo theory* of $\mathcal{E}$ when for all $(u =^? v) \in \mathcal{E}$, $u\sigma =_E v\sigma$. A standard procedure based on narrowing (not detailed here) allows to compute *most general unifiers modulo* E when E is subterm convergent among others. However unlike the syntactic case they are not unique in general:

Definition 3.6 (most general unifier modulo theory)

We let $\mathcal{E}$ be a set of equations and E be a convergent theory. A set of *most general unifiers modulo* E is a set of substitutions $mgu_E(\mathcal{E})$ that verifies the following properties:

1. for all $\sigma \in mgu_E(\mathcal{E})$, σ is a unifier of $\mathcal{E}$ modulo E
2. for all θ unifier of $\mathcal{E}$ modulo E, there exists $\sigma \in mgu_E(\mathcal{E})$ and a substitution τ such that for all $x \in vars(\mathcal{E})$, $x\theta =_E x\sigma\tau$

Again we emphasise that equality modulo E only operates on valid messages, that is, if $\sigma \in mgu_E(u =^? v)$ then $u\sigma$ and $v\sigma$ verify the msg predicate. A typical usecase we consider in the symbolic semantics is $mgu_E(u =^? u)$, which is the most general substitution σ such that $\mathsf{msg}(u\sigma)$ holds (if any). For example if $u = \mathsf{adec}(x, y)$ we have

$$mgu_E(u =^? u) = \{\sigma\} \qquad \text{where } \sigma = \{x \mapsto \mathsf{aenc}(x', \mathsf{pk}(y')), y \mapsto \mathsf{pk}(y')\}$$

This example also highlights that, unlike the syntactic case, computing mgu's modulo theory may require to introduce new variables. However it is still possible to enforce that $dom(\sigma) \cap vars(img(\sigma)) = \varnothing$.

Unification of second-order terms Intuitively the unification of two second-order terms ξ and ζ modulo theory means that they deduce the same first-order term u w.r.t. a given frame Φ. This unusual kind of unification is performed as a part of our constraint solving algorithm using a dedicated kind of constraints $\xi =^?_f \zeta$; we refer to Chapter 4 for more details. We therefore do not need to rely on an external, black-box unification algorithm. The situation is however different regarding syntactic mgu's and we give details below.

The definition is the same as usual, meaning that a syntactic unifier of ξ and ζ is a second-order substitution Σ such that $\xi\Sigma = \zeta\Sigma$. However additional care is required in our context due to the variable types. Indeed we recall that by definition a second-order substitution has

to respect types, that is, a variable $X{:}n$ cannot be mapped to a term containing axioms ax_i or variables $Y{:}i$ if $i > n$. Say for instance we want to unify the two second-order terms $X{:}1$ and $\mathsf{f}(Y{:}2)$: a syntactic computation of the mgu would give the substitution $\Sigma = \{X \mapsto \mathsf{f}(Y)\}$, which does not respect the type of X. In this case one solution is to introduce a fresh variable $Z{:}1$ and to choose the following unifier:

$$mgu(X =^? \mathsf{f}(Y)) = \{X \mapsto \mathsf{f}(Z), Y \mapsto Z\} = \Sigma\{Y \mapsto Z\}\,.$$

Formally given a second-order term ξ, let us write $\#(\xi)$ the maximal type appearing in ξ, that is, the integer $\#(\xi) = \min\{i \in \mathbb{N} \mid \xi \in \mathcal{T}_i^2\}$. The mgu of a conjunction of equations φ is then computed inductively as follows:

$$mgu(\top) = \top$$

$$mgu\left(\varphi \wedge \mathsf{f}(\xi_1, \ldots, \xi_n) =^? \mathsf{g}(\zeta_1, \ldots, \zeta_n)\right) = \begin{cases} \bot & \text{if } \mathsf{f} \neq \mathsf{g} \\ mgu\left(\varphi \wedge \bigwedge_{i=1}^n \xi_i =^? \zeta_i\right) & \text{if } \mathsf{f} = \mathsf{g} \end{cases}$$

$$mgu\left(\varphi \wedge X{:}i =^? \xi\right) = \begin{cases} \bot & \text{if } X \in vars^2(\xi) \text{ and } \xi \neq X \\ \bot & \text{else if } \exists \mathsf{ax}_j \in axioms(\xi), j > i \\ \Sigma_0\Sigma & \text{else if } \xi \notin \mathcal{X}^2,\ Y{:}j \in vars^2(\xi),\ j > i,\ Z{:}i \text{ fresh and} \\ & \text{with } \Sigma_0 = \{Y \mapsto Z\} \text{ and } \Sigma = mgu\left(\varphi\Sigma_0 \wedge X{:}i =^? \xi\Sigma_0\right) \\ \Sigma_0\Sigma & \text{else if } \#(\xi) \leqslant i, \text{ with } \Sigma_0 = \{X \mapsto \xi\} \text{ and } \Sigma = mgu\left(\varphi\Sigma_0\right) \end{cases}$$

As before we extend this notation to arbitrary sets $\mathcal{E}$, that is, we may write $mgu(\mathcal{E})$ even if $\mathcal{E}$ contains disequations (which are then ignored during the computation). The correctness of this function is proved below.

Proposition 3.1 (correctness of second-order mgu's)
For all sets of second-order equations $\mathcal{E}$, the computation of $mgu(\mathcal{E})$ terminates. Besides we have that $mgu(\mathcal{E}) = \bot$ *iff* there exist no unifiers of $\mathcal{E}$. Otherwise:
1 $mgu(\mathcal{E})$ is a second-order substitution, i.e., respects the types
2 for all unifiers Σ of $\mathcal{E}$, there exists a substitution Σ_0 such that $\Sigma = mgu(\mathcal{E})\Sigma_0$.

Proof. We only prove the termination since all other properties can be proved separately by straightforward inductions on the definition of mgu. We let the partial ordering on second order variables $\preccurlyeq$ given by the types, i.e. $X{:}i \preccurlyeq Y{:}j$ *iff* $i \leqslant j$. Given a set of second-order equations $\mathcal{E}$ we then let

$$\mu(\mathcal{E}) = (vars^2(\mathcal{E}), M(\mathcal{E}), F(\mathcal{E}))$$

where $M(\mathcal{E})$ is the multiset of variables of $\mathcal{E}$, i.e. multiplicity included, and $F(\mathcal{E})$ is the multiset of the sizes of the equations of $\mathcal{E}$ (where the size of $\xi =^? \zeta$ is the number of function symbols in ξ and ζ). The first two components are ordered w.r.t. the multiset extension of $\preccurlyeq$, and the third one w.r.t. the multiset extension of $\leqslant$. The overall tuple is ordered w.r.t. the lexicographic composition of the three components.

If we number from 1 to 7 the axioms defining mgu, we can show that μ decreases at each recursive call: (1), (2), (4) and (5) make no recursive calls; (3) preserves $vars^2$ and M, and

makes F decrease; (6) replaces all occurrences of Y with Z than has a lower type which makes $vars^2$ decrease. Regarding (7) two cases can arise: either $\xi = X$ or $X \notin vars^2(\xi)$. In the first case $vars^2$ is nonincreasing and M is decreasing since two occurrences of X are removed and the rest of the formula $\mathcal{E}$ is left unchanged. In the second case $vars^2$ is decreasing since all occurrences of X are removed and no variables are added. □

1.3 (Most general) solutions

Solutions Let us now formalise the semantics of constaints. Given a constraint φ, a frame Φ and second- and first-order substitutions Σ and σ we define the predicate $(\Phi, \Sigma, \sigma) \models \varphi$ by:

$$\begin{array}{lll}
(\Phi,\Sigma,\sigma) \models \xi \vdash^? u & \textit{iff} & \xi\Sigma\Phi\sigma =_E u\sigma \\
(\Phi,\Sigma,\sigma) \models \xi =^? \zeta & \textit{iff} & \xi\Sigma = \zeta\Sigma \\
(\Phi,\Sigma,\sigma) \models u =^? v & \textit{iff} & u\sigma = v\sigma \\
(\Phi,\Sigma,\sigma) \models \forall x.\,\varphi & \textit{iff} & \text{for all first-order terms } t, (\Phi,\Sigma,\sigma) \models \varphi\{x \mapsto t\} \\
(\Phi,\Sigma,\sigma) \models \forall X{:}n.\,\varphi & \textit{iff} & \text{for all } \xi \in \mathcal{T}^2_n, (\Phi,\Sigma,\sigma) \models \varphi\{X \mapsto \xi\}
\end{array}$$

The definition is extended the logical constructors $\neg, \wedge, \vee, \ldots$ in the natural way. By convention we may only write $(\Phi, \Sigma, \sigma) \models \varphi$ when $x\sigma$ and $X\Sigma\Phi\sigma$ are ground for all $x \in vars^1(\varphi)$ and $X \in vars^2(\varphi)$. Intuitively the second-order substitution Σ describes which recipes are used to deduce each input term appearing in φ and σ gives their actual values.

Definition 3.7 (solution of a constraint system)
We say that (Σ, σ) is a *solution* of $\mathcal{C}$ if $dom(\Sigma) = vars^2(\mathcal{C})$, $dom(\sigma) = vars^1(\mathcal{C})$ and $(\Phi(\mathcal{C}), \Sigma, \sigma) \models \mathsf{D}(\mathcal{C}) \wedge \mathsf{E}^1(\mathcal{C})$. We call Σ a *second-order solution* of $\mathcal{C}$ and σ its *first-order solution*. The set of solutions of $\mathcal{C}$ is written $Sol(\mathcal{C})$.

The solutions of a constraint system $\mathcal{C}$ indicate how the inputs of $\mathcal{C}$ (i.e., $vars^1(\mathsf{D}(\mathcal{C}))$) can be computed while satisfying the constraints imposed by $\mathsf{E}^1(\mathcal{C})$. Due to the origination property, the values the first-order solution σ takes on $vars^1(\mathsf{D}(\mathcal{C}))$ is uniquely determined by which recipes are used to deduce terms, i.e., by the second-order solution Σ.

▶ *Example 3.1*
Consider again the simple example $P = \mathsf{new}\ k.\,\overline{c}\langle k\rangle.\,c(x).\,\mathsf{if}\ \mathsf{fst}(x) = k\ \mathsf{then}\ \overline{c}\langle \mathsf{h}(x)\rangle$. The traces performing the final output $\mathsf{h}(x)$ are characterised by the constraint system

$$\Phi(\mathcal{C}) = \{\mathsf{ax}_1 \mapsto k, \mathsf{ax}_2 \mapsto \mathsf{h}(x)\} \qquad \mathsf{D}(\mathcal{C}) = \{X \vdash^? x\} \qquad \mathsf{E}^1(\mathcal{C}) = \{x =^? \langle k, y\rangle\}$$

where $X{:}1$ and y are fresh second- and first-order variables, respectively. Observe in particular that the informal constraint "*there exists a term y such that $x = \langle k, y\rangle$*" is not formalised using an explicit $\exists$ quantification but with a free variable y. All second-order solutions of $\mathcal{C}$ are instances of $\Sigma_0 = \{X \mapsto \langle \mathsf{ax}_1, Y\rangle\}$ where $Y{:}1$ is fresh, for example, $\Sigma = \{X \mapsto \langle \mathsf{ax}_1, a\rangle\}$ with $a \in \mathcal{F}_0$. The corresponding first-order solution is then $\sigma = \{x \mapsto \langle k, a\rangle, y \mapsto a\}$. ◀

Most general solutions Similarly to mgu's we now give a novel characterisation of solutions as instances of so-called most general solutions (*mgs*). The definition is parametrised with a predicate π on second-order substitutions, writing $Sol^\pi(\mathcal{C}) = \{(\Sigma, \sigma) \in Sol(\mathcal{C}) \mid \pi(\mathcal{C}) \text{ holds}\}$. This will permit later to consider solutions that have a certain form.

Definition 3.8 (most general solution)

A set of *most general solutions of* $\mathcal{C}$ *that satisfy* π is a set $mgs^{\pi}(\mathcal{C})$ of second-order substitutions such that:

1. for all $\Sigma_0 \in mgs^{\pi}(\mathcal{C})$, $dom(\Sigma_0) \subseteq vars^2(\mathcal{C})$, for all injections Σ_1 to fresh constants and of domain $dom(\Sigma_1) = vars^2(img(\Sigma_0), \mathcal{C}) \setminus dom(\Sigma_0)$, $(\Sigma_0\Sigma_1, \sigma) \in Sol^{\pi}(\mathcal{C})$ for some σ.
2. for all $(\Sigma, \sigma) \in Sol^{\pi}(\mathcal{C})$, there exists $\Sigma_0 \in mgs^{\pi}(\mathcal{C})$ and Σ_1 such that $\Sigma = \Sigma_0\Sigma_1$.

We omit the predicate π in the case where $\pi = \top$, i.e., $\pi(\Sigma)$ holds for any substitution.

The first condition of the definition states that a mgs Σ_0 is "almost" a solution of $\mathcal{C}$: it may not be one because Σ_0 may have a domain strictly included in $vars^2(\mathcal{C})$ and may not have a ground image; but we obtain a solution by replacing all pending variables by fresh names. The second condition then states that all solutions are instances of a mgs.

▶ *Example 3.2*

In Example 3.1 we have $mgs(\mathcal{C}) = \{\Sigma_0\}$ and $Sol(\mathcal{C})$ is the set of all ground instances of Σ_0. However in general the situation may be less ideal. For example a constraint system may have several most general solutions; a simple example being, with $\mathsf{h}/1$ and $k \in \mathcal{N}$:

$$\Phi(\mathcal{C}) = \{\mathsf{ax}_1 \mapsto \mathsf{h}(k), \mathsf{ax}_2 \mapsto k\} \qquad \mathsf{D}(\mathcal{C}) = \{X{:}2 \vdash^? x\} \qquad \mathsf{E}^1(\mathcal{C}) = \{x =^? \mathsf{h}(k)\}$$

The constraint system $\mathcal{C}$ expresses that an input x should be instantiated by $\mathsf{h}(k)$, potentially by using previous two outputs $\mathsf{h}(k)$ and k. There are therefore two ways of computing x: either using ax_1 or $\mathsf{h}(\mathsf{ax}_2)$, which is reflected as the fact that $mgs(\mathcal{C}) = \{\Sigma_1, \Sigma_2\}$ with

$$\Sigma_1 = \{X \mapsto \mathsf{h}(\mathsf{ax}_2)\} \qquad \Sigma_2 = \{X \mapsto \mathsf{ax}_1\}$$

Still, it is possible to obtain unique mgs' by performing a case analysis and restricting the solutions accordingly; typically here we have $mgs^{\pi_i}(\mathcal{C}) = \{\Sigma_i\}$ with

$$\pi_1(\Sigma) \stackrel{def}{=} \exists X'.\, X =^? \mathsf{h}(X') \qquad \pi_2(\Sigma) \stackrel{def}{=} \forall X'.\, X \neq^? \mathsf{h}(X')$$

Another notable point is that some ground instances of a mgs may not be solutions themselves. The simplest example is, with $a \in \mathcal{F}_0$, $\mathcal{C} = (\varnothing, X \vdash^? x, x \neq^? a)$: we have $mgs(\mathcal{C}) = \{id\}$ but the substitution $\{X \mapsto a\}$ is a ground instance of the identity but not a solution (which does not contradict Item 1 of Definition 3.8 since although a is a constant, it is not fresh). ◀

We explain in Chapter 4, Section 2 how to generate a finite set of most general solutions, at least in the context of our decision procedures.

1.4 Symbolic semantics

Symbolic execution We now present how to replace attacker inputs by constraints in practice. This takes the form of a symbolic semantics that recalls the usual semantics of the applied pi calculus (the "concrete semantics") except that a constraint system is carried over the execution to collect the trace constraints. The semantics operates on so-called *symbolic processes* $(\mathcal{P}, \mathcal{C})$ where $\mathcal{P}$ is a multiset of (non-necessarily ground) plain processes and $\mathcal{C}$ is a constraint system. All free variables of $\mathcal{P}$ are bound by deductions facts, that is, for all $x \in vars(\mathcal{P})$ there exists $(X \vdash^? x) \in \mathsf{D}(\mathcal{C})$. The semantics then takes the form of a labelled transition semantics $\xrightarrow{\alpha}_{\mathsf{s}}$ between symbolic processes, defined in Figure 3.1, where α ranges over the following alphabet of *symbolic actions*:

1. *symbolic input actions* $X(Y)$ where X and Y are second-order variables, modelling public inputs as in the concrete semantics except that the attacker recipes are replaced by the two placeholders X, Y;
2. *symbolic output actions* $\overline{X}\langle \mathsf{ax}_i \rangle$ that follow the same logic;
3. the *unobservable action* τ which has the exact same role as in the concrete semantics.

Before we define the semantics let us explain how we handle conditionals. First of all we recall our convention to interpret substitutions as sets of equalities, that is, the positive branch of "if $u = v$ then P else Q" will add one mgu of u and v modulo theory to E^1. Regarding the negative branch, we want to add a constraint that is satisfied *iff* u and v are not equal modulo theory. We write it $\neg mgu_E(u =^? v)$ and define it as follows:

$$\neg mgu_E(u =^? v) = \bigwedge_{\sigma \in mgu_E(u =^? v)} \forall \tilde{z_\sigma} \bigvee_{x \in vars(u,v)} x \neq^? x\sigma$$

where $\tilde{z_\sigma} = vars(u\sigma, v\sigma) \setminus vars(u, v)$.

If $\mathcal{C} = (\Phi, \mathsf{D}, \mathsf{E}^1)$, $\mu = mgu(\mathsf{E}^1) \neq \bot$ and $n = |dom(\Phi)|$:

$$(\{\!\{\mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q\}\!\} \cup \mathcal{P}, \mathcal{C}) \xrightarrow{\tau}_{\mathsf{s}} (\{\!\{P\}\!\} \cup \mathcal{P}, (\Phi, \mathsf{D}, \mathsf{E}^1 \wedge \sigma)) \quad \text{(S-Then)}$$
$$\text{if } \sigma \in mgu_E(u\mu =^? v\mu)$$

$$(\{\!\{\mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q\}\!\} \cup \mathcal{P}, \mathcal{C}) \xrightarrow{\tau}_{\mathsf{s}} (\{\!\{Q\}\!\} \cup \mathcal{P}, (\Phi, \mathsf{D}, \mathsf{E}^1 \wedge \neg mgu_E(u\mu =^? v\mu))) \quad \text{(S-Else)}$$

$$(\{\!\{u(x).P\}\!\} \cup \mathcal{P}, \mathcal{C}) \xrightarrow{Y(X)}_{\mathsf{s}} (\{\!\{P\}\!\} \cup \mathcal{P}, (\Phi, \mathsf{D} \wedge X \vdash^? x \wedge Y \vdash^? y, \mathsf{E}^1 \wedge \sigma)) \quad \text{(S-In)}$$
$$\text{if } Y{:}n,\ X{:}n \text{ and } y \text{ are fresh and } \sigma \in mgu_E(y =^? u\mu)$$

$$(\{\!\{\overline{u}\langle v\rangle.P\}\!\} \cup \mathcal{P}, \mathcal{C}) \xrightarrow{\overline{Y}\langle \mathsf{ax}_{n+1}\rangle}_{\mathsf{s}} (\{\!\{P\}\!\} \cup \mathcal{P}, (\Phi \cup \{\mathsf{ax}_{n+1} \mapsto v\sigma\downarrow\}, \mathsf{D} \wedge Y \vdash^? y, \mathsf{E}^1 \wedge \sigma)) \quad \text{(S-Out)}$$
$$\text{if } Y{:}n \text{ and } y \text{ are fresh and } \sigma \in mgu_E(y =^? u\mu \wedge v\mu =^? v\mu)$$

$$(\{\!\{\overline{u}\langle v\rangle.P, w(x).Q\}\!\} \cup \mathcal{P}, \mathcal{C}) \xrightarrow{\tau}_{\mathsf{s}} (\{\!\{P, Q\{x \mapsto v\sigma\}\}\!\} \cup \mathcal{P}, (\Phi, \mathsf{D} \wedge \varphi, \mathsf{E}^1 \wedge \sigma)) \quad \text{(S-Comm)}$$
$$\text{if for } Y{:}n \text{ and } y \text{ fresh, } \varphi = \forall Y.\, Y \not\vdash^? y \text{ and } \sigma \in mgu_E(u\mu =^? w\mu \wedge v\mu =^? v\mu \wedge y =^? u\mu)$$

$$(\{\!\{\mathsf{new}\ k.P\}\!\} \cup \mathcal{P}, \mathcal{C}) \xrightarrow{\tau}_{\mathsf{s}} (\{\!\{P\{k \mapsto k'\}\}\!\} \cup \mathcal{P}, \mathcal{C}) \qquad \text{if } k' \in \mathcal{N} \setminus names(P, \mathcal{P}, \mathcal{C}) \quad \text{(S-New)}$$

$$(\{\!\{P \mid Q\}\!\} \cup \mathcal{P}, \mathcal{C}) \xrightarrow{\tau}_{\mathsf{s}} (\{\!\{P, Q\}\!\} \cup \mathcal{P}, \mathcal{C}) \quad \text{(S-Par)}$$

$$(\{\!\{!\,P\}\!\} \cup \mathcal{P}, \mathcal{C}) \xrightarrow{\tau}_{\mathsf{s}} (\{\!\{!\,P, P\}\!\} \cup \mathcal{P}, \mathcal{C}) \quad \text{(S-Repl)}$$

Figure 3.1 A symbolic semantics for the applied pi-calculus

The rule (S-In) adds two deduction facts $X \vdash^? x$ and $Y \vdash^? y$ to D, modelling that the input term and communication channel should be deducible by the adversary; in particular the constraint $\sigma \in mgu_E(y =^? u\mu)$ indicates that the term deduced by Y is effectively the channel u. The rule (S-Out) essentially follows the same logic, adding a fresh deduction fact and a constraint indicating that the channel is deducible. We assume an implicit alpha renaming of bound variables so that each appear only once in the process: this prevents reference conflicts in D when applying the rule (S-In). Finally let us comment on the rule (S-Comm). The constraint φ added to D indicates that the channel u (bound to y using the equation $y =^? u\mu$)

used to perform the communication should not be deducible by the adversary. Removing the constraint leads to a symbolic semantics for the classical semantics which, we recall, is the semantics where internal communications can be performed on any channel. Then the constraint σ added to E^1 simply indicates that the two communication channels u and w should be identical.

Similarly to the concrete semantics we call a *symbolic trace* a sequence of transitions $(\mathcal{P}_0, \mathcal{C}_0) \xrightarrow{\alpha_1}_{\mathsf{s}} \cdots \xrightarrow{\alpha_n}_{\mathsf{s}} (\mathcal{P}_n, \mathcal{C}_n)$, which may be referred to as $(\mathcal{P}_0, \mathcal{C}_0) \overset{\mathsf{tr}}{\Rightarrow}_{\mathsf{s}} (\mathcal{P}_n, \mathcal{C}_n)$ if tr is obtained by removing the τ's from the word $\alpha_1 \cdots \alpha_n$. For simplicity the plain process P may be interpreted as the symbolic process $(\{\!\{P\}\!\}, (\varnothing, \top, \top))$.

▶ *Example 3.3*

We consider again the example of the private authentication protocol. We recall the process of the agent B receiving the communication, writing pk_X, sk_X instead of $\mathsf{pk}(\mathsf{sk}(X)), \mathsf{sk}(X)$:

$$\begin{aligned} B = d(x).\,&\mathsf{new}\ r.\\ &\mathsf{let}\ \langle n, p\rangle = \mathsf{radec}(x, sk_B)\ \mathsf{in}\\ &\mathsf{if}\ p = pk_A\ \mathsf{then}\ \overline{d}\langle \mathsf{raenc}(\langle n, N_B, pk_B\rangle, r, p)\rangle\\ &\mathsf{else}\ \overline{d}\langle \mathsf{raenc}(n, r, pk_B)\rangle \end{aligned}$$

and use a frame $\Phi_0 = \{\mathsf{ax}_1 \mapsto pk_A, \mathsf{ax}_2 \mapsto pk_B, \mathsf{ax}_3 \mapsto \mathsf{raenc}(\langle N_A, pk_A\rangle, r', pk_B)\}$, containing public keys and the connection request sent by A. We give in Figure 3.2 a tree of all symbolic executions of B (we only write the constraints added at each step). We execute let's using a rule (S-Let), not formalised, executing symbolically their encodings into standard conditionals.

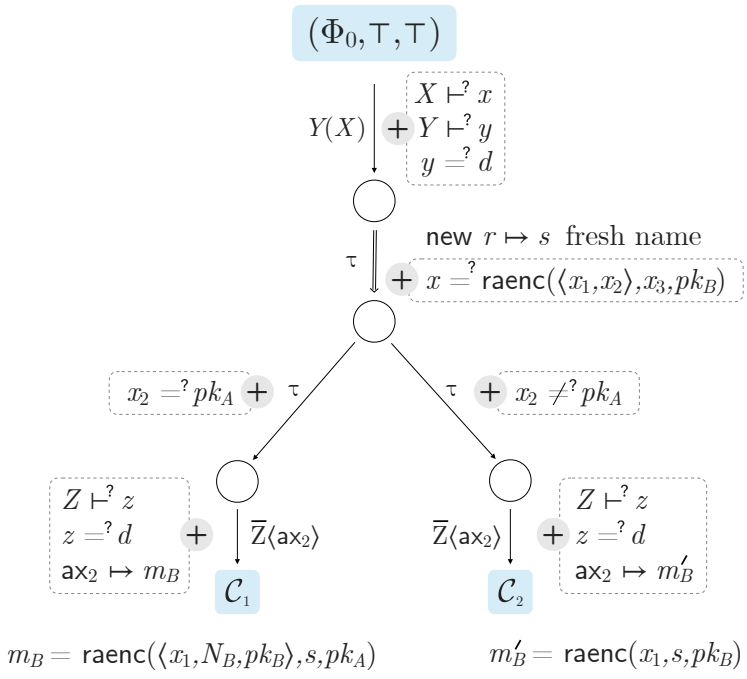


Figure 3.2 Tree of all constraint systems reachable by executing B symbolically

Intuitively, the branch of the constraint system $\mathcal{C}_1$ abstracts the set of concrete traces where B accepts the connection, and the branch of $\mathcal{C}_2$ those where B refuses it. Typically in the traces of the branch $\mathcal{C}_1$ the attacker forwards the message of A or forges one pretending to be A; this is formally expressed by the fact that $mgs(\mathcal{C}_1) = \{\Sigma_0 \cup \Sigma_{\mathsf{fwd}},\ \Sigma_0 \cup \Sigma_{\mathsf{att}}\}$ where:

$$\Sigma_0 = \{Y \mapsto d, Z \mapsto d\} \qquad \Sigma_{\mathsf{fwd}} = \{X \mapsto \mathsf{ax}_3\} \qquad \Sigma_{\mathsf{att}} = \{X \mapsto \mathsf{raenc}(\langle x_1, \mathsf{ax}_1\rangle, x_3, \mathsf{ax}_2)\} \blacktriangleleft$$

Soundness and completeness Similar symbolic semantics have been developed in the context of protocol analysis [Bau07, CCD13]. The general approach is to abstract the (infinite) set of concrete traces by the finite set of symbolic traces and to study the solutions of the resulting constraint systems. A typical example is that the following statements are equivalent:

1. *Weak secrecy of the term u in P*: for all traces $P \overset{\mathsf{tr}}{\Rightarrow} (\mathcal{P}, \Phi)$, u is not deducible from Φ
2. for all symbolic traces $P \overset{\mathsf{tr}}{\Rightarrow}_{\mathsf{s}} (\mathcal{P}, \mathcal{C})$, the system $(\Phi(\mathcal{C}), \mathsf{D}(\mathcal{C}) \wedge X \vdash^? x, \mathsf{E}^1(\mathcal{C}) \wedge x =^? u)$ has no solution, where $X{:}n$ and x are fresh, $n = |dom(\Phi)|$

This reduces weak secrecy (for a bounded number of sessions) to the decidability of whether a constraint system has a solution. Similar approaches have been developed in [Bau07, CCD13] to decide some equivalence properties for some classes of processes (we comment more on these related works in Chapter 7). This usually relies on a connection between the symbolic and the concrete semantics, under the form of two properties: 1 *soundness*: applying to a symbolic trace a solution of its final constraint system leads to a concrete trace; and 2 *completeness*: all concrete traces are obtained by applying a solution to a symbolic one. They are formalised below, the proof following from a straightforward induction on the length of the traces.

Proposition 3.2 (soundness and completeness of the symbolic semantics)
Let $(\mathcal{P}, \mathcal{C})$ be a symbolic process.

1. *Soundness:* for all symbolic traces $(\mathcal{P}, \mathcal{C}) \overset{\mathsf{tr}_s}{\Rightarrow}_{\mathsf{s}} (\mathcal{Q}, \mathcal{C}')$ and $(\Sigma, \sigma) \in Sol(\mathcal{C}')$, there exists a concrete trace of the form $(\mathcal{P}\sigma, \Phi(\mathcal{C})\sigma\!\downarrow) \overset{\mathsf{tr}_s\Sigma}{\Longrightarrow} (\mathcal{Q}\sigma, \Phi(\mathcal{C}')\sigma\!\downarrow)$
2. *Completeness:* for all symbolic processes $(\mathcal{P}, \mathcal{C})$, $(\Sigma, \sigma) \in Sol(\mathcal{C})$ and concrete traces $(\mathcal{P}\sigma, \Phi(\mathcal{C})\sigma\!\downarrow) \overset{\mathsf{tr}}{\Rightarrow} (\mathcal{Q}, \Phi)$, there exists a symbolic trace $(\mathcal{P}, \mathcal{C}) \overset{\mathsf{tr}'}{\Rightarrow}_{\mathsf{s}} (\mathcal{Q}', \mathcal{C}')$ and $(\Sigma', \sigma') \in Sol(\mathcal{C}')$ such that $\Sigma \subseteq \Sigma'$, $\mathcal{Q} = \mathcal{Q}'\sigma'$, $\mathsf{tr} = \mathsf{tr}'\Sigma'$ and $\Phi = \Phi(\mathcal{C}')\sigma'\!\downarrow$.

2 Decision procedures for equivalences: the partition tree

Section summary

We define the novel notion of *partition tree*, the core notion we use to decide trace equivalence and labelled bisimilarity of bounded processes. This is intuitively a *tree of symbolic executions* of two processes whose structure allows to easily conclude whether the processes are equivalent.

2.1 Definition of the partition tree

To decide trace equivalence and labelled bisimilarity, we introduce the novel notion of *partition tree* of two bounded processes P and Q. The point is to build a (finite) tree of all symbolic executions of P and Q, grouping into the same nodes intermediary processes as follows:

1. All processes of a same node should have a *common, unique mgs*. Since one symbolic process alone may already have several most general solutions, the node is parametrised by a restricting predicate π on second-order solutions (recall Example 3.2).
2. When applying the mgs of a node to all of the processes it contains, the resulting frames are *statically equivalent*; conversely all reachable symbolic processes that would verify this property should be in the node as well.

A branch of this tree therefore represents the set of all equivalent traces of P and Q taking a given sequence of visible actions. Taking profit of this observation we will show that whenever P and Q are not trace equivalent or labelled bisimilar, a witness of non-equivalence can be exhibited using the tree. Formally its nodes are modelled by *configurations* that consist of sets Γ of symbolic processes sharing a unique mgs and statically equivalent solutions.

Definition 3.9 (configuration)

A *configuration* is a pair (Γ, π) where Γ is a set of symbolic processes and π a predicate on second-order substitutions. We also require that:

1. the predicate π is defined on $vars^2(\Gamma)$, that is, for all Σ, $\pi(\Sigma)$ *iff* $\pi(\Sigma_{|vars^2(\Gamma)})$;
2. for all $(\mathcal{P}, \mathcal{C}) \in \Gamma$, $|mgs^\pi(\mathcal{C})| = 1$;
3. for all $(\mathcal{P}_1, \mathcal{C}_1), (\mathcal{P}_2, \mathcal{C}_2) \in \Gamma$, if $(\Sigma, \sigma_1) \in Sol^\pi(\mathcal{C}_1)$ then there exists σ_2 such that $(\Sigma, \sigma_2) \in Sol^\pi(\mathcal{C})$ and $\Phi(\mathcal{C}_1)\sigma_1 \sim \Phi(\mathcal{C}_2)\sigma_2$.

The predicate π can typically be described using second-order (dis)equations. We then consider trees with nodes labelled by configurations and edges by visible symbolic actions (i.e., not τ). Given a node n of such a tree, we write $\Gamma(n)$ and $\pi(n)$ the components of the corresponding configuration, and $n \xrightarrow{\alpha} n'$ to express that n' is a child node of n through an edge labelled by the symbolic action α. By definition of a mgs, the points 2 and 3 of Definition 3.9 above ensure that all symbolic processes in $\Gamma(n)$ have the same set of second-order variables, written $vars^2(n)$, and a common and unique mgs, written $mgs(n)$.

Definition 3.10 (partition tree)

A *partition tree* of two bounded processes P and Q is a tree T whose nodes are labelled by configurations and edges by visible symbolic actions, and that verifies the following properties. First of all $P, Q \in \Gamma(root(T))$ and $\pi(root(T)) = \top$, where $root(T)$ denotes the root node of the tree. Then for all nodes n of T, $(\mathcal{P}, \mathcal{C}) \in \Gamma(n)$ and visible symbolic actions α:

1. *Closure by τ-transition*: if $(\mathcal{P}, \mathcal{C}) \overset{\tau}{\Rightarrow}_{\mathsf{s}} (\mathcal{P}', \mathcal{C}')$ and $Sol^{\pi(n)}(\mathcal{C}') \neq \varnothing$ then $(\mathcal{P}', \mathcal{C}') \in \Gamma(n)$.
2. *All symbolic transitions are reflected in the tree:* if $(\mathcal{P}, \mathcal{C}) \overset{\alpha}{\Rightarrow}_{\mathsf{s}} (\mathcal{P}', \mathcal{C}')$ and $(\Sigma, \sigma) \in Sol^{\pi(n)}(\mathcal{C}')$ then there exists an edge $n \xrightarrow{\alpha} n'$ in T such that $(\mathcal{P}', \mathcal{C}') \in \Gamma(n')$ and $(\Sigma', \sigma) \in Sol^{\pi(n')}(\mathcal{C}')$ for some Σ' that coincides with Σ on $vars^2(n)$.

Moreover for all edges $n \xrightarrow{\alpha} n_c$ of T and $(\mathcal{P}_c, \mathcal{C}_c) \in \Gamma(n_c)$:

3. *Predicates are refined along branches:* for all Σ, if Σ verifies $\pi(n_c)$ then it verifies $\pi(n)$.
4. *Nodes are maximal:* if $(\Sigma, \sigma) \in Sol^{\pi(n)}(\mathcal{C})$, $(\Sigma_c, \sigma_c) \in Sol^{\pi(n_c)}(\mathcal{C}_c)$ and $\Sigma \subseteq \Sigma_c$, then $\Gamma(n_c)$ contains all symbolic processes $(\mathcal{P}', \mathcal{C}')$ such that $(\mathcal{P}, \mathcal{C}) \overset{\alpha}{\Rightarrow}_{\mathsf{s}} (\mathcal{P}', \mathcal{C}')$ and, for some substitution σ', $(\Sigma_c, \sigma') \in Sol(\mathcal{C}')$ and $\Phi(\mathcal{C}_c)\sigma_c \sim \Phi(\mathcal{C}')\sigma'$.

The set of partition trees of P and Q is written $\mathsf{PTree}(P, Q)$.

The set $\mathsf{PTree}(P, Q)$ is infinite (at least because arbitrarily many processes can be put in the root configuration) but our decision procedures only require to construct one, arbitrary partition tree. The children n' of a node n represent the sets of processes, grouped w.r.t. static equivalence, reachable by one transition from a process of n. The Item 2 ensures that all cases are covered, that is, for all symbolic transitions from n and all solutions Σ, at least one child n' should contain the resulting symbolic process. Note that we do not impose that Σ verifies $\pi(n')$, but that there exists another solution Σ' computing the same first-order terms that does. This more permissive approach will allow us, when generating partition-tree nodes in Chapter 4, to use families of predicates π that only consider solutions of a certain form (which therefore requires to prove that any deducible term can be computed by a recipe of this form). Item 4 then formalises that the nodes are saturated under static equivalence: if n' is a child of n and a symbolic transition $A \overset{\alpha}{\Rightarrow}_{\mathsf{s}} B$ from a process $A \in \Gamma(n)$ may result into a process statically equivalent to a process $C \in \Gamma(n')$ then B should be in $\Gamma(n')$ as well.

▶ *Example 3.4*

Let us draw a partition tree corresponding to an anonymity analysis in the private authentication protocol, simplified for the sake of readability. We consider the following light version of the role of the process B accepting a connection from an agent X, removing the identification nonces N_A, N_B from the protocol and replacing the decoy message by a fresh nonce r:

$$B_X = d(x).\ \mathsf{new}\ r.\ \mathsf{if}\ \mathsf{radec}(x, sk_B) = pk_X\ \mathsf{then}\ \overline{d}\langle \mathsf{raenc}(\mathsf{ok}, r, pk_X)\rangle\ \mathsf{else}\ \overline{d}\langle r\rangle$$

We consider a 3-agent scenario (A, B, C) where A has already emitted $\mathsf{raenc}(pk_A, r_A, pk_B)$ to initiate a communication with B and the security property we study is whether the identity of B's accepted recipient remains anonymous. That is we want to prove $P \approx Q$ where

$$P = C[B_A] \qquad Q = C[B_C] \qquad C[R] = \overline{c}\langle pk_A\rangle.\ \overline{c}\langle pk_B\rangle.\ \overline{c}\langle pk_C\rangle.\ \overline{c}\langle \mathsf{raenc}(pk_A, r_A, pk_B)\rangle.\ R$$

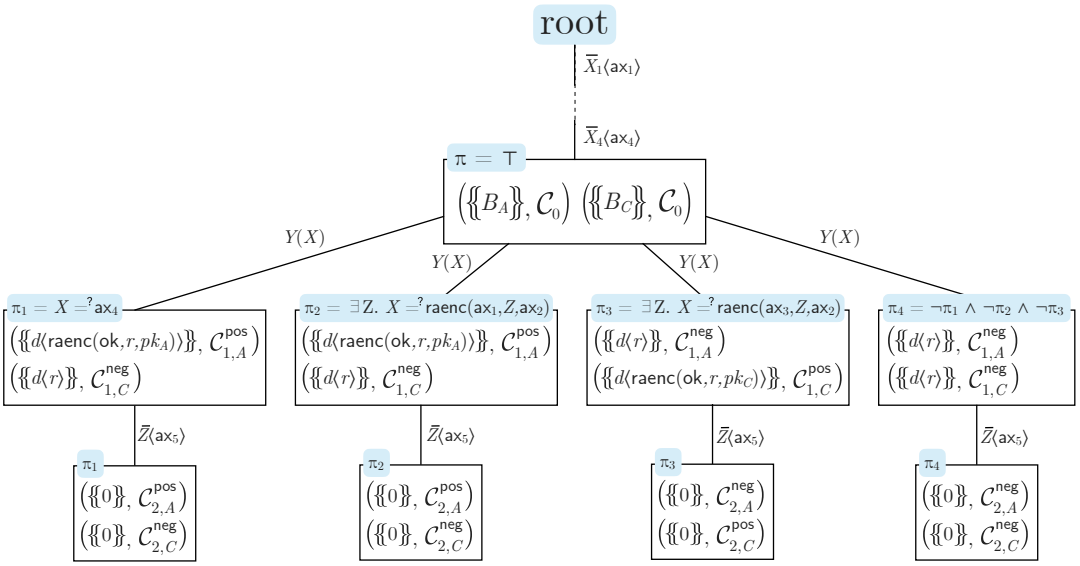


Figure 3.3 A simplified partition tree of P and Q

The partition tree in Figure 3.3 is lightened for the sake of readability, that is, if a node contains two symbolic processes A_s, B_s such that $A_s \xrightarrow{\tau}_{\mathsf{s}} B_s$ then the process A_s contains less constraints than B_s and is omitted from the node. The configuration at the root of the tree only contains P and Q. After the four initial outputs of the context C, we reach the constraint system $\mathcal{C}_0$ defined by:

$$\begin{aligned}\Phi(\mathcal{C}_0) &= \{\mathsf{ax}_1 \mapsto pk_A, \mathsf{ax}_2 \mapsto pk_B, \mathsf{ax}_3 \mapsto pk_C, \mathsf{ax}_4 \mapsto \mathsf{raenc}(pk_A, r_A, pk_B)\}\\ \mathsf{D}(\mathcal{C}_0) &= X_1 \vdash^? x_1 \wedge X_2 \vdash^? x_2 \wedge X_3 \vdash^? x_3 \wedge X_4 \vdash^? x_4\\ \mathsf{E}^1(\mathcal{C}_0) &= x_1 =^? c \wedge x_2 =^? c \wedge x_3 =^? c \wedge x_4 =^? c\end{aligned}$$

The next step is the first one inducing a non-trivial case analysis. This node has four children, each corresponding to a way for the adversary to compute the input $d(x)$: π_1 forwards the message of A, π_2 forges a message pretending it is from A, π_3 forges a message pretending it is from C, π_4 any other case. Depending on the case this will trigger the positive or the negative branch of the conditional in the processes B_A and B_C. More precisely we write $\Phi(\mathcal{C}^{\mathsf{pos}}_{1,X}) = \Phi(\mathcal{C}^{\mathsf{neg}}_{1,X}) = \Phi(\mathcal{C}_0)$, $\mathsf{D}(\mathcal{C}^{\mathsf{pos}}_{1,X}) = \mathsf{D}(\mathcal{C}^{\mathsf{neg}}_{1,X}) = \mathsf{D}(\mathcal{C}_0) \wedge Y \vdash^? y$ and

$$\begin{aligned}\mathsf{E}^1(\mathcal{C}^{\mathsf{pos}}_{1,X}) &= \mathsf{E}^1(\mathcal{C}_0) \wedge y =^? d \wedge x =^? \mathsf{raenc}(pk_X, x', pk_B)\\ \mathsf{E}^1(\mathcal{C}^{\mathsf{pos}}_{1,X}) &= \mathsf{E}^1(\mathcal{C}_0) \wedge y =^? d \wedge \forall x'.\, x =^? \mathsf{raenc}(pk_X, x', pk_B)\end{aligned}$$

Then the final transitions simply execute the resulting outputs, i.e. $\mathcal{C}^s_{2,X}$, $s \in \{\mathsf{pos}, \mathsf{neg}\}$, is obtained by adding $Z \vdash^? z$ and $z =^? d$ to $\mathcal{C}^s_{1,X}$. Since a ciphertext is indistinguishable from a nonce, the two outputs always end up in the same nodes; that is, all leaves contain at least one process originated from P and at least one from Q, which is how we prove trace equivalence. The situation would be different with a rewrite rule such as $\mathsf{testEnc}(\mathsf{raenc}(x, y, \mathsf{pk}(z))) \to \mathsf{ok}$; a partition tree of P and Q with this extended theory can be found in Figure 3.4.

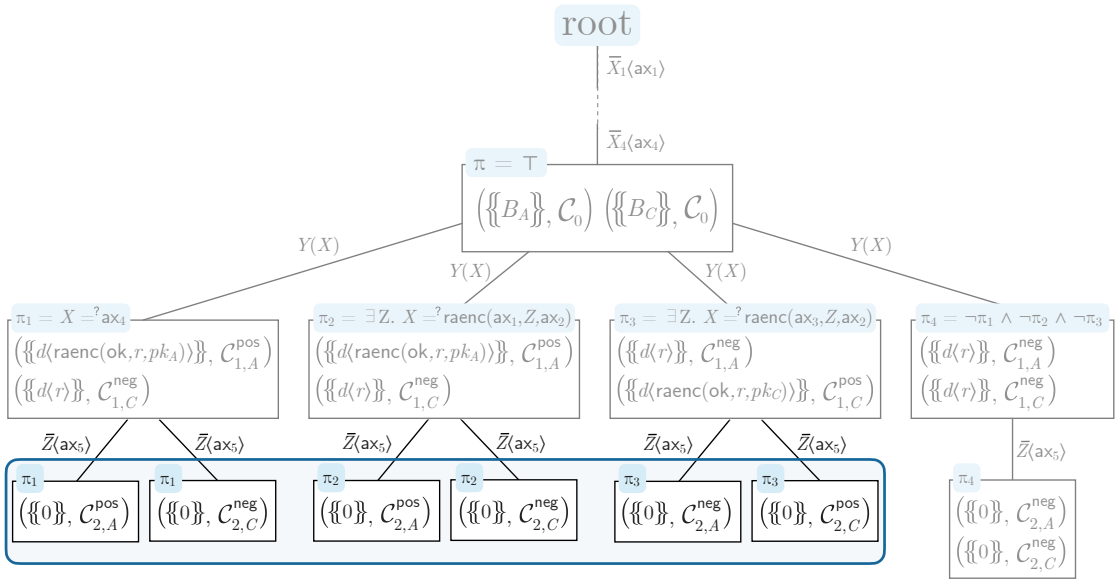


Figure 3.4 Partition tree with the theory extended with $\mathsf{testEnc}(\mathsf{raenc}(x, y, \mathsf{pk}(z))) \to \mathsf{ok}$

We highlighted the part differing from the previous tree. Essentially some leaf nodes have been split in two due to the enhanced capabilities of the adversary to disprove static equivalence, inducing a violation of trace equivalence. For example the leftmost leaf's mgs is

$$\{X \mapsto \mathsf{ax}_4, X_1 \mapsto d, \ldots, X_4 \mapsto d, Y \mapsto d, Z \mapsto d\}$$

which corresponds to an attack trace where the attacker forwards the message of A and observes whether the response of B is a ciphertext, which reveals whether B accepts connections from A or not. ◀

In the remaining of the section we formalise how to decide trace equivalence and labelled bisimilarity of two processes, given a partition tree the mgs' of each of its nodes. For that we will rely on the following notion of reduction:

Definition 3.11 (partition-tree trace)

Let T be a partition tree. We write $(\mathcal{P},\mathcal{C}), n \xrightarrow{\alpha}_T (\mathcal{P}',\mathcal{C}'), n'$ when:

1. n and n' are nodes of T such that $(\mathcal{P},\mathcal{C}) \in \Gamma(n)$ and $(\mathcal{P}',\mathcal{C}') \in \Gamma(n')$; and
2. if $\alpha = \tau$ then $n = n'$, otherwise $n \xrightarrow{\alpha} n'$ and $(\mathcal{P},\mathcal{C}) \xrightarrow{\alpha}_{\mathsf{s}} (\mathcal{P}',\mathcal{C}')$.

For convenience this notion is to be understood up to alpha renaming of the variables of the symbolic action α. We write $A^s_0, n_0 \overset{\mathsf{tr}}{\Rightarrow}_T A^s_p, n_p$ instead of $A^s_0, n_0 \xrightarrow{\alpha_1}_T \cdots \xrightarrow{\alpha_p}_T A^s_p, n_p$ if tr is the word obtained after removing τ symbols from $\alpha_1 \cdots \alpha_p$. If P is a plain process we may also write $P \overset{\mathsf{tr}}{\Rightarrow}_T A_s, n$ instead of $(\{\!\{P\}\!\}, (\varnothing, \top, \top)), root(T) \overset{\mathsf{tr}}{\Rightarrow}_T A_s, n$.

2.2 Deciding trace equivalence

As hinted in our various examples, we now formalise how trace equivalence can be reduced to an analogue form of equivalence using the finite transition relation $\rightarrow_T$, given a partition tree. More precisely the goal of this section is to prove the following theorem:

Theorem 3.3 *(partition-tree-based characterisation of trace equivalence)*

If $T \in \mathsf{PTree}(P_1, P_2)$, the following points are equivalent:

1. $P_1 \sqsubseteq_t P_2$
2. for all partition-tree traces $P_1 \overset{\mathsf{tr}}{\Rightarrow}_T (\mathcal{P}_1,\mathcal{C}_1), n$, we have $P_2 \overset{\mathsf{tr}}{\Rightarrow}_T (\mathcal{P}_2,\mathcal{C}_2), n$

We rely on the soundness and completeness of the symbolic semantics, as well as two technical lemmas generalising the edge properties of the partition tree to branches. For example we can generalise as follows the fact that the nodes of the tree are labelled by maximal configurations, i.e., Definition 3.10, Item 4:

Lemma 3.4

Assume that $(\mathcal{P}_1,\mathcal{C}_1), n \overset{\mathsf{tr}}{\Rightarrow}_T (\mathcal{P}'_1,\mathcal{C}'_1), n'$ and $(\mathcal{P}_2,\mathcal{C}_2) \overset{\mathsf{tr}}{\Rightarrow}_{\mathsf{s}} (\mathcal{P}'_2,\mathcal{C}'_2)$ with $(\mathcal{P}_2,\mathcal{C}_2) \in \Gamma(n)$. We also consider, for all $i \in \{1,2\}$, a solution $(\Sigma', \sigma'_i) \in Sol^{\pi(n')}(\mathcal{C}'_i)$ such that $\Phi(\mathcal{C}'_1)\sigma'_1 \sim \Phi(\mathcal{C}'_2)\sigma'_2$. Then we have $(\mathcal{P}_2,\mathcal{C}_2), n \overset{\mathsf{tr}}{\Rightarrow}_T (\mathcal{P}'_2,\mathcal{C}'_2), n'$.

This lemma can be proved by induction on the length of tr; the proof in question can be found in Appendix A. Combined with the soundness and the completeness of the symbolic semantics, this permits to prove one direction of Theorem 3.3:

Proof of Theorem 3.3, 1⇒2.
Let us consider a trace $P_1 \overset{\mathsf{tr}}{\Rightarrow}_T (\mathcal{P}_1, \mathcal{C}_1), n$ and exhibit a trace $P_2 \overset{\mathsf{tr}}{\Rightarrow}_T (\mathcal{P}_2, \mathcal{C}_2), n$. We decompose the proof into the following steps:

1. By *soundness* of the symbolic semantics we obtain a trace $P_1 \overset{\mathsf{tr}\Sigma}{\Longrightarrow} (\mathcal{P}_1\sigma_1, \Phi(\mathcal{C}_1)\sigma_1\downarrow)$ for an arbitrary solution $(\Sigma, \sigma_1) \in Sol(\mathcal{C}_1)$.
2. By *hypothesis 1* there exists a concrete trace $P_2 \overset{\mathsf{tr}\Sigma}{\Longrightarrow} (\mathcal{P}, \Phi)$ such that $\Phi \sim \Phi(\mathcal{C}_1)\sigma\downarrow$.
3. By *completeness* of the symbolic semantics we obtain a symbolic trace $P_2 \overset{\mathsf{tr}'}{\Longrightarrow}_{\mathsf{s}} (\mathcal{P}_2, \mathcal{C}_2)$ and $(\Sigma', \sigma_2) \in Sol(\mathcal{C}_2)$ such that $\mathsf{tr}\Sigma = \mathsf{tr}'\Sigma'$, $\mathcal{P}_2\sigma_2 = \mathcal{P}$ and $\Phi(\mathcal{C}_2)\sigma_2\downarrow = \Phi$. Due to the form of symbolic actions, we know that there exists a second-order-variable renaming ϱ such that $\mathsf{tr} = \mathsf{tr}'\varrho$; in particular $P_2 \overset{\mathsf{tr}}{\Rightarrow}_{\mathsf{s}} (\mathcal{P}_2, \mathcal{C}_2\varrho)$ and $(\Sigma, \sigma_2) \in Sol(\mathcal{C}_2\varrho)$.
4. By *Lemma 3.4* we therefore obtain that $P_2 \overset{\mathsf{tr}}{\Rightarrow}_T (\mathcal{P}_2, \mathcal{C}_2\varrho), n$, which gives the expected conclusion. □

The second property of the partition tree we extend is the fact that symbolic transitions are reflected in the tree, i.e., Definition 3.10, Item 2:

Lemma 3.5
Let n be a node of a partition tree T and $(\mathcal{P}, \mathcal{C}) \in \Gamma(n)$. If $(\mathcal{P}, \mathcal{C}) \overset{\mathsf{tr}}{\Rightarrow}_{\mathsf{s}} (\mathcal{P}', \mathcal{C}')$ and $(\Sigma, \sigma) \in Sol^{\pi(n)}(\mathcal{C}')$ then there exist a node n' and a substitution Σ' such that $(\mathcal{P}, \mathcal{C}), n \overset{\mathsf{tr}}{\Rightarrow}_T (\mathcal{P}', \mathcal{C}'), n'$ and $(\Sigma', \sigma) \in Sol^{\pi(n')}(\mathcal{C}')$.

Note that unlike the definition of partition tree, we do not require that Σ' coincides with Σ on $vars^2(n)$. This additional requirement would not make the lemma false but appears to be unecessary to prove Theorem 3.3. The lemma is proved by induction on tr in Appendix A.

Proof of Theorem 3.3, 2⇒1.
Let us consider a trace $P_1 \overset{\mathsf{tr}}{\Rightarrow} (\mathcal{P}, \Phi)$ and exhibit a trace $P_2 \overset{\mathsf{tr}}{\Rightarrow} (\mathcal{Q}, \Psi)$ such that $\Phi \sim \Psi$. We decompose the proof into the following steps:

1. By *completeness* of the symbolic semantics we obtain a symbolic trace $P_1 \overset{\mathsf{tr}_s}{\Longrightarrow}_{\mathsf{s}} (\mathcal{P}_1, \mathcal{C}_1)$ and $(\Sigma, \sigma_1) \in Sol(\mathcal{C})$ such that $\mathsf{tr}_s\Sigma = \mathsf{tr}$, $\mathcal{P}_1\sigma_1 = \mathcal{P}$ and $\Phi(\mathcal{C}_1)\sigma_1\downarrow = \Phi$.
2. By *Lemma 3.5* we then obtain a partition-tree trace $P_1 \overset{\mathsf{tr}_s}{\Longrightarrow}_T (\mathcal{P}_1, \mathcal{C}_1), n$ and Σ' such that $(\Sigma', \sigma_1) \in Sol^{\pi(n)}(\mathcal{C}_1)$.
3. By *hypothesis 2* there also exists a partition-tree trace $P_2 \overset{\mathsf{tr}_s}{\Longrightarrow}_T (\mathcal{P}_2, \mathcal{C}_2), n$. By definition of a configuration we also know that there exists σ_2 such that $(\Sigma', \sigma_2) \in Sol^{\pi(n)}(\mathcal{C}_2)$ and $\Phi(\mathcal{C}_1)\sigma_1 \sim \Phi(\mathcal{C}_2)\sigma_2$.
4. By *soundness* of the symbolic semantics applied to we then obtain a concrete trace $P_2 \overset{\mathsf{tr}_s\Sigma'}{\Longrightarrow} (\mathcal{Q}, \Psi)$ with $\mathcal{Q} = \mathcal{P}_2\sigma_2$ and $\Psi = \Phi(\mathcal{C}_2)\sigma_2\downarrow \sim \Phi(\mathcal{C}_1)\sigma_1\downarrow = \Phi$.

However we may have $\mathsf{tr}_s\Sigma' \neq \mathsf{tr}$ and, to conclude the proof, we prove that $P_2 \overset{\mathsf{tr}}{\Rightarrow} (\mathcal{Q}, \Psi)$ as well. For that it sufficies to prove that $\mathsf{tr}\Psi =_E \mathsf{tr}_s\Sigma'\Psi$, that is, although the recipes or tr and $\mathsf{tr}_s\Sigma'$ are different they produce the same first-order terms. Since Φ and Ψ are statically equivalent, $\mathsf{tr} = \mathsf{tr}_s\Sigma$ and $\Phi = \Phi(\mathcal{C}_1)\sigma_1\downarrow$, it sufficies to prove that $\mathsf{tr}_s\Sigma\Phi(\mathcal{C}_1)\sigma =_E \mathsf{tr}_s\Sigma'\Phi(\mathcal{C}_1)\sigma$. Let $X \in vars^2(\mathsf{tr}_s)$. A quick look at the rules of the symbolic semantics shows that there exists a deduction fact $(X \vdash^? x) \in \mathsf{D}(\mathcal{C}_1)$. In particular, since (Σ, σ_1) and (Σ', σ_1) are both solutions of $\mathcal{C}_1$ we have $X\Sigma\Phi(\mathcal{C}_1)\sigma_1 =_E x\sigma_1 =_E X\Sigma'\Phi(\mathcal{C}_1)\sigma_1$, hence the conclusion. □

2.3 Deciding labelled bisimilarity

In the case of trace equivalence, a witness that $A \not\approx_t B$ is simply a trace of A or B that has no equivalent trace in the other process. The case of labelled bisimilarity is more involved. Using vocabulary borrowed from game theory, the definition of bisimilarity can be seen as an *prover-disprover game*: at each state of the game the disprover chooses a transition from one of the two processes and the prover answers by choosing a transition of the same type from the other process (plus some potential τ-transitions). The disprover wins the game if they manage to reach a state with non-statically-equivalent processes or if the prover cannot answer to one of the moves: a witness of non-equivalence is then a winning strategy for the disprover in this game. We formalise this as follows; we recall that if α is an action, we write $\bar{\alpha} = \alpha$ if $\alpha \neq \tau$ and $\bar{\alpha} = \varepsilon$ if $\alpha = \tau$.

Definition 3.12 (witness of non-bisimilarity)

A *witness* w is a set of pairs (A_0, A_1) verifying the following two conditions:

1. A_0 and A_1 are ground extended processes such that $A_0 \sim A_1$
2. there exists $b \in \{0,1\}$ and a transition $A_b \xrightarrow{\alpha} A'_b$ such that for all traces $A_{1-b} \stackrel{\bar{\alpha}}{\Rightarrow} A'_{1-b}$, either $A'_0 \not\sim A'_1$ or $(A'_0, A'_1) \in \mathsf{w}$.

We say that w is a witness for (A_0, A_1) if $(A_0, A_1) \in \mathsf{w}$.

Proposition 3.6 (witness-based characterisation of labelled bisimilarity)

If $A_0 \sim A_1$ then $A_0 \not\approx_l A_1$ *iff* there exists a witness w for (A_0, A_1).

Proof. First, we observe that $A_0 \not\approx_l A_1$ *iff* there exists a binary relation $\mathcal{S}$ on ground extended processes such that $A_0 \mathcal{S} A_1$ and, for all $(B_0, B_1) \in \mathcal{S}$, either 1 $B_0 \not\sim B_1$, or 2 there exists $b \in \{0,1\}$ and a transition $B_b \xrightarrow{\alpha} B'_b$ such that for all traces $B_{1-b} \stackrel{\bar{\alpha}}{\Rightarrow} B'_{1-b}$, $B'_0 \mathcal{S} B'_1$. Let us call such a relation $\mathcal{S}$ a *labelled attack on* (A_0, A_1). Since processes are bounded there exist no infinite sequences of transitions and $A \not\approx_l B$ therefore straightforwardly rephrases to the existence of a labelled attack $\mathcal{S}$ such that $A \mathcal{S} B$. It then sufficies to observe that

1. If $\mathcal{S}$ is a labelled attack on (A_0, A_1) then $\mathcal{S} \setminus \not\sim$ is a witness for (A_0, A_1).
2. If w is a witness for (A_0, A_1) then $\mathsf{w} \cup \not\sim$ is a labelled attack on (A_0, A_1). □

We now define a symbolic variant of the notion of witness that can be constructed within a partition tree T. In essence a symbolic witness can therefore be seen as a winning strategy for the disprover in a bisimulation game limited to the finite transition relation $\rightarrow_T$.

Definition 3.13 (symbolic witness of non-bisimilarity)

A *symbolic witness* w_s w.r.t. a partition tree T is a finite tree whose nodes N are labelled by pairs (S, n) with n a node of T and $S \subseteq \Gamma(n)$ is either a singleton or contains exactly two elements. We also require that if N is labelled $(\{A_0, A_1\}, n)$, there exist $b \in \{0,1\}$ and a transition $A_b, n \xrightarrow{\alpha}_T A'_b, n'$ (possibly $\alpha = \tau$) such that:

1. If A_{1-b} is not reducible by $\stackrel{\bar{\alpha}}{\Rightarrow}_T$ then N has a unique child labelled $(\{A'_b\}, n')$;
2. otherwise the children of N are the nodes labelled $(\{A'_0, A'_1\}, n')$, $A_{1-b}, n \stackrel{\bar{\alpha}}{\Rightarrow}_T A'_{1-b}, n'$.

We say that w_s is a symbolic witness for A_0, A_1, n when $root(\mathsf{w}_s)$ is labelled by $(\{A_0, A_1\}, n)$.

However purely symbolic witnesses do not exhibit consistent proofs of non-equivalence in general. Indeed a concrete execution fixes the effective value of an input x at the moment it is performed, whereas in a symbolic execution the constraints on x are collected later, all along the trace. In some sense the symbolic semantics puts the prover at a disadvantage in the game, since they have to answer to the disprover's input actions without knowing the values of the input terms. Symbolic witnesses inducing invalid winning strategies for the disprover will be discarded in that they have no *solutions* in the following sense:

Definition 3.14 (solution of a symbolic witness)

Let w_s be a symbolic witness. A *solution* of w_s is a function f_{sol} that maps nodes of w_s to ground second-order substitutions such that for all nodes N labelled (S, n),

1 for all $(\mathcal{P}, \mathcal{C}) \in S$, $(f_{\mathsf{sol}}(N), \sigma) \in Sol^{\pi(n)}(\mathcal{C})$ for some σ;

2 for all children nodes N_1, N_2 of N, $f_{\mathsf{sol}}(N) \subseteq f_{\mathsf{sol}}(N_1) = f_{\mathsf{sol}}(N_2)$.

We denote $Sol(\mathsf{w}_s)$ the set of solutions of w_s.

Theorem 3.7 *(partition-tree-based characterisation of labelled bisimilarity)*

If $T \in \mathsf{PTree}(P_1, P_2)$, the following points are equivalent:

1 $P_0 \approx_l P_1$

2 for all symbolic witnesses w_s for $(P_1, P_2, root(T))$, we have $Sol(\mathsf{w}_s) = \varnothing$

The proof, although technical, simply connects the symbolic witnesses to concrete ones using the soundness and completeness of the symbolic semantics as well as the properties of the partition tree, following similar ideas as the analogue proof for trace equivalence. The detailed proof can be found in Appendix A, Lemma A.1.

Assuming one has computed a partition tree $T \in \mathsf{PTree}(P_1, P_2)$ and the mgs of each of its nodes, since there are finitely-many possible symbolic witnesses, Theorem 3.7 yields a decision procedure for the labelled bisimilarity of P_1 and P_2 provided one can decide whether a given symbolic witness has a solution. For that we rely on a simple, bottom-up unification of the mgs' appearing in the witness; details can be found in Chapter 6, Section 3 where we study more precisely the complexity of partition-tree-based decision procedures.

3 Generating partition trees (with oracle to a constraint solver)

Section summary

We explain how to generate the partition tree, assuming an *oracle to a contraint-solving algorithm* that is described in more details in Chapter 4. The nodes are generated and refined from the root to the leaves, thus permitting to *distribute* the generation. The tool implementation, DeepSec, is compared against similar analysers in terms of verification time on benchmarks including those presented in Chapter 2.

3.1 Overview of the top-down tree generation

In this section we detail the skeleton of the procedure for computing a partition tree of two plain processes P_1 and P_2. The description is modular in that most of the technical details,

in particular the modelling of the node predicates and how we obtain the expected properties of the tree, are abstracted by a *constraint-solving oracle* that we detail in the next chapter. This section should therefore be seen as the overview of the whole algorithm for deciding equivalence properties, which gives enough insight to discuss our implementation.

The algorithm generates the nodes of the tree top-down, that is, from the root to the leaves. We outline the procedure in Figure 3.5.

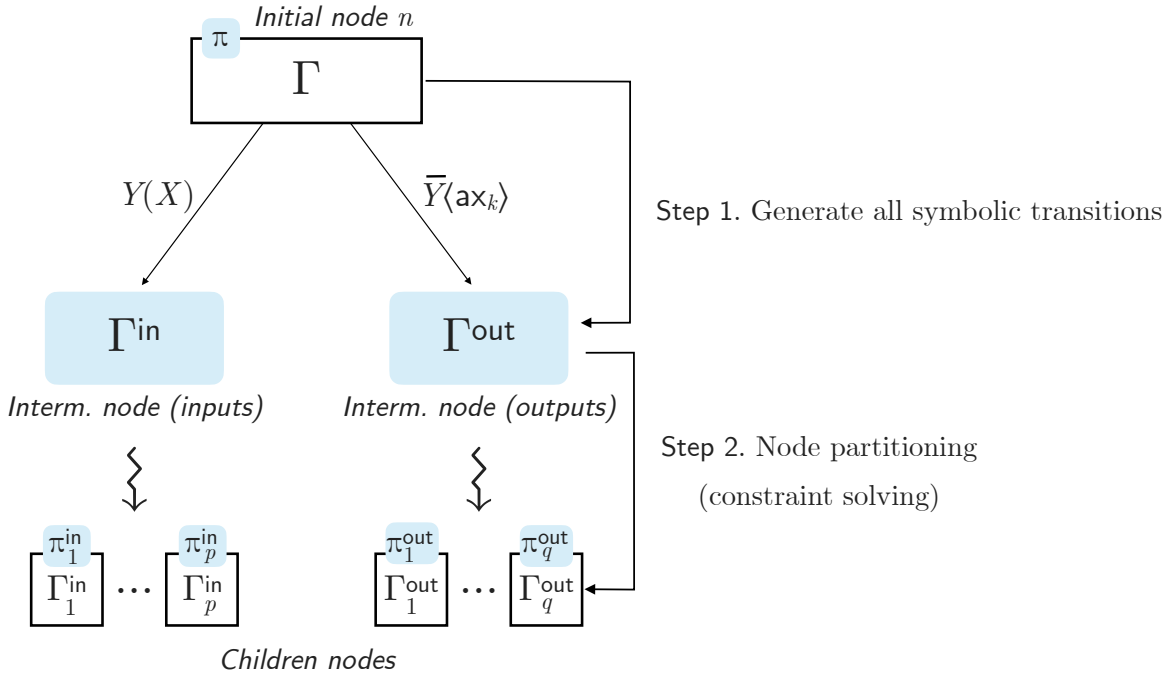


Figure 3.5 Computing the subtree of a partition tree rooted in a node n

Let us now describe the algorithm to compute $T \in \mathsf{PTree}(P_1, P_2)$ in more details, up to the technical developments detailed in the next chapter.

1 First, we initiate a root containing P_1 and P_2 and saturate the configuration by τ transitions. That is, we consider the set of symbolic processes

$$\Gamma(root(T)) = \left\{ (\mathcal{P}, \mathcal{C}) \mid P_i \overset{\varepsilon}{\Rightarrow}_{\mathsf{s}} (\mathcal{P}, \mathcal{C}), i \in \{1,2\}, Sol(\mathcal{C}) \neq \varnothing \right\}$$

Note that the constraint systems $\mathcal{C}$ involved in this definition do not contain deduction facts, which makes the decision of the emptiness of $Sol(\mathcal{C})$ relatively straightforward. Using the terminology of Chapter 4, using *simplification rules* permits to put the constraints into a simple form where the existence of a solution is trivial to decide.

2 Then let us assume we already constructed a node n of the tree using this algorithm, in particular the corresponding configuration $(\Gamma(n), \pi(n))$. To compute the children of n we first enumerate all symbolic transitions from processes of $\Gamma(n)$, separating input and output actions. That is, we compute the two sets

$$\Gamma^{\mathsf{in}} = \left\{ B \mid A \in \Gamma(n), A \xrightarrow{Y(X)}_{\mathsf{s}} B \right\} \qquad \Gamma^{\mathsf{out}} = \left\{ B \mid A \in \Gamma(n), A \xrightarrow{Y(\mathsf{ax}_p)}_{\mathsf{s}} B \right\}$$

3 Γ^{in} and Γ^{out} are two intermediary sets that do not satisfy yet the father-child properties of the partition tree. For that we use a constraint-solving algorithm detailed in Chapter 4 (*simplification rules* again, but also *case distinction rules*) that will partition Γ^{in} and Γ^{out} to gather symbolic processes with statically-equivalent solutions and remove those with no solutions. This constraint solving results into a sequence of configurations

$$(\Gamma_1^{\mathsf{in}}, \pi_1^{\mathsf{in}}), \ldots, (\Gamma_p^{\mathsf{in}}, \pi_p^{\mathsf{in}}) \qquad\qquad (\Gamma_1^{\mathsf{out}}, \pi_1^{\mathsf{out}}), \ldots, (\Gamma_q^{\mathsf{out}}, \pi_q^{\mathsf{out}})$$

that will label the children of n. The procedure is then carried out recursively from these child nodes until no more symbolic transitions are available.

In Chapter 4 we detail the missing parts of this procedure that take the form of constraint-solving rules, in the context of constructor-destructor subterm convergent theories. Note that the approach is modular in that the proofs we have carried so far are independent of the assumptions on the theory: generalising the results of Chapter 4 will automatically result in the decidability of trace equivalence and labelled bisimilarity of bounded processes for the extended class of theories.

3.2 Implementation and performances

The DeepSec prover Building on the procedure's structure described above and the internal solver developed in the next chapter, we have implemented a prototype in Ocaml, called DeepSec (DEciding Equivalence Properties in SECurity protocols). The user specifies a theory (that is checked to be constructor-destructor and subterm convergent by the tool), two bounded processes, and the tool verifies whether they are trace equivalent. If not, a concrete attack trace is returned in a dedicated graphical interface; we refer to the DeepSec's website for development credits, tutorials and details on practical usage [CKRY20]:

https://deepsec-prover.github.io/

The tool's specification language implements the grammar presented in Chapter 1, Sections 2 and 2.3, including non-deterministic choice, private function symbols, a restricted form of patterned let bindings[1], as well as bounded replication $!^n P$ defining n copies of P in parallel. As we explained in Chapter 1 these additional primitives are only here for modelling convinience; yet the native integration allowed specific optimisations compared to encodings within the initial calculus. The syntax and structure of DeepSec's input files are similar to the widely used ProVerif [BSCS20] tool to make it easier for new users to discover and use it.

Partial order reductions The tool also implements *partial order reductions* (POR), an optimisation technique for protocol analysis developed by Baelde et al. [BDH15]. The basic idea is to discard part of the state space that is redundant but this optimisation is only sound when processes are *action-determinate*, as defined in [BDH15]. We will study more formally action determinacy and its implications on decidability and complexity in Chapters 5 and 7 and we omit the definition of this class of processes for now. Still, we mention that not

[1] only patterns of the form $\langle u_1, \ldots, u_n \rangle$ are allowed where each u_i is either a free variable or a term without free variables. That is, we only allow patterns that decompose tuples and possibly check that some components are equal to a given term.

using private channels and assigning a different channel name to each parallel process is a simple, syntactic way to ensure this property although this is not always possible—typically when looking at some anonymity or unlinkability properties. In the benchmarks presented in Chapter 2 the private authentication protocol can be modelled as a determinate process, but not the Helios and BAC protocols (due to private channels or because this introduces artifical violations of the equivalence property).

In practice, DeepSec automatically detects action-determinate processes and activates the POR, which drastically reduces the number of symbolic executions that need to be considered. Going further we study in Chapter 5 how to use similar POR techniques in general, that is, without the restriction to determinate processes. The experimental results presented in the current chapter only include the base POR of [BDH15].

Distributing the computation The main task of DeepSec is to generate a partition tree and, as we explained, this is done using a top-down approach. This task can be distributed as computing a given node of the tree can be done independently of its sibling nodes. However, some engineering is needed to avoid heavy communication overhead due to task scheduling. Indeed, the partition tree is not a balanced tree and we do not know in advance which branches will be larger than others. Because of this, we do not directly compute and return the children of each node in the most straightforward manner, but proceed in two steps:

1. We start with a breadth-first generation of the partition tree. The number of pending nodes will gradually grow until eventually exceeding a threshold parameter n.
2. Each available core focuses on one of these nodes, computes the whole subtree rooted in this node in a depth-first manner and, when this the task is completed, is assigned to a new node until none remain.

If some cores become idle for too long in Step 2 (because the number of non-completed nodes exceeds the number of cores), we perform a *new round*, that is, we interrupt the working nodes and restart this two-step procedure on incomplete nodes. Although doing so wastes some proof work, this improves performances for particularly unbalanced trees. Note that parallelisation is also supported by other automated analysers such as Akiss [CCCK16], but DeepSec goes one step further as it is able to distribute the computation not only on multiple cores of a given machine but also clusters of computers.

Benchmarks We performed extensive benchmarks to compare DeepSec against other tools that verify equivalence properties for a bounded number of sessions: Akiss [CCCK16], APTE [Che14], SatEquiv [CDD17] and SPEC [TNH16]. Experiments are carried out on Intel Xeon 3.10GHz cores, with 40Go of memory. We distributed the computation on 20 cores for Akiss and DeepSec as they support parallelisation—unlike the others which therefore use a single core. The results are summarised in Figures 3.6 and 3.7 with the following symbol conventions:

- ✓ analysis terminates and equivalence holds
- ↯ analysis terminates and an attack is found
- OM analysis aborted due to memory overflow
- ⏱ analysis aborted due to timeout (12 hours)
- ✗ the tool is not expressive enough to analyse the protocol

1 *Scalability examples.*

First of all we study a few examples of protocols and privacy-type properties with a fixed scenario, to measure the scalability of the compared tools on simple examples when increasing the number of sessions (Figure 3.6). We first analysed strong secrecy and anonymity for several classical authentication protocols. The DeepSec tool clearly outperforms Akiss, APTE, and SPEC. The SatEquiv tool becomes more efficient, when the number of sessions significantly increases.

To put more emphasis on the broad scope we also include analyses of unlinkability and anonymity properties for a number of other protocols. We experiments on the examples mentioned in Chapter 2 (still with a fixed scenario), i.e., Private authentication, BAC (property ($\mathsf{UL}_n^{\approx_t}$)) and Helios (vote swap model). In addition we study a simplified version of the AKA protocol deployed in 3G telephony networks without XOR [AMR+12], the Passive Authentication protocol implemented in the European passport [For04], as well as the Prêt-à-Voter protocol (PaV) [RS06]. Note that, while PaV is a priori in the scope of Akiss, it failed to produce a proof: Akiss only approximates trace equivalence of non-determinate processes and finds a false attack here. Finally we note that BAC, PaV and Helios protocols are not action-determinate and therefore do not benefit from the POR optimisation, which explains the much higher verification times when increasing the sessions. Nevertheless, as exemplified by some examples, attacks may be found very efficiently, as it generally does not require to explore the entire state space.

2 *Full fledged examples.*

Now that we have compared the performances of DeepSec with other tools, we focus more on its capabilities to handle complex scenarios by analysing the different models developed in Chapter 2—which, we recall, are parametric in the scenario by involving the adversary and may include adversarial sessions. The experimental results can be found in Figure 3.7. We refer to the discussions of Chapter 2 for the description of the protocols, assumptions on the adversary, and interpretations of the violations found. Let us still emphasise that DeepSec in unable to analyse 2 roles of BAC with potential adversarial sessions in a 12h timeout. To scale to this example we will need the optimisation techniques of Chapter 5.

	Protocol (# of roles)		Akiss	APTE	SPEC	SatEquiv	DeepSec
Strong secrecy	Denning-Sacco	3	✓ <1s	✓ <1s	✓ 11s	✓ <1s	✓ <1s
		6	✓ <1s	✓ 1s	OM	✓ <1s	✓ <1s
		7	✓ 6s	✓ 3s		✓ <1s	✓ <1s
		10	OM	✓ 9m49		✓ <1s	✓ <1s
		12		⏱		✓ <1s	✓ <1s
		29				✓ <1s	✓ 1s
	Wide Mouth Frog	3	✓ <1s	✓ <1s	✓ 5s	✓ <1s	✓ <1s
		6	✓ <1s	✓ <1s	✓1h11m	✓ <1s	✓ <1s
		7	✓ <1s	✓ 1s	OM	✓ <1s	✓ <1s
		10	✓ 10s	✓ 3m35		✓ <1s	✓ 1s
		12	✓22m16s	⏱		✓ <1s	✓ <1s
		14	OM			✓ <1s	✓ <1s
		23				✓ <1s	✓ 1s
	Yahalom-Lowe	3	✓ <1s	✓ <1s	✓ 7s	✓ <1s	✓ <1s
		6	✓ 2s	✓ 41s	OM	✓ <1s	✓ <1s
		7	✓ 42s	✓34m38s		✓ 1s	✓ <1s
		10	OM	⏱		✓ 1s	✓ <1s
		12				✓ 4s	✓ 2s
		14				✓ 7s	✓ 2s
		17				✓ 12s	✓ 8s
Anonymity	Private Authentication	2	✓ <1s	✓ <1s	✗	✗	✓ <1s
		4	✓ <1s	✓ 1s			✓ <1s
		6	✓ 21s	✓ 4m18s			✓ <1s
		8	OM	⏱			✓ 1s
		10					✓ 2s
		15					✓ 32s
Unlinkability	3G-AKA	4	✓ 1m35s	✓ 1h23m	✗	✗	✓ <1s
		6	OM	⏱			✓ 2s
	Passive Authentication	4	✓ <1s	✓ 1s	✗	✗	✓ <1s
		6	✓ 2m15s	✓ 1m27s			✓ <1s
		7	✓ 1h40m	✓ 1m44s			✓ 1s
		9	⏱	✓ 2h08m			✓ <1s
		15		⏱			✓ 9s
		21					✓ 15s
	BAC	4	OM	↯ 38m56s	✗	✗	↯ 1s
		6		⏱			⏱
Ballot privacy	Prêt-à-Voter	6	✗	✗	✗	✗	✓ 2s
	Helios Vanilla	6	↯ 47s	↯ <1s	✗	✗	↯ <1s
	Helios ZKP (vote swap)	10					✓ <1s
		11	OM	✗	✗	✗	✓7m 24s
		12					✓1h 38m

Figure 3.6 Performances of DeepSec (20 cores) against other protocol analysers (**fixed scenario**)

		Protocol + roles		Analysis
Private authentication (anonymity and strong secrecy)		only honest roles	**4** roles	✓ <1s
			7 roles	✓ 6s
			8 roles	✓ 43s
			9 roles	✓ 7m 41s
			10 roles	✓ 1h 30m
		arbitrary roles	**1** role	✓ <1s
			2 roles	↯ <1s
		arbitrary roles (+ decoy nonce excl.)	**3** roles	✓ 1s
			4 roles	✓ 30s
			5 roles	✓ 1h 18m
Helios (BPRIV ballot privacy)	ZKP with revote	**2** voters + honest ballot box	**3** ballots	✓ <1s
			8 ballots	✓ 44s
			9 ballots	✓ 4m 33s
			10 ballots	✓ 10m 44s
			11 ballots	✓ 3h 21m
		3 voters + honest ballot box	**3** ballots	✓ <1s
			6 ballots	✓ 11s
			7 ballots	✓ 1m 25s
			8 ballots	✓ 11m 52s
			9 ballots	✓ 1h 40m
		2 voters + ∼dishonest ballot box	**1** ballot	✓ <1s
			2 ballots	✓ <1s
			3 ballots	✓ 1m 3s
	vanilla	**2** voters + honest ballot box	**1** ballot	✓ <1s
			2 ballots	↯ <1s
	weeding	**2** voters + honest ballot box	**1** ballot	✓ <1s
			2 ballots	✓ <1s
		2 voters + ∼dishonest ballot box	**1** ballot	✓ <1s
			2 ballots	↯ <1s
BAC		only honest sessions	**2** sessions	✓ 13m 12s

Figure 3.7 Performances of DeepSec (20 cores) (**parametric scenario**)

Chapter 4:

DeepSec's constraint solver

Summary.

Technical Chapter. In this chapter we detail how to generate *partition trees*, the key tool used in Chapter 3 to prove equivalence of bounded processes. Our procedure is correct for any constructor-destructor subterm convergent theories. As sketched in the previous chapter, the generation mostly consists of *constraint solving* procedures that refine and partition a set of symbolic constraints to separate those that have non-statically-equivalent solutions. Termination and complexity-related concerns are investigated in the corresponding part of the thesis (Part III, Chapter 6).

1 Extended constraint systems

> ***Section summary***
> In order to carry out the constraint solving required to construct the partition tree, we extend constraint systems with components allowing to reason more finely about the *attacker's knowledge*. The notion of solution of constraint system is also extended to capture their expected properties.

1.1 Knowledge base and formulas

New constraints From now on we assume an implicit theory E that is constructor-destructor and subterm convergent. We introduce an extension of constraint systems with second-order constraints that serve key roles in the generation of the partition tree:

▷ *Giving a finite representation of the deductive capabilities of the attacker.*

This takes the form of a *knowledge base* K which is a finite set of deduction facts. By relying on subterm convergence among others, our procedure will ensure that a term u is deducible *iff* it can be deduced by applying constructor symbols to deduction facts of K, which makes deducibility easily decidable due to the constructor-destructor property. In particular we will only consider solutions that compute terms using entries of K this way.

▷ *Giving a finite representation of the distinguishing capabilities of the attacker.*

This takes the form of a set of *formulas* F that is, in short, a finite representation of the term equalities that hold in the current frame. In particular static equivalence will be characterisable only from the formulas of F.

▷ *Recording the constraints imposed on second-order solutions during the constraint solving.*

When computing most general solutions or performing case analyses on the form of solutions, we track the resulting effect on second-order solution in a set E^2 that is the second-order analogue of E^1. This is mostly how we model the predicates that appear in the configurations in partition trees (Chapter 3, Definition 3.9).

More formally we consider, in addition to deduction facts and second-order equations, a new atomic second-order constraint, *equality facts* $\xi =_f^? \zeta$, ξ and ζ second-order terms. Unlike second-order equations that model syntactic equalities, equality facts capture equalities modulo theory, that is, the fact that ξ and ζ deduce the same first-order term. Concretely we extend the relation $\models$ (Chapter 3, Section 1.3) with

$$(\Phi, \Sigma, \sigma) \models \xi\Sigma\Phi\sigma =_E \zeta\Sigma\Phi\sigma$$

We now define the constraints that are typically put in the set F.

Definition 4.1 (deduction formula, equality formula)

A *deduction* (resp. *equality*) *formula* is a constraint of the form $\forall S.\, (C_1 \wedge \ldots \wedge C_n) \Rightarrow H$:

1. S is a set of (both first-order and second-order) variables;
2. H is a deduction fact (resp. an equality fact);
3. for all $i \in \{1, \ldots, n\}$, C_i is either a deduction fact of the form $X \vdash^? t$, $X \in \mathcal{X}^2$, or a first-order syntactic equation $u =^? v$.

A formula ψ is called *solved* when it contains no hypotheses, i.e., $\psi = (\forall \varnothing.\, \top \Rightarrow H) = H$. Given a formula $\psi = \forall S.\, \varphi \Rightarrow H$, we denote by $\mathsf{hyp}(\psi)$ the set of the syntactic equations appearing in the hypotheses φ, and by $\mathsf{D}(\psi)$ the set of deduction facts in φ.

Intuitively, a formula captures a deduction or comparison that the attacker may perform and the premisses $C_1, \ldots, C_n$ express conditions under which this is possible. Typically if the attacker observed a ciphertext $\mathsf{rsenc}(m, r, k)$ (bound to an axiom ax), we may express the deducibility of m through the formula

$$\forall X.\, X \vdash^? k \Rightarrow \mathsf{rsdec}(\mathsf{ax}, X) \vdash^? m$$

Another example is the following formula that expresses the tautology that two recipes deducing the same term should be equal in the sense of an equality fact:

$$\forall X, Y, z.\, (X \vdash^? z \wedge Y \vdash^? z) \Rightarrow X =_f^? Y$$

This formula will serve as a generic placeholder when computing equality formulas during the constraint solving, that is, we will always add equality formulas obtained by substituting variables in the above formula. Although we consider arbitrary formulas such as the above two during the computation of the partition tree, note that only formulas of a certain shape will eventually be added in the set F recording the attacker's distinguishing capabilities. We give more details on the invariants of the procedure in Appendix B, Section 1 but we can mention for example that the formulas effectively recorded in F will be of the form $\varphi \Rightarrow H$, i.e., there are no universally-quantified variables, and φ only contains first-order equations.

Extended constraint systems We now formalise how we extend constraint systems to store the knowledge base, formulas, and to capture restrictions on the form of solutions.

Definition 4.2 (extended constraint system)

An *extended constraint system* $\mathcal{C}^e$ is a tuple $(\Phi, \mathsf{D}, \mathsf{E}^1, \mathsf{E}^2, \mathsf{K}, \mathsf{F})$ where:

1. $(\Phi, \mathsf{D}, \mathsf{E}^1)$ is a constraint system, although more general in that D may contain constraints of the form $X \vdash^? u$ or $\forall X.\, X \nvdash^? u$ where u may be an arbitrary constructor term;
2. E^2 is a set of second-order equations and constraints of the form $\forall Y_1, \ldots, Y_k. \bigvee_{j=1}^{p} \xi_j \neq^? \zeta_j$
3. K is a set of deduction facts;
4. F is a set of deduction and equality formulas.

As explained earlier, the set E^2 gathers constraints to be satisfied by the second-order solutions of the system, K is a finite representation of the attacker knowledge, and F characterises the attacker capabilities to deduce and compare terms modulo theory. In particular the set E^2 contains additional constraints to be satisfied by solutions while K and F are valid formulas that characterise potential attacker actions. For example, the (unsolved) deduction formulas in F reason about potentially deducible terms: when such formula contains premisses, the procedure will perform a case analysis to distinguish cases where the hypotheses hold or not, leading to solved or trivial formulas, respectively. When a solved deduction formula is obtained this way, we add it to the knowledge base K if u is not already deducible from it.

1.2 (Most general) solutions

We now define how the notion of solutions is lifted to extended constraint systems and how this embeds the predicates π used in the definition of partition-tree configurations. The definition of a solution (Σ, σ) of $\mathcal{C}^e$ follows three guidelines: 1 it should be a solution in the usual sense and satisfy $\mathsf{E}^2(\mathcal{C}^e)$; 2 the set of formulas $\mathsf{F}(\mathcal{C}^e)$ plays no role in the definition of solutions: we will only prove invariants that this set verifies during our specific constraint–solving procedure (see Appendix B, Section 1); and 3 all recipes used in the solution should have been constructed from the knowledge base $\mathsf{K}(\mathcal{C}^e)$, *uniformly* (that is, a same first-order term should not be deduced by different recipes in the solution). In particular this requires a notion of *consequence*, indicating that a recipe can be deduced from the knowledge base.

Definition 4.3 (consequence)

We define the set of *consequences* of a set of deduction facts S, denoted $\mathsf{Conseq}(S)$, as the set of pairs $(C[\xi_1, \ldots, \xi_n], C[u_1, \ldots, u_n])$ where C is a context built using $\mathcal{F}_\mathsf{c} \cup \mathcal{F}_0$ and for all $i \in \{1, \ldots, n\}$, $\xi_i \vdash^? u_i \in S$. We write $\xi \in \mathsf{Conseq}(S)$ if $\exists t.\, (\xi, t) \in \mathsf{Conseq}(S)$.

We recall that by definition a deduction fact never has a constructor function symbol at its root (Chapter 3, Definition 3.3): in particular if $\xi \in \mathsf{Conseq}(S)$, the context C in the above definition is unique. Writing $\xi = C[\xi_1, \ldots, \xi_n]$ it is therefore possible to define unambiguously the set of *consequential subterms* of ξ

$$subterms_\mathsf{c}(\xi, S) = \{\xi_{|p} \mid p \text{ position of } C\}$$

If R is a set of recipes we write $subterms_\mathsf{c}(R, S) = \bigcup_{\xi \in R} subterms_\mathsf{c}(\xi, S)$. From this we can define solutions of extended constraint systems.

Definition 4.4 (solution of an extended constraint system)
A pair of substitutions (Σ, σ) is a solution of $(\Phi, \mathsf{D}, \mathsf{E}^1, \mathsf{E}^2, \mathsf{K}, \mathsf{F})$ if $(\Phi, \Sigma, \sigma) \models \mathsf{D} \wedge \mathsf{E}^1 \wedge \mathsf{E}^2$ and the following two properties hold:
1 K-*Basis:* for all $\xi \in \mathit{subterms}^2(\mathit{img}(\Sigma) \cup \mathsf{K}\Sigma)$, $\mathsf{msg}(\xi\Phi\sigma)$ and $(\xi, \xi\Phi\sigma\downarrow) \in \mathsf{Conseq}(\mathsf{K}\Sigma\sigma)$
2 *Uniformity:* for all $\xi, \xi' \in \mathit{subterms}_{\mathsf{c}}(\mathit{img}(\Sigma), \mathsf{K}\Sigma)$, $\xi\Phi\sigma =_E \xi'\Phi\sigma$ implies $\xi = \xi'$.
The set of solutions of $\mathcal{C}^e$ is written $\mathit{Sol}(\mathcal{C}^e)$ and $\mathcal{C}^e$ is *satisfiable* if $\mathit{Sol}(\mathcal{C}^e) \neq \varnothing$. We will denote by $\bot$ an unsatisfiable extended constraint system. The notion of most general solution of $\mathcal{C}^e$ is adapted in a straightforward way from the analogue for regular constraint systems.

Intuitively when computing a node n of a partition tree, the extended constraint systems represent the predicate $\pi(n)$: it will be defined so that given $(\mathcal{P}, \mathcal{C}) \in \Gamma(n)$ attached with $\mathcal{C}^e$, we have $\mathit{Sol}^{\pi(n)}(\mathcal{C}) = \mathit{Sol}(\mathcal{C}^e)$ (up to domain restriction). We detail this in Sections 1.3 and 5.

▶ *Example 4.1*
Consider the extended constraint system $\mathcal{C}^e$ defined by

$$\Phi = \{\mathsf{ax}_1 \mapsto \langle k, x\rangle\} \qquad \mathsf{D} = X{:}0 \vdash^? x \wedge Y{:}1 \vdash^? y \qquad \mathsf{E}^1 = y =^? x \qquad \mathsf{E}^2 = \top$$

$$\mathsf{K} = \mathsf{ax}_1 \vdash^? \langle k, x\rangle \qquad \mathsf{F} = \mathsf{K} \wedge \mathsf{fst}(\mathsf{ax}_1) \vdash^? k \wedge \mathsf{snd}(\mathsf{ax}_1) \vdash^? x \wedge X =^?_f \mathsf{snd}(\mathsf{ax}_1)$$

This system involves an adversarial input x computable from an empty frame, which produces in response an output of $\langle k, x\rangle$ for some name k, and then the adversary inputs again $y = x$. The set F, although not impacting the notion of solution, characterises here all successful operations that the attacker may perform in this situation: applying destructors to the term bound to ax_1 and observe that X and $\mathsf{snd}(\mathsf{ax}_1)$ deduce the same term.

We have for example $(\langle X, \mathsf{ax}_1\rangle, \langle x, \langle k, x\rangle\rangle) \in \mathsf{Conseq}(\mathsf{K} \cup \mathsf{D})$. However the knowledge base is not *saturated* in the sense that there are deducible terms u, for example $u = k$, such that there exist no recipes ξ such that $(\xi, u) \in \mathsf{Conseq}(\mathsf{K} \cup \mathsf{D})$. In our procedure, the saturation is done by adding to K all destructor applications that result into a non-consequence term. A saturated version of the constraint system would be

$$\mathcal{C}^e_s = \mathcal{C}^e[\mathsf{K} \mapsto \mathsf{K} \wedge \mathsf{fst}(\mathsf{ax}_1) \vdash^? k]$$

Note that adding the deduction fact $\mathsf{snd}(\mathsf{ax}_1) \vdash^? x$ to the knowledge base is possible but redundant since x is already deducible from X. The saturation ensures that for all (Σ, σ) satisfying $\mathsf{D}(\mathcal{C}^e_s) \wedge \mathsf{E}^1(\mathcal{C}^e_s) \wedge \mathsf{E}^2(\mathcal{C}^e_s)$, there exists Σ' such that $(\Sigma', \sigma) \in \mathit{Sol}(\mathcal{C}^e_s)$, meaning that the requirement that solutions verify K-basis can always be satisfied (which is key for satisfying the requirement that all symbolic transitions are reflected in the partition tree, recall Item 2 of the definition in Chapter 3). Let us then consider

$$\Sigma = \{X \mapsto a, Y \mapsto \mathsf{snd}(\mathsf{ax}_1)\} \qquad \Sigma' = \{X \mapsto a, Y \mapsto a\} \qquad \sigma = \{x \mapsto a, y \mapsto a\}$$

Both (Σ, σ) and (Σ', σ) are solutions of the regular constraint system $(\Phi(\mathcal{C}^e_s), \mathsf{D}(\mathcal{C}^e_s), \mathsf{E}^1(\mathcal{C}^e_s))$, but only (Σ', σ) is a solution of $\mathcal{C}^e$. This is because Σ does not verify uniformity: two different recipes a and $\mathsf{snd}(\mathsf{ax}_1)$ are used to deduce the same first-order term a. More generally we have $\mathit{mgs}(\mathcal{C}^e_s) = \{Y \mapsto X\}$. To obtain this result, the constraint-solving procedure for computing mgs', detailed in Section 2, will observe that X and Y deduce the same term and should therefore be unified to satisfy uniformity. A second-order equation $X =^? Y$ is thus added in E^2, whose mgu is then the expected most general solution. ◀

▶ *Remark: uniformity and complexity*
In some sense enforcing that solutions are uniform ensures their minimality in terms of DAG size, by forcing identical recipes to be reused as much as possible when constructing the solution. This will be key for the complexity of our decision procedure, see Chapter 6. ◀

1.3 *Constraint solving*: the basics

Now we give details about the organisation of our constraint solver, detailed and proved correct in the next sections. As explained in Section 1.2, the goal of extended constraint system is to carry additional, structural information about solutions in a node n, thus playing the role of the predicate $\pi(n)$. More formally the procedure operates on:

Definition 4.5 (extended symbolic process, vector)
An *extended symbolic process* is a tuple $(\mathcal{P}, \mathcal{C}, \mathcal{C}^e)$ where $(\mathcal{P}, \mathcal{C})$ is a symbolic process and $\mathcal{C}^e$ an extended constraint system. We call a *vector* a set of sets of extended symbolic processes $\mathbb{S} = \{\Gamma_1, \ldots, \Gamma_n\}$. Each set Γ_i is called a *component* of $\mathbb{S}$.

An extended symbolic process $(\mathcal{P}, \mathcal{C}, \mathcal{C}^e)$ induces a predicate π on the solutions of $(\mathcal{P}, \mathcal{C})$ defined as follows: if $(\Sigma, \sigma) \in Sol(\mathcal{C})$, then $\pi(\Sigma)$ holds *iff* there exists $(\Sigma', \sigma') \in Sol(\mathcal{C}^e)$ such that $\Sigma \subseteq \Sigma'$ and $\sigma \subseteq \sigma'$. In particular $Sol^{\pi}(\mathcal{C}) = Sol(\mathcal{C}^e)$ (up to domain restriction), but this predicate may differ from one extended symbolic process to another. However, assuming a set Γ of such processes where this predicate π is uniform across all elements of Γ, this may be used to model a partition-tree node (up to the additional properties required by the definition). Given such a set Γ modelling a node n, the goal of the constraint-solving procedure is therefore to refine Γ until obtaining a vector $\mathbb{S} = \{\Gamma_1, \ldots, \Gamma_n\}$ such that

1. each component Γ_i can be used to model a partition-tree node, that is, a predicate π_i can be defined as above uniformly across all elements of Γ_i;
2. the underlying nodes verify the properties of the partition tree w.r.t. their father node Γ.

The procedure takes the form of various reduction relations that are used to refine a set of sets of extended symbolic processes, progressively, until reaching the final vector $\mathbb{S}$:

1. A set of rules to compute *most general solutions* (Section 2).
2. A set of *symbolic rules* (Section 3.1) that formalise how to apply symbolic transitions to extended symbolic processes.
3. Various sets of *simplification rules* (Sections 2.3, 3.2 and 3.3) that simplify vectors to remove unsatisfiable systems, or to split components that contain processes with non-statically-equivalent solutions.
4. A set of *case distinction rules* (Section 4) that refines the current vector based on case analyses to enforce the various properties of the partition tree (unique mgs in each component, maximal components w.r.t. static equivalence...).

The overall procedure organising the above sets of rules into a complete algorithm to compute a partition tree is then detailed in Section 5. This is therefore the detailed version of the outline of Chapter 3, Section 3.1. The main arguments for proving the correctness of the computation are also provided in Section 5; note however that these are only partial-correctness arguments in that the termination of the procedure is studied in Chapter 6.

2 *Constraint solving*: computing most general solutions

Section summary

We describe a *constraint solving* procedure permitting to compute the most general solution(s) of an extended constraint system. Due to the particular structure the solutions are required to have (K-basis and uniformity) the procedure becomes rather lightweight.

2.1 Applying solutions and unifiers

Because solutions Σ may introduce new second-order variables, their applications to a constraint system or a formula is not straightforward. Let for example $\mathcal{C}^e = (\Phi, \mathsf{D}, \mathsf{E}^1, \mathsf{E}^2, \mathsf{K}, \mathsf{F})$ where a variable $X{:}k$ is used to deduce a term u, i.e. $(X \vdash^? u) \in \mathsf{D}$. Now say we want consider the scenario where u is computed using a constructor $\mathsf{f}/3$ and an entry of the knowledge base $(\xi \vdash^? v) \in \mathsf{K}$ as a first argument, that is, we want to apply to $\mathcal{C}^e$:

$$\Sigma = \{X \to \mathsf{f}(\xi, X_1, X_2)\} \qquad\qquad X_1, X_2 \text{ fresh}$$

The raw application $\mathcal{C}^e\Sigma$ has a flawed structure, in particular because the variables X_1 and X_2 would not be bound in the resulting system. To solve this issue we use a custom application mechanism that replaces $X \vdash^? u$ in D by $X_1 \vdash^? x_1, X_2 \vdash^? x_2$, x_1, x_2 fresh, and we add the equality $u =^? \mathsf{f}(v, x_1, x_2)$ to E^1 to express the logical link between X and $X\Sigma = \mathsf{f}(\xi, X_1, X_2)$.

Definition 4.6 (application of a substitution to an extended constraint system)
Let $\mathcal{C}^e = (\Phi, \mathsf{D}, \mathsf{E}^1, \mathsf{E}^2, \mathsf{K}, \mathsf{F})$ and Σ be a substitution. We write $\mathcal{C}^e{:}\Sigma$ the constraint system:

$$(\Phi, \mathsf{D}', \mathsf{E}^1 \wedge E_\Sigma, \mathsf{E}^2\Sigma \wedge \Sigma_{|vars^2(\mathcal{C}^e)}, \mathsf{K}\Sigma, \mathsf{F}\Sigma)$$

where $\mathsf{D}' = (\mathsf{D} \setminus D_{dom}) \cup D_{fresh}$ with the sets of:
1 deduction facts removed by the application of Σ: $D_{dom} = \{Y \vdash^? u \in \mathsf{D} \mid Y \in dom(\Sigma)\}$
2 binding facts: $D_{fresh} = \{Y \vdash^? y \mid Y \in vars^2(img(\Sigma_{|vars^2(\mathcal{C}^e)})) \setminus vars^2(\mathcal{C}^e), y \text{ fresh}\}$
3 linking equations: $E_\Sigma = \{u =^? v \mid Y \vdash^? u \in D_{dom}, (Y\Sigma, v) \in \mathsf{Conseq}(\mathsf{K}\Sigma \cup \mathsf{D}')\}$
By abuse of notation we may write $S{:}\Sigma$ for $(\mathcal{P}, \mathcal{C}, \mathcal{C}^e{:}\Sigma)$ if $S = (\mathcal{P}, \mathcal{C}, \mathcal{C}^e)$.

We will also use a similar mechanism for applying substitutions to formulas:

Definition 4.7 (application of a substitution to a formula)
Let $\mathcal{C}^e = (\Phi, \mathsf{D}, \mathsf{E}^1, \mathsf{E}^2, \mathsf{K}, \mathsf{F})$, $\psi = \forall S.\, \varphi \Rightarrow H$ be a formula, and Σ be a substitution. We denote $\psi{:}(\Sigma, \mathcal{C}^e)$ (or $\psi{:}(\Sigma, S)$ by abuse of notations if $S = (\mathcal{P}, \mathcal{C}, \mathcal{C}^e)$) the formula

$$\forall S'.\, (\mathsf{D}' \wedge \mathsf{hyp}(\psi) \wedge E_\Sigma) \Rightarrow H\Sigma$$

where $\mathsf{D}' = (\mathsf{D}(\psi) \setminus D_{dom}) \cup D_{fresh}$, $S' = (S \setminus dom(\Sigma)) \cup vars^1(D_{fresh})$ and:
1 $D_{dom} = \{Y \vdash^? u \in \mathsf{D}(\psi) \mid Y \in dom(\Sigma)\}$
2 $D_{fresh} = \{Y \vdash^? y \mid Y \in vars^2(img(\Sigma)) \setminus vars^2(\mathcal{C}, \psi), y \text{ fresh}\}$
3 $E_\Sigma = \{u =^? v \mid Y \vdash^? u \in D_{dom}, (Y\Sigma, v) \in \mathsf{Conseq}\,(\mathsf{K} \cup \mathsf{D} \cup \mathsf{D}')\}$

2.2 Constraint-solving rules

A complete example By definition, the solutions of an extended constraint systems $\mathcal{C}^e$ have to verify $\mathsf{K}(\mathcal{C}^e)$-basis, which means that in practice we only have to compute solutions constructed by applying constructors to the entries of the knowledge base and D. Besides due to the uniformity requirement we can always unify two recipes that deduce the same first-order term. Putting everything together the most general solutions of an extended constraint system can then be computed with a simple transition system. Let us detail a complete example to illustrate the mechanisms in play, before formalising the corresponding constraint-solving rules.

▶ *Example 4.2*

Given $k, r \in \mathcal{N}$, let us consider a situation where the attacker has observed the output of a hash $\mathsf{h}(r)$, then inputs a term x, receives in response a ciphertext $\mathsf{raenc}(k, r, x)$ encrypted with x, and finally inputs a term y that should verify the equation $y =^? \langle k, \mathsf{h}(x)\rangle$. This is modelled by the frame $\Phi = \{\mathsf{ax}_1 \mapsto \mathsf{h}(r), \mathsf{ax}_2 \mapsto \mathsf{raenc}(k, r, x)\}$ and the constraints

$$\mathsf{D} = X{:}1 \vdash^? x \wedge Y{:}2 \vdash^? y \qquad\qquad \mathsf{E}^1 = y =^? \langle k, \mathsf{h}(x)\rangle$$

At this point a saturated knowledge base should contain the two entries of the frame and one recipe indicating that decrypting ax_2 results in obtaining the name k.

$$\mathsf{K} = \mathsf{ax}_1 \vdash^? \mathsf{h}(r) \wedge \mathsf{ax}_2 \vdash^? \mathsf{raenc}(k, r, x) \wedge \mathsf{radec}(\mathsf{ax}_2, X) \vdash^? k$$

We consider that $\mathsf{E}^2 = \top$ and we leave the set of formulas F unspecified since it has no influence on solutions. First of all some *simplification rules* will be applied to propagate the equations on x and y to the whole system; here it will apply $mgu(\mathsf{E}^1)$ to D, resulting in

$$\mathsf{D} = X \vdash^? x \wedge Y \vdash^? \langle k, \mathsf{h}(x)\rangle$$

The constraint-solving rules detailed in the remaining of this section consider all ways to compute recipes for X and Y from the knowledge base. For each of these recipes two cases arise: either 1 it is picked directly from the knowledge base; or 2 it starts with a constructor symbol. This will correspond to the constraint-solving rules (MGS-Res) and (MGS-Cons), respectively. Finally, to satisfy the uniformity property, the procedure unifies any second-order terms in the system that deduce the same first-order term (Rule (MGS-Conseq)). We keep on refining the case analysis with these three rules, removing branches yielding contradictions, until no more rules are applicable. The resulting systems will either have no solutions, or be in a so-called *solved form* and have $mgu(\mathsf{E}^2)$ as a unique mgs. Let us do it for our example:

▷ *case 1: the recipe for Y has a constructor symbol at its root (only possible case)*

The constructor in question is necessarily the pair. Therefore we let two fresh second-order variables $Y_1{:}2, Y_2{:}2$ and apply the substitution $\{Y \mapsto \langle Y_1, Y_2\rangle\}$ to the system (in the sense of Definition 4.6). After simplification this leads to the updated second-order constraints:

$$\mathsf{D} = X \vdash^? x \wedge Y_1 \vdash^? k \wedge Y_2 \vdash^? \mathsf{h}(x) \qquad\qquad \mathsf{E}^2 = Y =^? \langle Y_1, Y_2\rangle$$

▷ *case 1.1: the recipe for Y_1 is $\mathsf{radec}(\mathsf{ax}_2, X)$ from the knowledge base (only possible case)*

We thus apply the substitution $\{Y_1 \mapsto \mathsf{radec}(\mathsf{ax}_2, X)\}$, resulting in the updated constraints:

$$\mathsf{D} = X \vdash^? x \wedge Y_2 \vdash^? \mathsf{h}(x) \qquad \mathsf{E}^2 = Y =^? \langle \mathsf{radec}(\mathsf{ax}_2, X), Y_2\rangle \wedge Y_1 =^? \mathsf{radec}(\mathsf{ax}_2, X)$$

▷ *case 1.1.1: the recipe for Y_2 is the entry ax_1 from the knowledge base*

We therefore apply the substitution $\{Y_2 \mapsto \mathsf{ax}_1\}$, resulting in the updated constraints:

$$\mathsf{D} = X \vdash^? r \qquad \mathsf{E}^2 = Y =^? \langle \mathsf{radec}(\mathsf{ax}_2, X), \mathsf{ax}_1 \rangle \wedge Y_1 =^? \mathsf{radec}(\mathsf{ax}_2, X) \wedge Y_2 =^? \mathsf{ax}_1$$

However the constraints on X are now unsatisfiable: the corresponding recipe can neither start with a constructor nor be an entry of the knowledge base. The constraints in this branch of the case analysis therefore have no solutions.

▷ *case 1.1.2: the recipe for Y_2 has a constructor symbol at its root*

The constructor in question is necessarily h. Similarly to case 1 we apply the substitution $\{Y_2 \mapsto \mathsf{h}(Y_3)\}$ for some fresh variable Y_3:2 which results in the updated constraints:

$$\mathsf{D} = X \vdash^? x \wedge Y_3 \vdash^? x \quad \mathsf{E}^2 = Y =^? \langle \mathsf{radec}(\mathsf{ax}_2, X), \mathsf{h}(Y_3) \rangle \wedge Y_1 =^? \mathsf{radec}(\mathsf{ax}_2, X) \wedge Y_2 =^? \mathsf{h}(Y_3)$$

Then we observe that X and Y_3 should be unified by uniformity because they deduce the same first-order term x. We have $mgu(X =^? Y_3) = \{Y_3 \mapsto X\}$ (we recall that $\{X \mapsto Y_3\}$ is *not* a valid second-order substitution because Y_3 has a strictly greater type than X) which, after application to the system, results in the updated constraints:

$$\mathsf{D} = X \vdash^? x \quad \mathsf{E}^2 = Y =^? \langle \mathsf{radec}(\mathsf{ax}_2, X), \mathsf{h}(X) \rangle \wedge Y_1 =^? \mathsf{radec}(\mathsf{ax}_2, X) \wedge Y_2 =^? \mathsf{h}(X) \wedge Y_3 =^? X$$

This will be a typical example of system in solved form. Since we considered all cases and only this branch was successful we conclude that the overall system has a unique mgs which is $mgu(\mathsf{E}^2) = \{Y \mapsto \langle \mathsf{radec}(\mathsf{ax}_2, X), \mathsf{h}(X) \rangle\}$. ◀

Formalisation We will formalise the simplification rules in the next section and focus here on the main three rules (MGS-Conseq), (MGS-Res) and (MGS-Cons) mentioned in the above example. For that we reason about a set $used^2(\mathcal{C}^e)$ that represents all recipes that are already used to constraint the solutions of $\mathcal{C}^e$:

$$used^2(\mathcal{C}^e) = subterms_{\mathsf{c}}(img(mgu(\mathsf{E}^2(\mathcal{C}^e)), \mathsf{K}(\mathcal{C}^e) \cup \mathsf{D}(\mathcal{C}^e)) \cup vars^2(\mathsf{D}(\mathcal{C}^e))$$

As we saw in the example, the mgs is gradually constructed "within E^2", in the sense that after normalising $\mathcal{C}^e$ with the transition system defined in this section, it will have $mgu(\mathsf{E}^2)$ as a unique mgs. In particular an invariant of our transition system is that $img(mgu(\mathsf{E}^2(\mathcal{C}^e)))$ is consequence of $\mathsf{K}(\mathcal{C}^e)$ and $\mathsf{D}(\mathcal{C}^e)$, hence the notation $used^2(\mathcal{C}^e)$ is well defined. Formally speaking the transition system relies on three rules of the form

$$\mathcal{C}^e \xrightarrow{\Sigma} \mathcal{C}^e{:}\Sigma \qquad (\star)$$

for some subtitution Σ and under various conditions capturing the possible ways to satisfy the constraints of $\mathcal{C}^e$. For example the uniformity property is expressed by applying $(\star)$ with

$$\Sigma = mgu(\xi =^? \zeta) \text{ for some } \xi \in used^2(\mathcal{C}^e) \cup \mathcal{F}_0,\ \zeta \in used^2(\mathcal{C}^e), \text{ and provided } \Sigma \neq \top,\ \Sigma \neq \bot, \text{ and } \exists u.(\xi, u), (\zeta, u) \in \mathsf{Conseq}(\mathsf{K}(\mathcal{C}^e) \cup \mathsf{D}(\mathcal{C}^e)) \qquad \text{(MGS-Conseq)}$$

The result is the unification in $\mathcal{C}^e$ of the two second-order terms ξ and ζ that deduce the same term u. It then remains to add rules that express how each term u, $(X \vdash^? u) \in \mathsf{D}(\mathcal{C}^e)$, can be

constructed by the adversary from the knowledge base. When Rule (MGS-Conseq) is not applicable we thus apply ($\star$) under one of the following two conditions. The first one expresses that u is computed by directly using an entry from the knowledge base:

$\Sigma = mgu(X =^? \xi) \neq \bot$ where, for some $u \notin \mathcal{X}$, there exist deduction facts $(X{:}k \vdash^? u) \in \mathsf{D}(\mathcal{C}^e)$ and $(\xi \vdash^? v) \in \mathsf{K}(\mathcal{C}^e)$ (MGS-Res)

Then the last rule expresses that the computation of u starts by applying a constructor f:

$\Sigma = \{X \to \mathsf{f}(X_1, \ldots, X_n)\}$ where $X_1{:}k, \ldots, X_n{:}k$ are fresh, and there exists a deduction fact $(X{:}k \vdash^? \mathsf{f}(u_1, \ldots, u_n)) \in \mathsf{D}(\mathcal{C}^e)$ (MGS-Cons)

As said above we always apply Rule (MGS-Conseq) in priority, that is, we add to the last two rules the condition that Rule (MGS-Conseq) cannot be applied. This will be crucial in particular when studying the complexity of the procedure in Chapter 6.

2.3 First set of simplification rules

Between each application of the above three rules, the constraint systems are then simplified by the following set of *simplification rules*. Other simplification rules serving different purposes will be introduced in the remaining of the procedure. The rules here are separated in two kinds: *simplification rules for formulas* that simply compute most general unifiers and simplify the hypotheses of formulas, and *simplification rules for mgs'* that apply the unifiers computed by above rules to the rest of the system and detect contradictions and violations of uniformity.

Simplification rules for formulas We first introduce basic simplification rules for formulas that will be used even outside of the computation of most general solutions. We define five sets rules in Figure 4.1 that apply on constraints of E^1, E^2 and F.

Misc. $\quad \neg\top \rightsquigarrow \bot \qquad \neg\bot \rightsquigarrow \top \qquad \varphi \wedge \top \rightsquigarrow \varphi \qquad \varphi \wedge \bot \rightsquigarrow \bot$

Universal vars. $\quad \forall S \cup \{x\}.\, (x =^? u \wedge \varphi) \Rightarrow H \rightsquigarrow \forall S.\, \varphi\sigma \Rightarrow H\sigma$ if $\sigma = mgu(x =^? u) \neq \bot$
$\forall S \cup \{x\}.\, \varphi \Rightarrow H \rightsquigarrow \forall S.\, \varphi \Rightarrow H$ if $x \notin vars^1(\varphi)$

1st order eq. $\quad u =^? v \rightsquigarrow mgu(u =^? v)$

1st order diseq. $\quad \forall S.\, \phi \rightsquigarrow \begin{cases} \forall S.\ \bigvee_{x \in dom(\sigma)} x \neq^? x\sigma & \text{with } \sigma = mgu(\neg\phi) \neq \bot \\ \top & \text{if } mgu(\neg\phi) = \bot \end{cases}$

2nd order diseq. $\quad \forall S.\, \phi \rightsquigarrow \begin{cases} \forall S \cup S'.\ \bigvee_{X \in dom(\Sigma)} X \neq^? X\Sigma & \text{with } \Sigma = mgu(\neg\phi) \neq \bot \\ & \text{and } S' = vars^2(img(\Sigma)) \setminus vars^2(\phi) \\ \top & \text{if } mgu(\neg\phi) = \bot \end{cases}$

Figure 4.1 Simplification rules on formulae

No rules are needed for second-order equations in the context of our decision procedure, since Rules (MGS-Conseq), (MGS-Res) and (MGS-Cons) already apply mgu's to the entire system. The simplification rules are lifted to extended constraint systems $\mathcal{C}^e$ in the natural way, by applying the simplifications to all formulas of $\mathsf{E}^1(\mathcal{C}^e)$, $\mathsf{E}^2(\mathcal{C}^e)$ and $\mathsf{F}(\mathcal{C}^e)$.

Simplification rules for MGS In addition of the rules of Figure 4.1 we define a couple of other rules specific to the computation of most general solutions. First of all the rule

$$(\Phi, \mathsf{D}, \mathsf{E}^1 \wedge x =^? u, \mathsf{E}^2, \mathsf{K}, \mathsf{F}) \quad \rightsquigarrow \quad (\Phi\sigma, \mathsf{D}\sigma, \mathsf{E}^1\sigma \wedge x =^? u, \mathsf{E}^2, \mathsf{K}\sigma, \mathsf{F}\sigma) \qquad \text{(MGS-Unif)}$$

where $x \in vars^1(\mathsf{E}^1, \mathsf{D}, \Phi, \mathsf{K}, \mathsf{F}) \setminus vars(u)$ and $\sigma = \{x \mapsto u\}$, propagates first-order mgu's in the whole system. We also consider the following rule discarding a system with no solutions

$$\mathcal{C}^e \rightsquigarrow \bot \qquad \text{(MGS-Unsat)}$$

where either of the following three conditions is satisfied:

1. $\mathsf{E}^1 = \bot$
2. there exist $\xi, \zeta \in used^2(\mathcal{C}^e)$ such that $(\xi, u), (\zeta, u) \in \mathsf{Conseq}(\mathsf{K}(\mathcal{C}^e) \cup \mathsf{D}(\mathcal{C}^e))$ and, writing $\Sigma = mgu(\xi =^? \zeta)$, either $\Sigma = \bot$ or $\mathsf{E}^2\Sigma \rightsquigarrow^* \bot$ with the rules of Figure 4.1
3. there exist $(\forall X{:}i.\, X \not\vdash^? u) \in \mathsf{D}(\mathcal{C}^e)$ and $\xi \in \mathcal{T}_i^2$ such that $(\xi, u) \in \mathsf{Conseq}(\mathsf{K}(\mathcal{C}^e) \cup \mathsf{D}(\mathcal{C}^e))$

The first condition captures trivially unsatisfiable systems, the second one systems with no uniform solutions, and the third one exhibits a public channel that has been used for an internal communication (which is forbidden by the semantics). Since the whole set of simplification rules (Figure 4.1 and the above two) is convergent modulo renaming of variables, we denote $\mathcal{C}\!\downarrow$ a normal form of the extended constraint system $\mathcal{C}$ w.r.t. $\rightsquigarrow$.

2.4 Overall procedure and correctness

Description of the procedure The point of the transition systems above is to transform an extended constraint system into a form where it has a unique mgs. More formally:

Definition 4.8 (solved extended constraint system)
An extended constraint system $\mathcal{C}^e$ is in *solved form* if $\mathcal{C}^e \neq \bot$, $\mathcal{C}^e$ is irreducible w.r.t. $\rightsquigarrow$ and $\rightarrow\!\downarrow$, and all deduction facts in $\mathsf{D}(\mathcal{C}^e)$ have variables as first-order terms.

Intuitively for such constraint systems, $mgu(\mathsf{E}^2(\mathcal{C}^e))$ is the unique mgs of $\mathcal{C}^e$. Note however that this method for computing mgs' is only correct under some invariants of Algorithm 1. Typically, since second-order equations are not handled by simplification rules, if E^2 contains two equations $X =^? a$ and $X =^? b$ for two constants $a \neq b$, our procedure would fail to detect the contradiction. If we define the reduction relations $\xrightarrow{\Sigma}\!\downarrow$ and $\overset{\Sigma}{\Longrightarrow}\!\downarrow$ by the inference rules

$$\frac{\mathcal{C}_1^e \xrightarrow{\Sigma} \mathcal{C}^e \qquad \mathcal{C}^e\!\downarrow = \mathcal{C}_2^e}{\mathcal{C}_1^e \xrightarrow{\Sigma}\!\downarrow \mathcal{C}_2^e} \qquad \frac{}{\mathcal{C}^e \overset{id}{\Longrightarrow}\!\downarrow \mathcal{C}^e} \qquad \frac{\mathcal{C}_1^e \overset{\Sigma}{\Longrightarrow}\!\downarrow \mathcal{C}_2^e \qquad \mathcal{C}_2^e \xrightarrow{\Sigma'}\!\downarrow \mathcal{C}_3^e}{\mathcal{C}_1^e \overset{\Sigma\Sigma'}{\Longrightarrow}\!\downarrow \mathcal{C}_3^e}$$

then under the invariants of the procedure we compute a set of most general solutions of $\mathcal{C}^e$ as the set $\{\Sigma_{|vars^2(\mathcal{C}^e)} \mid \mathcal{C}^e \overset{\Sigma}{\Longrightarrow}\!\downarrow \mathcal{C}^{e\prime}, \mathcal{C}^{e\prime} \text{ solved}\}$.

▶ *Remark: notation for extended symbolic processes*
For convenience we often abuse notations and, if $S = (\mathcal{P}, \mathcal{C}, \mathcal{C}^e)$ is an extended symbolic process, we write $mgs(S)$ instead of $mgs(\mathcal{C}^e)$ or say that S is in solved form. ◀

Correctness arguments As mentioned earlier this procedure is only correct under some additional properties verified all along Algorithm 1. For the sake of precision we make explicit mention to these two invariants, $\mathsf{Inv}_{wf}(\mathcal{C}^e)$ and $\mathsf{Inv}_{sound}(\mathcal{C}^e)$. They are formally defined in Appendix B, Section 1 with a proof that they are preserved during the whole computation of the partition tree, but knowing their exact definition is not necessary to understand the results of this section. The core correctness arguments can be decomposed into following propositions, proved in Appendix B. The first one states that when an extended constraint system cannot be reduced anymore then its set of most general solutions is either empty or a singleton:

Proposition 4.1 (mgs of an irreducible system)
Let $\mathcal{C}^e$ be an extended constraint system that is irreducible w.r.t. $\rightsquigarrow$ and $\rightarrow\!\!\!\!\downarrow$, and such that the invariants $\mathsf{Inv}_{wf}(\mathcal{C}^e)$ and $\mathsf{Inv}_{sound}(\mathcal{C}^e)$ hold. Then
1 if $\mathcal{C}^e$ is in solved form then $mgs(\mathcal{C}^e) = \{mgu(\mathsf{E}^2(\mathcal{C}^e))\}$
2 otherwise $mgs(\mathcal{C}^e) = \varnothing$

The second argument is that applying the mgs constraint-solving rules is correct w.r.t. the solutions of the initial system.

Proposition 4.2 (soundness of one step of the mgs constraint solving)
Let $\mathcal{C}^e$ be an extended constraint system such that $\mathcal{C}^e = \mathcal{C}^e\!\downarrow$. If $\mathcal{C}^e \xrightarrow{\Sigma} \!\!\downarrow \mathcal{C}^{e\prime}$ and $(\Sigma, \sigma) \in Sol(\mathcal{C}^{e\prime})$ then $(\Sigma_{|vars^2(\mathcal{C}^e)}, \sigma_{|vars^1(\mathcal{C}^e)}) \in Sol(\mathcal{C}^e)$.

Finally the last argument formalises than all solutions can be expressed as a sequence of mgs constraint-solving transitions.

Proposition 4.3 (completeness of one step of the mgs constraint solving)
Let $\mathcal{C}^e$ be an extended constraint system such that $\mathcal{C}^e\!\downarrow = \mathcal{C}^e$ and the invariants $\mathsf{Inv}_{wf}(\mathcal{C}^e)$ and $\mathsf{Inv}_{sound}(\mathcal{C}^e)$ hold. We also assume that at least one mgs constraint-solving rule is applicable to $\mathcal{C}^e$. Then for all $(\Sigma, \sigma) \in Sol(\mathcal{C}^e)$, there exist a constraint-solving transition $\mathcal{C}^e \xrightarrow{\Sigma_0} \!\!\downarrow \mathcal{C}^{e\prime}$ and $\Sigma \subseteq \Sigma'$, $\sigma \subseteq \sigma'$ such that $(\Sigma', \sigma') \in Sol(\mathcal{C}^{e\prime})$.

Together these three results give the partial correctness of the procedure, that is, the correctness of the computation when it terminates. The termination is studied in Chapter 6:

Theorem 4.4 *(partial correctness of mgs computation)*
Let $\mathcal{C}^e$ be an extended constraint system such that $\mathsf{Inv}_{wf}(\mathcal{C}^e)$ and $\mathsf{Inv}_{sound}(\mathcal{C}^e)$ hold. Then, assuming there exist no infinite sequences of $\rightarrow\!\!\downarrow$ reductions from $\mathcal{C}^e$, we have

$$mgs(\mathcal{C}^e) = \{\Sigma_{|vars^2(\mathcal{C}^e)} \mid \mathcal{C}^e\!\downarrow \overset{\Sigma}{\Rightarrow}\!\!\downarrow \mathcal{C}^{e\prime}, \mathcal{C}^{e\prime} \text{ solved}\}$$

Proof. Since a set of mgs' of $\mathcal{C}^e\{$ is also a set of mgs' of $\mathcal{C}^e$, we assume without loss of generality that $\mathcal{C}^e\{ = \mathcal{C}^e$. Let us write $S = \{\Sigma_{|vars^2(\mathcal{C}^e)} \mid \mathcal{C}^e \overset{\Sigma}{\Rightarrow}\{ \mathcal{C}^{e\prime}, \mathcal{C}^{e\prime} \text{ solved}\}$ and prove that S is a set of mgs' of $\mathcal{C}^e\{$. By the termination assumption, we can reason by well-founded induction on the reduction relation $\to\{$ from $\mathcal{C}^e$. Using such an induction we can prove the two requirements of the definition, that is:

1. that all $\Sigma \in S$ are solutions of $\mathcal{C}^e$ after replacing their second-order variables by fresh constants (base case: Proposition 4.1; inductive case: soundness, i.e., Proposition 4.2).
2. that all solutions of $\mathcal{C}^e$ are instances of a substitution of S (base case: Proposition 4.1 again; inductive case: completeness, i.e., Proposition 4.3). □

3 *Constraint solving*: symbolic and simplification rules

Section summary

We now introduce: *symbolic rules* that apply symbolic transitions to extended symbolic processes and collect the corresponding constraints, *normalisation rules* that simplify the resulting constraints based on the analysis of mgs', and *vector-simplification rules* that separate systems with non-equivalent solutions based on the analysis of the formulas in F among other things.

3.1 Symbolic rules

The symbolic rules simply apply the transitions of the symbolic semantics to extended symbolic processes, adding the corresponding constraints to both the symbolic process and the extended constraint system. In that sense most rules are close to identical to those of the symbolic semantics (Chapter 3, Section 1.4). Typically the analogue of the rule (S-In) is:

$$(\{\!\{u(x).P\}\!\} \cup \mathcal{P}, \mathcal{C}, \mathcal{C}^e) \xrightarrow{Y(X)}_{\mathsf{s}} (\{\!\{P\}\!\} \cup \mathcal{P}, incr(\mathcal{C}), incr(\mathcal{C}^e)) \qquad \text{(E-In)}$$

where, if $\mathcal{D} \in \{\mathcal{C}, \mathcal{C}^e\}$, $incr(\mathcal{D}) = \mathcal{D}[\mathsf{D} \mapsto \mathsf{D} \wedge X \vdash^? x \wedge Y \vdash^? y, \mathsf{E}^1 \mapsto \mathsf{E}^1 \wedge \sigma]$ with $Y{:}n$, $X{:}n$ and y fresh and $\sigma \in mgu_E(y =^? u\mu)$, $\mu = mgu(\mathsf{E}^1(\mathcal{C}))$. The only rule that is not a trivial extension of the symbolic semantics is the one for outputs that puts a deduction fact in F to model the additional capability this offers to the attacker:

$$(\{\!\{\overline{u}\langle v\rangle.P\}\!\} \cup \mathcal{P}, \mathcal{C}, \mathcal{C}^e) \xrightarrow{\overline{Y}\langle \mathsf{ax}_{n+1}\rangle}_{\mathsf{s}} (\{\!\{P\}\!\} \cup \mathcal{P}, incr(\mathcal{C}), incr(\mathcal{C}^e)[\mathsf{F} \mapsto \mathsf{F} \wedge \mathsf{ax}_{n+1} \vdash^? v\sigma\downarrow]) \quad \text{(E-Out)}$$

where, if $\mathcal{D} \in \{\mathcal{C}, \mathcal{C}^e\}$, $incr(\mathcal{D}) = \mathcal{D}[\Phi \mapsto \Phi \cup \{\mathsf{ax}_{n+1} \mapsto v\sigma\downarrow\}, \mathsf{D} \mapsto \mathsf{D} \wedge Y \vdash^? y, \mathsf{E}^1 \mapsto \mathsf{E}^1 \wedge \sigma]$ with $Y{:}n$ and y fresh and $\sigma \in mgu_E(y =^? u\mu \wedge v\mu =^? v\mu)$, $\mu = mgu(\mathsf{E}^1(\mathcal{C}))$. We omit the definition of the remaining rules corresponding to the other symbolic transitions, all constructed being constructed similarly to (E-In) by copying the new constraints of $\mathcal{C}$ into $\mathcal{C}^e$.

3.2 Normalisation rules

We define a new set of simplification rules, called *normalisation rules*, that operate on extended constraint systems. Similarly to the simplification rules for most general solutions introduced in Section 2.3 they propagate first-order unifiers across the system and replace unsatisfiable systems by $\bot$. They also rely on the computation of mgs' of Section 2, for example to identify

$$\mathcal{C}^e \rightsquigarrow \mathcal{C}^{e\prime} \quad \text{if } \mathcal{C}^e \rightsquigarrow \mathcal{C}^{e\prime} \text{ by rule (MGS-Unif)} \qquad \text{(Norm-Unif)}$$

$$\mathcal{C}^e \rightsquigarrow \bot \quad \text{if } mgs(\mathcal{C}^e) = \varnothing \qquad \text{(Norm-no-MGS)}$$

$$\mathcal{C}^e[\mathsf{E}^1 \mapsto \mathsf{E}^1 \wedge \forall \tilde{x}.\phi] \rightsquigarrow \mathcal{C}^e \quad \text{if } mgs(\mathcal{C}^e[\mathsf{E}^1 \mapsto \mathsf{E}^1 \wedge \neg\phi]) = \varnothing \qquad \text{(Norm-Diseq)}$$

$$\mathcal{C}^e[\mathsf{F} \mapsto \mathsf{F} \wedge \psi] \rightsquigarrow \mathcal{C}^e \quad \text{if } mgs(\mathcal{C}^e[\mathsf{E}^1 \mapsto \mathsf{E}^1 \wedge \mathsf{hyp}(\psi)]) = \varnothing \qquad \text{(Norm-Formula)}$$

$$\mathcal{C}^e[\mathsf{F} \mapsto \mathsf{F} \wedge \psi] \rightsquigarrow \mathcal{C}^e \quad \text{if } \exists \psi' \in \mathsf{F},\ \psi' \simeq_{\mathsf{h}} \psi \text{ and } \psi' \text{ solved} \qquad \text{(Norm-Dupl)}$$

Figure 4.2 Normalisation rules on extended constraint systems

and remove trivial constraints such as formulas with unsatisfiable hypotheses. They are defined in Figure 4.2 and commented below (in particular regarding the definition of $\simeq_{\mathsf{h}}$).

We recall that we also write $\mathcal{C}^e \rightsquigarrow \mathcal{C}^{e\prime}$ if a constraint of $\mathsf{E}^1(\mathcal{C}^e)$, $\mathsf{E}^2(\mathcal{C}^e)$ or $\mathsf{F}(\mathcal{C}^e)$ can be simplified using one of the simplification rules on formulas (Figure 4.1). The relation $\rightsquigarrow$ can be lifted to sets of (sets of) extended constraint systems or symbolic processes in the natural way. Let us now comment on the rules of Figure 4.2. Rule (Norm-Unif) uses the same rule as in the mgs constraint solving to propagate first-order unifiers to the whole system. The next three rules exploit the existence of a most general solution of the constraint system to simplify some constraints:

1. Rule (Norm-no-MGS) checks whether the constraint system is unsatisfiable, i.e., does not have a most general solution, and in this case transforms it into $\bot$.
2. Rule (Norm-Diseq) similarly removes a disequation $\forall \tilde{x}.\phi$ in E^1 when it does not effectively restrict the solutions: for that we require the constraint system not to have solutions that contradict the disequation.
3. Analoguously Rule (Norm-Formula) removes a formula with unsatisfiable hypotheses.

Following the same approach as the last two rules, one could imagine a rule removing a non-deducibility constraint $\forall X.\, X \not\vdash^? u$ from D provided $mgs(\mathcal{C}^e[\mathsf{D} \mapsto \mathsf{D} \wedge X \vdash^? u]) = \varnothing$. This would however not be sound in general since the knowledge base may not be saturated, that is, there may exist a solution satisfying $X \vdash^? u$ due to an entry added to $\mathsf{K}(\mathcal{C}^e)$ in the future. The treatment of non-deducibility facts will be handled by a dedicated case-distinction rule, see Section 4.4. Finally Rule (Norm-Dupl) removes an unsolved deduction or equality formula ψ from F when it is subsumed by another formula ψ'. This is formalised by the following notion of equivalence:

Definition 4.9 (head equivalence of formulas)

Let $\psi = \varphi \Rightarrow H$ and $\psi' = \varphi' \Rightarrow H'$ be two formulas. We say that ψ and ψ' are *head equivalent*, written $\psi \simeq_{\mathsf{h}} \psi'$, if for some ξ, ζ, u, u' either $H = H' = (\xi =^?_f \zeta)$, or $H = (\xi \vdash^? u)$ and $H' = (\xi \vdash^? u')$.

That is, two formulas are head equivalent if their heads have the same second-order terms (but may differ on their first-order terms), which means they model the same attacker action. In particular if $\psi \simeq_{\mathsf{h}} \psi'$ and ψ' is solved (namely has no hypotheses anymore) then the formula ψ is already implied by ψ' which is why Rule (Norm-Dupl) can remove it from F.

3.3 Vector-simplification rules

We now define simplification rules that focus on vector, thus called *vector-simplification rules*. They are described in Figure 4.3 and focus among other things on adding formulas and entries in the knowledge base. This has to be done concurrently on an entire vector component to ensure that the same attacker actions can be performed in all of its elements, that is, that they have statically-equivalent solutions. The rules assume that the constraint systems have been normalised by the normalisation rules (see Figure 4.2), and one of them uses our custom notation for applying a substitution Σ to a formula (Section 2.1, Definition 4.7). Finally, for the sake of succinctness, if $S = (\mathcal{P}, \mathcal{C}, \mathcal{C}^e)$ is an extended symbolic process we refer as $\Phi(S), \mathsf{E}^1(S), \mathsf{E}^2(S), \ldots$ to the corresponding components of $\mathcal{C}^e$.

$$\mathbb{S} \cup \{\Gamma \cup \{(\mathcal{P}, \mathcal{C}, \bot)\}\} \rightsquigarrow \mathbb{S} \cup \{\Gamma\} \qquad \text{(Vect-rm-Unsat)}$$

$$\mathbb{S} \cup \{\Gamma\} \rightsquigarrow \mathbb{S} \cup \{\Gamma^+, \Gamma^-\} \qquad \text{(Vect-Split)}$$

if Γ^+, Γ^- is a partition of Γ and there exists a formula ψ such that
1 $\forall S \in \Gamma^+$, $\exists \psi' \in \mathsf{F}(S)$, $\psi \simeq_{\mathsf{h}} \psi'$ and ψ' solved; and
2 $\forall S \in \Gamma^-$, $\forall \psi' \in \mathsf{F}(S)$, $\psi \not\simeq_{\mathsf{h}} \psi'$

$$\mathbb{S} \cup \{\Gamma\} \rightsquigarrow \mathbb{S} \cup \{\{S[\mathsf{K} \mapsto \mathsf{K} \wedge \xi \vdash^? u_S] \mid S \in \Gamma\}\} \qquad \text{(Vect-add-Conseq)}$$

if for all $S \in \Gamma$, S is solved, $\xi \vdash^? u_S \in \mathsf{F}(S)$ and for all second-order term ζ, $(\zeta, u_S) \notin \mathsf{Conseq}(\mathsf{K}(S) \cup \mathsf{D}(S))$

$$\mathbb{S} \cup \{\Gamma\} \rightsquigarrow \mathbb{S} \cup \{\{S[\mathsf{F} \mapsto \mathsf{F} \wedge \psi{:}(\Sigma, S))] \mid S \in \Gamma\}\} \qquad \text{(Vect-add-Formula)}$$

if $\psi = \forall X, Y, z.\, (X \vdash^? z \wedge Y \vdash^? z) \Rightarrow X =_f^? Y$ and $\Sigma = \{X \mapsto \xi, Y \mapsto \zeta\}$, and for all $S \in \Gamma$,
1 S is solved
2 $\mathsf{F}(S)$ contains a formula of the form $\xi \vdash^? u_S$ and, for at least one of the first-order terms u_S, we have $(\zeta, u_S) \in \mathsf{Conseq}(\mathsf{K}(S) \cup \mathsf{D}(S))$
3 for all $(\varphi \Rightarrow \zeta_1 =_f^? \zeta_2) \in \mathsf{F}(S)$, $\zeta_1 \neq \xi$ and $\zeta_2 \neq \xi$

Figure 4.3 Vector-simplification rules for sets of sets of extended symbolic processes

Rule (Vect-rm-Unsat) removes $\bot$ elements from the vector. Rule (Vect-Split) splits a component whenever a common solution would yield statically inequivalent frames. More specifically, the rule separates the constraint systems in Γ^+ in which a given recipe always yields a message (resp. an equality always holds) from the constraint systems in Γ^- in which the same recipe would never yield a message (resp. the same equality would never hold). This is characterised by the fact that a deduction (resp. equality) formula is solved in some constraint systems and not in the others. Rule (Vect-add-Conseq) adds a solved deduction formula from $\mathsf{F}(S)$ to $\mathsf{K}(S)$ when this formula is solved in the entire component Γ and the new knowledge-base entries are not redundant with existing ones. Finally, when an equality fact $\xi =_f^? \zeta$ should hold in one constraint system, Rule (Vect-add-Formula) adds it to the entire component Γ (with appropriate hypotheses). Observe that we use in this rule the placeholder formula $\psi = \forall X, Y, z.\, (X \vdash^? z \wedge Y \vdash^? z) \Rightarrow X =_f^? Y$, introduced in Section 1.1, stating that two recipes deducing the same term should verify an equality fact.

4 *Constraint solving*: case distinction rules

> **Section summary**
> These rules are the core of the constraint solver: they generate constraints, based on *case analyses*, that help refining vector components to obtain remarkable properties (each component has a unique mgs, saturated knowledge base, statically-equivalent solutions...).

Our case distinction rules take the form of a transition system on vectors $\mathbb{S}$ of extended symbolic processes similarly to the vector-simplification rules. There are three different rules, each operating in a similar manner: given a vector $\mathbb{S} \cup \{\Gamma\}$, all rules perform a transformation of the following form on one component Γ:

$$\mathbb{S} \cup \{\Gamma\} \to \mathbb{S} \cup \{\Gamma^+, \Gamma^-\} \qquad (\star\star)$$

where Γ^+ (the *positive branch*) is intuitively obtained by applying a mgs Σ on each symbolic processes of Γ and Γ^- (the *negative branch*) by adding the formula $\neg\Sigma$ to each symbolic process $S \in \Gamma$, where

$$\neg\Sigma = \forall S. \bigvee_{X \in dom(\Sigma)} X \neq^? X\Sigma \qquad \text{with } S = vars^2(img(\Sigma)) \setminus vars^2(S)$$

Intuitively this refines the component Γ by considering the cases where Σ is a solution or not. After that, normalising the refined components Γ^+ and Γ^- with the simplification rules—in particular Rules (Vect-rm-Unsat) and (Vect-Split)—will discard impossibles cases and separate processes with newly-found non-statically-equivalent solutions. The four case distinction rules (Sat), (Eq), (Rew) and (Chan) are presented in the next sections by specifying how Γ^+ and Γ^- are computed from Γ. They are applied using a particular strategy defined by the following ordering on rules (where $<$ means "has priority over"):

$$\text{Sat} < \text{Eq} < \text{Rew} < \text{Chan}$$

Note that only the fact that Rew $<$ Chan is needed to prove the correctness of the procedure, while the minimality of Sat will be needed in Chapter 6 for complexity. The relative ordering of (Eq) and (Rew) is arbitrary.

4.1 Rule Sat

The first rule focuses on satisfiability: its goal is to separate extended constraint systems of Γ that have not the same solutions. For example if we have $S = (\mathcal{P}, \mathcal{C}, \mathcal{C}^e) \in \Gamma$ and $\Sigma \in mgs(\mathcal{C}^e)$, all other symbolic processes $S' \in \Gamma$ should also have a solution that is an instance of Σ (and if not, the component Γ should be split to separate S and S'). In particular this ensures that when this rule cannot be applied anymore, all extended constraint system in Γ share a common, unique mgs (in particular they are in solved form). The same mechanism can be used to consider the solutions Σ making trivial some disequations of E^1 or hypotheses of some formulas in F. In particular the normalisation rules defined earlier in Section 3.2 will then handle the now trivial or unsatisfiable constraints. All this can be formalised as an instance of $(\star\star)$ with:

$$\begin{aligned}\Gamma^+ &= \{S{:}\Sigma \mid S \in \Gamma\}\\ \Gamma^- &= \{S[\mathsf{E}^2 \mapsto \mathsf{E}^2 \wedge \neg\Sigma] \mid S \in \Gamma\}\end{aligned} \tag{Sat}$$

where there exists $S \in \Gamma$ such that either

1. S not solved and $\Sigma \in mgs(S)$; or
2. there exists $\psi \in \mathsf{F}(S)$ not solved and $\Sigma \in mgs(S[\mathsf{E}^1 \mapsto \mathsf{E}^1 \wedge \mathsf{hyp}(\psi)])$; or
3. $\mathsf{E}^1(S)$ contains a disequality $\psi = \forall \tilde{x}.\phi$ and $\Sigma \in mgs(S[\mathsf{E}^1 \wedge \psi \mapsto \mathsf{E}^1]mgu(\neg\phi))$.

4.2 Rule Eq

The second case distinction rule focuses on the static equivalence between solutions of extended constraint systems. More specifically, the rule (Eq) checks whether an entry $\xi_1 \vdash^? u_1$ of one knowledge base of Γ can deduce the same term as another recipe ξ_2 consequence of K. The rule is formalised as an instance of $(\star\star)$ with

$$\begin{aligned}\Gamma^+ &= \{S{:}\Sigma[\mathsf{F} \mapsto \mathsf{F} \wedge \psi{:}(\Sigma_0\Sigma, S{:}\Sigma)] \mid S \in \Gamma\}\\ \Gamma^- &= \{S[\mathsf{E}^2 \mapsto \mathsf{E}^2 \wedge \neg\Sigma] \mid S \in \Gamma\}\end{aligned} \tag{Eq}$$

if there exist $S \in \Gamma$, $\Sigma \in mgs(S[\mathsf{E}^1 \mapsto \mathsf{E}^1 \wedge \mathsf{hyp}(\psi{:}(\Sigma_0, S))])$, such that $\psi{:}(\Sigma_0\Sigma, S{:}\Sigma)$ is a solved formula and:

1. either $\Sigma_0 = \{X \to \xi_1, Y \to \xi_2\}$ for some $(\xi_1 \vdash^? u_1), (\xi_2 \vdash^? u_2) \in \mathsf{K}(S)$ and for all $(\varphi \Rightarrow H) \in \mathsf{F}(S)$, $H \neq (\xi_1 =^?_f \xi_2)$; or
2. $\Sigma_0 = \{X \to \xi_1, Y \to \mathsf{f}(X_1, \ldots, X_n)\}$ for some $(\xi_1 \vdash^? u_1) \in \mathsf{K}(S)$ and $\mathsf{f}/n \in \mathcal{F}_\mathsf{c}$ with $X_1{:}k, \ldots, X_n{:}k$ fresh and for all $(\varphi \Rightarrow \zeta_1 =^?_f \zeta_2) \in \mathsf{F}(S)$, $\zeta_b = \xi_1$ implies $root(\zeta_{1-b}) \neq \mathsf{f}$.

where $k = |dom(\Phi(S))|$ and $\psi = \forall X, Y, z.\,(X \vdash^? z \wedge Y \vdash^? z) \Rightarrow X =^?_f Y$ with $X{:}k, Y{:}k, z$ fresh variables.

Similarly to Rule (Vect-add-Formula) the rule uses the generic equality formula ψ and the hypotheses of $\psi{:}(\Sigma_0, S)$ express that $X\Sigma_0$ and $Y\Sigma_0$ deduce the same term. Since a recipe consequence of K can either be coming from a deduction fact in K or be a recipe with a constructor symbol at its root, we consider the two cases 1 and 2 each with the appropriate instantiation Σ_0 of the placeholders X and Y. The side requirements that head-equivalent formulas should not already be present in $\mathsf{F}(S)$ are simply here for termination purpose, thus avoiding infinite aggregation of redundant formulas.

4.3 Rule Rew

The third case distinction rule focuses on saturating the knowledge base. For example when outputting a term u, the corresponding symbolic rule (E-Out) will add a deduction fact $\mathsf{ax}_n \vdash^? u$ to F; the rule (Rew) will apply rewrite rules on u to determine whether new messages can be learned by the attacker. Typically if $u = \langle u_1, u_2\rangle$ the following actions will happen:

1. after $\mathsf{ax}_n \vdash^? u$ has been added to F by the symbolic rule (E-Out), it will be copied to the knowledge base K by the simplification rules (Vect-add-Conseq) (assuming u is not already deducible from any knowledge base of the component)
2. after that, the case-distinction rule (Rew) will add the two deduction facts $\mathsf{fst}(\mathsf{ax}_n) \vdash^? u_1$ and $\mathsf{snd}(\mathsf{ax}_n) \vdash^? u_2$ to F, which may in turn be transferred to K as well.

More precisely, given a deduction fact $\xi_0 \vdash^? u_0$, the rule checks whether one may apply a rewrite rule $\ell \to r$ to u_0, which may require to first apply a context on u_0 (for example if $u_0 = \mathsf{h}(a)$ and $\ell = \mathsf{f}(\mathsf{g}(\mathsf{h}(x)))$). For that we introduce a notion of *skeleton* of ℓ.

Definition 4.10 (rewriting skeleton)

Let p be a position of a first-order term ℓ. A *skeleton for* (ℓ, p) is a tuple (ξ, t, D) such that $\xi \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_0 \cup \mathcal{X}^2)$, $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_0 \cup \mathcal{X}^1)$, D is a set of deduction facts and

$$(root(\xi_{|q}), root(t_{|q})) = \begin{cases} (root(\ell_{|q}), root(\ell_{|q})) & \text{for any strict prefix } q \text{ of } p \\ (X_q, x_q) & \text{for any other position } q \text{ of } \xi \end{cases}$$

where the set of variables X_q (resp. x_q), q a position of ξ, are fresh pairwise distinct second-order (resp. first-order) variables and D is the set of all the deduction facts $X_q \vdash^? x_q$. The set of all such skeletons (which, notably, are all identical up to variable renaming but may therefore differ on the second-order-variable types) is written $\mathsf{Skel}(\ell, p)$.

For a skeleton $(\xi, t, D) \in \mathsf{Skel}(\ell, p)$, the recipe ξ represents the context that the attacker will apply on top of the deduction fact $\xi_0 \vdash^? u_0$ at the position p to obtain the left-hand side ℓ. The term t represents the corresponding generic term on which the rewrite rule will be applied. Finally D is the set of deduction facts linking the variables of ξ and t.

Consider now a component Γ, a symbolic process $S \in \Gamma$, a deduction fact $(\xi_0 \vdash^? u_0) \in \mathsf{K}(S)$, and a context C. The first role of Rule (Rew) is to saturate the knowledge base, that is, to deduce the new term $C[u_0]\!\downarrow$ using $C[\xi_0]$. However after adding the new deduction fact $C[\xi_0] \vdash^? C[u_0]\!\downarrow$ to $\mathsf{F}(S)$, a head-equivalent formula should be added to all other symbolic processes of Γ whenever it is possible, so that the vector-simplification rule (Vect-Split) (which separates processes with non-statically-equivalent solutions) only separates $S' \in \Gamma$ from S if $C[\xi_0]$ yields a valid message in S but not in S'. Yet the behaviour of a destructor symbol may be described by multiple rewrite rules: the rewrite rule used to normalise $C[u_0]$ may therefore not be the same as the one used to normalise the term deduced by $C[\xi_0]$ in S'. Because of this we have to add to $\mathsf{F}(S')$ all formulas corresponding to using all possible rewrite rules. For that we consider the following set of generic formulas:

$$\mathsf{RewF}(\xi, \ell \to r, p) = \left\{ \forall S.\, (D \wedge mgu(\ell' =^? t)) \Rightarrow \xi \vdash^? r' \;\middle|\; \begin{array}{l} \ell' \to r' \in \mathcal{R} \\ (\xi, t, D) \in \mathsf{Skel}(\ell, p) \\ S = vars(D, \ell') \end{array} \right\}$$

Let us now give a complete example to illustrate all these notions. The goal is to detail what formulas will be added to F by Rule (Rew) on a concrete case as the actual definition of the rule is quite technical and hard to read—although the intuition behind it is rather simple.

▶ *Example 4.3*

Consider a theory defined by a binary symbol h and the two rewrite rules

$$\mathsf{getOther}(\mathsf{h}(x, y), x) \to y \qquad\qquad \mathsf{getOther}(\mathsf{h}(x, y), y) \to x$$

These two rewrite rules give access to either argument of h assuming the other one is known. Now consider a component Γ containing two extended symbolic processes S_1 and S_2 with the

respective frames, given $k, k', s, s' \in \mathcal{N}$:

$$\Phi(S_1) = \{\mathsf{ax}_1 \mapsto \mathsf{h}(k, k'), \mathsf{ax}_2 \mapsto k\} \qquad \Phi(S_2) = \{\mathsf{ax}_1 \mapsto \mathsf{h}(s, s'), \mathsf{ax}_2 \mapsto s'\}$$

They are are statically equivalent, even if the recipe $\mathsf{getOther}(\mathsf{ax}_1, \mathsf{ax}_2)$ is not normalised using the same rewrite rule in $\Phi(S_1)$ and $\Phi(S_2)$. We assume that $\mathsf{K}(S_1) = \mathsf{ax}_1 \vdash^? \mathsf{h}(k, k') \wedge \mathsf{ax}_2 \vdash^? k$ and $\mathsf{K}(S_2) = \mathsf{ax}_1 \vdash^? \mathsf{h}(s, s') \wedge \mathsf{ax}_2 \vdash^? s'$. We describe the application of Rule (Rew) that uses the rewrite rule $(\ell \to r) = (\mathsf{getOther}(\mathsf{h}(x, y), x) \to y)$ to deduce a new term in S_1 by putting ax_1 at the position of $\mathsf{h}(x, y)$ in ℓ. To begin the rule considers a skeleton $(\xi, t, D) \in \mathsf{Skel}(\ell, 1)$:

$$\xi = \mathsf{getOther}(X_1, X_2) \qquad t = \mathsf{getOther}(x_1, x_2) \qquad D = X_1 \vdash^? x_1 \wedge X_2 \vdash^? x_2$$

The set $\mathsf{RewF}(\xi, \ell \to r, 1)$ therefore contains the following two formulas (normalised by the simplification rules on formulae):

$$\psi_1 = \forall X_1, X_2, x, y.\ (X_1 \vdash^? \mathsf{h}(x, y) \wedge X_2 \vdash^? x) \Rightarrow \mathsf{getOther}(X_1, X_2) \vdash^? y$$
$$\psi_2 = \forall X_1, X_2, x, y.\ (X_1 \vdash^? \mathsf{h}(x, y) \wedge X_2 \vdash^? y) \Rightarrow \mathsf{getOther}(X_1, X_2) \vdash^? x$$

They are only generic formulas indicating that when the left side of a rewrite rule can be computed then the right side can be computed as well. Since our goal is to apply the rewrite rule $\ell \to r$ where the deduction fact $\mathsf{ax}_1 \vdash^? \mathsf{h}(k, k')$ is used to deduce the subterm $\ell_{|1}$, we have to replace the variable at position 1 in ξ, namely X_1, by ax_1 in these formulas. We do this by applying the substitution $\Sigma_0 = \{X_1 \mapsto \mathsf{ax}_1\}$ to ψ_1, ψ_2 (in the sense defined in Section 2.1, again normalised by simplification rules):

$$\psi_1{:}(\Sigma_0, S_1) = \forall X_2.\ X_2 \vdash^? k \Rightarrow \mathsf{getOther}(\mathsf{ax}_1, X_2) \vdash^? k'$$
$$\psi_2{:}(\Sigma_0, S_1) = \forall X_2.\ X_2 \vdash^? k' \Rightarrow \mathsf{getOther}(\mathsf{ax}_1, X_2) \vdash^? k$$

Rule (Rew) will then compute most general solutions to instantiate X_2 in a way that satisfies the hypotheses of these formulas. In the case of the first formula we have the unique solution $mgs(S_1[\mathsf{D} \mapsto \mathsf{D} \wedge X_2 \vdash^? k]) = \{\Sigma\}$ where $\Sigma = \{X_2 \mapsto \mathsf{ax}_2\}$; the algorithm will therefore add the following deduction fact to $\mathsf{F}(S_1)$:

$$\psi_1{:}(\Sigma_0\Sigma, S_1{:}\Sigma) = \mathsf{getOther}(\mathsf{ax}_1, \mathsf{ax}_2) \vdash^? k'$$

On the contrary for the second formula we have $mgs(S_1[\mathsf{D} \mapsto \mathsf{D} \wedge X_2 \vdash^? k']) = \varnothing$, meaning that no solutions satisfy its hypotheses. Then we are almost done: as explained earlier it only remains to add a head-equivalent formula to $\mathsf{F}(S_2)$, if any, so that the vector-simplification rule (Vect-Split) does not split Γ if the recipe $\mathsf{getOther}(\mathsf{ax}_1, \mathsf{ax}_2)$ yields a valid message in both S_1 and S_2. That is, we should add to $\mathsf{F}(S_2)$:

$$\psi_1{:}(\Sigma_0\Sigma, S_2{:}\Sigma) = (s' =^? s \Rightarrow \mathsf{getOther}(\mathsf{ax}_1, \mathsf{ax}_2) \vdash^? s')$$
$$\psi_2{:}(\Sigma_0\Sigma, S_2{:}\Sigma) = (s' =^? s' \Rightarrow \mathsf{getOther}(\mathsf{ax}_1, \mathsf{ax}_2) \vdash^? s)$$

This time the situation is reversed compared to S_1: the first formula has unsatisfiable hypotheses (and will therefore be discarded at the next round of normalisation rules by Rule (Norm-Formula)) and the second one will be simplified to $\mathsf{getOther}(\mathsf{ax}_1, \mathsf{ax}_2) \vdash^? s$. This illustrates why we add one formula for each rewrite rule: should we have only considered ψ_1, we would have missed the head-equivalent formula $\mathsf{getOther}(\mathsf{ax}_1, \mathsf{ax}_2) \vdash^? s$ in S_2, resulting in the incorrect conclusion that $\Phi(S_1) \not\sim \Phi(S_2)$. ◀

Let us now formalise Rule (Rew) in full generality. It can now be defined as an instance of $(\star\star)$ under the following conditions:

$$\begin{aligned}\Gamma^+ &= \{S{:}\Sigma[\mathsf{F} \mapsto \mathsf{F} \wedge \mathsf{F}_S]) \mid S \in \Gamma\}\\ \Gamma^- &= \{S[\mathsf{E}^2 \mapsto \mathsf{E}^2 \wedge \neg\Sigma] \mid S \in \Gamma\}\end{aligned} \tag{Rew}$$

if there exist $S \in \Gamma$, $\ell \to r \in \mathcal{R}$, p position of ℓ, $\xi \in \mathcal{T}(\mathcal{F} \cup \mathcal{X}^2_{\leqslant k})$ with $k = |dom(\Phi(S))|$, $\psi_0 \in \mathsf{RewF}(\xi, \ell \to r, p)$, $(\xi_0 \vdash^? u_0) \in \mathsf{K}(S)$, such that the following conditions are met:

1. $p \neq \varepsilon$ and $\ell_{|p} \notin \mathcal{X}^1$
2. $\mathsf{F}_S = \{\psi{:}(\Sigma_0\Sigma\Sigma_1, S{:}\Sigma) \mid \psi \in \mathsf{RewF}(\xi, \ell \to r, p)\}$
3. $\Sigma_0 = \{\xi_{|p} \to \xi_0\}$ and $\Sigma \in mgs(S[\mathsf{D} \mapsto \mathsf{D} \wedge \mathsf{D}(\psi_1), \mathsf{E}^1 \mapsto \mathsf{E}^1 \wedge \mathsf{hyp}(\psi_1)])$ if $\psi_1 = \psi_0{:}(\Sigma_0, S)$
4. $\psi_0{:}(\Sigma_0\Sigma, S{:}\Sigma)$ is a formula of the form $\forall S.\ (\bigwedge_{i=1}^n X_i \vdash^? x_i) \Rightarrow H$ with all X_i, x_i distinct
5. Σ_1 is an injection from $\{X_i\}_{i=1}^n$ to fresh constants, and for any such injection Σ_1', we have $\psi_0{:}(\Sigma_0\Sigma\Sigma_1', S{:}\Sigma) \notin \mathsf{F}(S)$.

The conditions are rather technical but simply capture the steps of the example. Commenting the requirements of the rule, Item 1 ensures that the rewriting is not performed at a trivial position. Items 2 and 3 describe the formulas added to each symbolic process of Γ: just as in the example they are obtained by choosing one $S \in \Gamma$, computing a formula ψ_0 (ψ_1 in the example) corresponding to applying one given rewrite rule $\ell \to r$ in S, replacing the position p of ℓ by an entry of $\mathsf{K}(S)$ by applying Σ_0 and computing a solution Σ of the hypotheses of the resulting formula. Finally all other symbolic processes $S' \in \Gamma$ receive the formulas of $\mathsf{F}_{S'}$, each attempting to apply a rewrite rule to yield a valid message with the same recipe as in ψ_0.

Note that the computation of the mgs Σ may leave some second-order variables $X_1, \ldots, X_n$ unconstrained because they do not need to be instantiated in a particular way to obtain a solution. This is where Items 4 and 5 come into play, replacing these pending variables by fresh constants.

4.4 Rule Chan

Finally we present a rule dedicated to the treatment of internal-communication channels in the private semantics. Due to the solving strategy mentioned at the beginning of the section, this rule will only by applied when Rule (Rew) is not applicable anymore, ensuring that K has been saturated, i.e., that any deducible term is a consequence of the knowledge base. The rule can therefore be formalised as an instance of $(\star\star)$ with

$$\begin{aligned}\Gamma^+ &= \{S{:}\Sigma \mid S \in \Gamma \setminus \{S_0\}\}\\ \Gamma^- &= \left\{S[\mathsf{E}^2 \mapsto \mathsf{E}^2 \wedge \neg\Sigma] \mid S \in (\Gamma \setminus \{S_0\}) \cup \{S_0'\}\right\}\end{aligned} \tag{Chan}$$

if $S_0 \in \Gamma$ and $S_0' = S_0[\mathsf{D} \wedge \varphi \mapsto \mathsf{D}]$ for some $\varphi = (\forall X{:}i.\ X \not\vdash^? u) \in \mathsf{D}(S_0)$ such that either

1. $\Sigma \in mgs(S_0'[\mathsf{D} \wedge \mathsf{D} \wedge X \vdash^? u])$; or
2. $mgs(S_0'[\mathsf{D} \mapsto \mathsf{D} \wedge X \vdash^? u]) = \varnothing$ and $\Sigma = \bot$ (with the convention $\Gamma^+ = \varnothing$).

The rule picks an arbitrary extended symbolic process $S_0 \in \Gamma$ such that $\mathsf{D}(S_0)$ still contains a non-deducibility constraint φ. Then the rule checks whether a solution Σ that falsifies φ

exists. If there is one (case 1), S_0 is removed from the positive branch and φ is removed from the negative branch. If there are no such solutions (case 2) this means that the adversary has no mean to compute the channel u: the positive branch is therefore entirely aborted, while the constraint φ is safely removed from the negative branch without adding new constraints.

5 All in all: computing a partition tree

> ***Section summary***
> In this section we gather all of our constraint-solving relations to give a *complete procedure* for computing a partition tree, thus detailing the informal algorithm provided in Chapter 3. We prove that the constraint solving is *partially correct*, i.e. that it has the expected outcome when it terminates (termination itself being studied in Chapter 6).

Overall procedure We make reference to the various constraint-solving relations defined in the previous sections using the following notations:

$\overset{\mathsf{simpl}}{\rightsquigarrow}$: simplification rules on formulas $\qquad$ $\overset{\mathsf{norm}}{\rightsquigarrow}$: normalisation rules

$\overset{\mathsf{vect}}{\rightsquigarrow}$: vector-simplification rules

$\xrightarrow{\textsc{Sat}}$: (Sat) case-distinction rule $\qquad$ $\xrightarrow{\textsc{Eq}}$: (Eq) case-distinction rule

$\xrightarrow{\textsc{Rew}}$: (Rew) case-distinction rule $\qquad$ $\xrightarrow{\textsc{Chan}}$: (Chan) case-distinction rule

All these transition relations are interpreted as binary relations on vectors. We recall that we call a *component* a set Γ of extended symbolic processes and a *vector* a set $\mathbb{S}$ of components, and that all $(\mathcal{P}, \mathcal{C}, \mathcal{C}^e) \in \Gamma$ induce a predicate π on second-order solutions of $\mathcal{C}$ such that

$$Sol^{\pi}(\mathcal{C}) = \{(\Sigma_{|vars^2(\mathcal{C})}, \sigma_{|vars^1(\mathcal{C})}) \mid (\Sigma, \sigma) \in Sol(\mathcal{C}^e)\}$$

We therefore propose in Algorithm 1 a procedure to compute the partition tree of two bounded plain processes, where the nodes are labelled by components instead of regular partition-tree configurations; in particular the proof of correctness of this algorithm has to justify that the above predicate π can be defined uniformly across the entire nodes of the computed tree.

Correctness arguments To conclude we mention that the core arguments justifying that Algorithm 1 effectively generates a partition tree can be found in Appendix B. Technically, most of the theorem statements rely on a collection of invariants, with a proof of their preservation at each step of the procedure (called with "Inv" names such as Inv_{wf}, Inv_{sound},...). For the sake of precision they will be mentioned in another technical chapter of this thesis (termination proof in Chapter 6) but knowing their exact definition is not necessary to understand the body of the thesis.

Algorithm 1: Computation of the partition tree, with nodes labelled with components

// Application of simplification rules, as much as possible
Procedure applySimpl($\mathbb{S}$: Vector) : Vector =
  if $\mathbb{S} \overset{\mathsf{simpl}}{\rightsquigarrow} \mathbb{S}'$ **then** return applySimpl($\mathbb{S}'$)
  else if $\mathbb{S} \overset{\mathsf{norm}}{\rightsquigarrow} \mathbb{S}'$ **then** return applySimpl($\mathbb{S}'$)
  else if $\mathbb{S} \overset{\mathsf{vect}}{\rightsquigarrow} \mathbb{S}'$ **then** return applySimpl($\mathbb{S}'$)
  else return $\mathbb{S}$

// Application of case-distinction rules, with simplification rules in between
Procedure applyCase($\mathbb{S}$: Vector) : Vector =
  if $\mathbb{S} \xrightarrow{\textsc{Sat}} \mathbb{S}'$ **then** return applyCase(applySimpl($\mathbb{S}'$))
  else if $\mathbb{S} \xrightarrow{\textsc{Eq}} \mathbb{S}'$ **then** return applyCase(applySimpl($\mathbb{S}'$))
  else if $\mathbb{S} \xrightarrow{\textsc{Rew}} \mathbb{S}'$ **then** return applyCase(applySimpl($\mathbb{S}'$))
  else if $\mathbb{S} \xrightarrow{\textsc{Chan}} \mathbb{S}'$ **then** return applyCase(applySimpl($\mathbb{S}'$))
  else return $\mathbb{S}$

// Generates the subtree rooted on a node labelled by the component Γ
Procedure generateSubtree(Γ : Component) : Tree =
  $\Gamma_{\mathsf{in}} \leftarrow \{S' \mid S \overset{Y(X)}{\Longrightarrow}_{\mathsf{s}} S', S \in \Gamma\}$
  $\Gamma_{\mathsf{out}} \leftarrow \{S' \mid S \overset{\overline{Y}\langle \mathsf{ax} \rangle}{\Longrightarrow}_{\mathsf{s}} S', S \in \Gamma\}$
  if $\Gamma_{\mathsf{in}} = \Gamma_{\mathsf{out}} = \varnothing$ **then**
    return a tree reduced to its root, labelled Γ
  else
    $\mathbb{S} \leftarrow$ applyCase(applySimpl($\{\Gamma_{\mathsf{in}}, \Gamma_{\mathsf{out}}\}$))
    $T \leftarrow$ tree with root Γ and children generateSubtree(Γ') for each $\Gamma' \in \mathbb{S}$
    return T

// Generates the root and then an entire partition tree of P_1 and P_2
Procedure PTree(P_1, P_2 : Processes) : Tree =
  $\Gamma_1 \leftarrow \{S \mid P \overset{\varepsilon}{\Rightarrow}_{\mathsf{s}} S\}$
  $\Gamma_2 \leftarrow \{S \mid Q \overset{\varepsilon}{\Rightarrow}_{\mathsf{s}} S\}$
  $\{\Gamma\} \leftarrow$ applySimpl($\{\Gamma_1 \cup \Gamma_2\}$) // in the root, simplification rules never split the vector
  return generateSubtree(Γ)

Chapter 5:

Exploiting proof symmetries: equivalence by session

Summary.

In this chapter we study how the procedure for trace equivalence developed in the previous chapters can exploit the *symmetries* that often arise during practical verification. Indeed in our formalism, privacy is often modelled as the equivalence of two processes that *only differ on the value of a sensitive attribute*: in particular they share a very similar structure that is often reflected as symmetries in the equivalence proof. We present here *equivalence by session*, a sound refinement of trace equivalence that benefits from *partial-order reduction* optimisations that factor out this redundant proof work. Experiments exhibit verification times reduced by orders of magnitude using this technique.

1 Equivalence by session

> ***Section summary***
>
> We introduce *equivalence by session*, a novel refinement of trace equivalence and an abstract notion of *correct refinement* to help proving optimisations correct in a modular way. We also compare it with other existing techniques that also exploit the structural properties of processes to prove equivalence properties (*diff-equivalence* and partial order reductions for *determinate processes*).

1.1 Motivations: exploiting the process structure in equivalence proofs

In this chapter we focus on optimisations for the decision of trace equivalence in the private semantics. The procedure of the previous chapter relies on a constraint-solving procedure that enumerates a finite set of symbolic traces that abstracts all traces of the processes to be proved equivalent. This results into a general procedure that can handle arbitrary (bounded) processes. However the processes to be proved equivalent are generally not completely arbitrary: in our benchmarks for example we always prove statements of the form $P \approx Q$ where P and Q have the same overall structure but differ on the value of the attributes to be proved private. Exploiting this structural information may lead to a more efficient procedure generating less traces in practice. We mention below several approaches to achieve this goal.

Restricted optimisations The first appraoch consists of developing optimisation techniques that are only sound for some classes of processes. This is for example the case of

[BDH15] that develops *partial order reductions* for proving equivalence of *action-determinate processes*. Intuitively a process P is action-determinate if it contains no private channels and no sequences of transitions from P can reach a process where two inputs (resp. outputs) on the same channel are executable in parallel, which ensures that the traces of two equivalent processes always have identical actions available in parallel. We define this property formally later in Section 1. For this class of processes the optimisation of [BDH15] takes the form of a strategy when executing transitions that may discard exponentially-many traces (typically the strategy always executes public outputs in priority over public inputs). We already mentioned this work in Chapter 3, Section 3.2 as this optimisation is integrated into the DeepSec prover. Just as this resulted in decreasing the verification time of the APTE tool by orders of magnitude in [BDH15], significant improvements could also be observed after integration into the DeepSec prover. To illustrate the gain we used a simple model of anonymity in the Private Authentication Protocol, similar to the one used in Chapter 3 but with a fixed scenario:

Number of parallel sessions	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
verif. time (non-determinate model)	<1s	1s	3h 42m	>12h				
verif. time (determinate model)	<1s	<1s	<1s	<1s	1s	1s	5s	42s

Figure 5.1 Verification times for a fixed scenario of anonymity in the PAP

Many processes in practice can be turned determinate: consider the simple example of two processes of the form $P = P_1 \mid \cdots \mid P_n$ and $Q = Q_1 \mid \cdots \mid Q_n$ where P_i, Q_i do not contain parallel operators and use a unique, public channel c. If proving $P \approx_t Q$ only requires to match actions of each P_i with actions of $Q_{\pi(i)}$ (π permutation of $[\![1, n]\!]$) then for some fresh channels $c_1, \ldots, c_n$ the processes

$$P_1[c \mapsto c_i] \mid \cdots \mid P_n[c \mapsto c_i] \qquad\qquad Q_{\pi(1)}[c \mapsto c_i] \mid \cdots \mid Q_{\pi(n)}[c \mapsto c_i]$$

are trace equivalent as well and determinate. Although this kind of structural property is rather common it does not hold in many cases: among the examples studied in Chapter 2 we can mention the Helios protocol (due to the presence of private channels) and BAC (because the choice of the permutation π depends on dynamic values such as attacker inputs).

An extension of some partial order reductions of [BDH15] has also been considered for arbitrary processes [BDH18a], but has a less spectacular impact on performances than [BDH15]. Moreover the technique requires a precomputation that limits the efficiency gain, resulting in an overall rather weak reduction of the verification time.

Restricted equivalences The second approach, similar in spirit, is to design more restricted notions of equivalence, expected to be easier to prove than trace equivalence while overlapping with it as much as possible in practice. This is for example the case of *diff equivalence*, which can intuitively be seen as an analogue of trace equivalence where the two equivalent processes are required to follow the exact same control flow. It is used by the popular ProVerif tool that exploits the structural properties of diff equivalence to prove it using a translation of the problem into Horn clauses, making the verification very efficient in practice. Some variants of diff-equivalence (in the sense that the formalism differs) are also implemented in Tamarin

and Maude-NPA. We define formally this equivalence notion in the context of the applied pi calculus later in Section 1. The counterpart of this proof technique is precision: processes that are non-diff-equivalent may be trace equivalent or labelled bisimilar and the analysis is therefore not always conclusive regarding the latter equivalences. Because of these reasons, ProVerif may typically have difficulty analysing protocols using private channels or that cannot be converted into determinate processes, similarly to the previous paragraph.

Approach of this chapter In this chapter we use, similarly to diff-equivalence, a sound refinement of trace equivalence that is easier to prove in practice. We call it *equivalence by session* in that it proves the equivalence of protocols session by session: the approach is therefore also similar to the one turning the processes determinate. Equivalence by session however permits to conclude equivalence proofs than cannot be successful using diff-equivalence or determinate processes. Typically if we consider the theory defined by the rewrite rules, where $0, 1, \mathsf{ok} \in \mathcal{F}_0$:

$$\mathsf{bool}(0) \to \mathsf{ok} \qquad \mathsf{bool}(1) \to \mathsf{ok} \qquad \neg 0 \to 1 \qquad \neg 1 \to 0$$

then the following processes P and Q are trace equivalent and labelled bisimilar:

$$P = c(x).\,\mathsf{if}\ \mathsf{bool}(x) = \mathsf{ok}\ \mathsf{then}\ (\overline{c}\langle 0\rangle \mid \overline{c}\langle 1\rangle)$$
$$Q = c(x).\,\mathsf{if}\ \mathsf{bool}(x) = \mathsf{ok}\ \mathsf{then}\ (\overline{c}\langle x\rangle \mid \overline{c}\langle \neg x\rangle)$$

They also appear to be equivalent by session: the intuition is that for all values of $x \in \{0, 1\}$, there exists a bijection between the subprocesses of P (i.e., $\{\!\{\overline{c}\langle 0\rangle, \overline{c}\langle 1\rangle\}\!\}$) and the subprocesses Q (i.e., $\{\!\{\overline{c}\langle x\rangle, \overline{c}\langle \neg x\rangle\}\!\}$) matching equivalent processes. The matching is however different for each value of x, which is why the equivalence proof cannot be successful if P and Q are transformed in determinate processes. On the contrary the matching will be selected *dynamically* when proving equivalence by session, which is the reason why the proof is successful with this new technique. Similarly the definition of diff-equivalence will only consider the identity matching, that is, $\overline{c}\langle 0\rangle$ is required to be matched with $\overline{c}\langle x\rangle$ and $\overline{c}\langle 1\rangle$ with $\overline{c}\langle \neg x\rangle$; P and Q can therefore not be proved trace equivalent using diff-equivalence neither. Yet, some modelling tricks can make this simple example provable by the ProVerif tool[1] due to extensions of its theory beyond the original definition of diff-equivalence [CB13]. The situation is however different for more complex, real-world examples such as the BAC protocol that induces similar issues as the above example.

In addition to the scope improvement, we prove that most of the partial order reductions developed in [BDH15] as well as novel ones (e.g., for internal communications) can be used for proving equivalence by session of *any* process. This results in improvements of the verification time of non-determinate processes of several orders of magnitude. Altogether this significantly improves DeepSec's scalability on many examples such as the Helios and BAC protocols.

1.2 Equivalence by session

We now formalise *equivalence by session*: as explained above the main idea is that, when proving the equivalence of P and Q, every action of a given parallel subprocess (i.e., session) of

[1] ProVerif successfully proves $P \approx Q'$ where $Q' = c(x).\,\mathsf{if}\ \mathsf{bool}(x) = \mathsf{ok}\ \mathsf{then}\ \mathsf{if}\ x = 0\ \mathsf{then}\ R(x)\ \mathsf{else}\ R(x)$ and $R(x) = \overline{c}\langle x\rangle \mid \overline{c}\langle \neg x\rangle$.

P should be matched by the actions of a same subprocess in Q. By requiring to match sessions rather than individual actions, this yields a more fine-grained equivalence and effectively reduces the combinatorial explosion.

Session traces We first define a relation $\rightsquigarrow$ that discharges most of the deterministic, unobservable transitions of the semantics so that the strucural requirements of our new equivalence can focus on the other types of transitions only. The relation is defined in Figure 5.2 and gets rid of 0 processes, unexecutable instructions (i.e., inputs and outputs involving non-message terms) and evaluate conditionals and new instructions at toplevel.

$$\overline{u}\langle v\rangle.P \rightsquigarrow 0 \text{ if } \neg\mathsf{msg}(u,v) \qquad u(x).P \rightsquigarrow 0 \text{ if } \neg\mathsf{msg}(u)$$

$$P \mid 0 \rightsquigarrow P \qquad 0 \mid P \rightsquigarrow P \qquad (P \mid Q) \mid R \rightsquigarrow P \mid (Q \mid R)$$

$$\left.\begin{array}{l} P \mid Q \rightsquigarrow P' \mid Q \\ Q \mid P \rightsquigarrow Q \mid P' \end{array}\right\} \text{ if } P \rightsquigarrow P' \qquad P \rightsquigarrow Q \text{ if } P \xrightarrow{\tau} Q \text{ by Rules (Test) or (New)}$$

Figure 5.2 Simplification rules for plain processes

We say that a process P on which no more rules apply is in $\rightsquigarrow$-*normal form*, which means it is of the form

$$P = P_1 \mid (P_2 \mid (P_3 \mid \cdots \mid P_n)) \stackrel{def}{=} P_1 \mid \cdots \mid P_n$$

where each P_i starts with a replication, an input or an output instruction. This witnesses a change of paradigm compared to trace equivalence, where only public inputs and outputs were considered as visible actions. Here unobservable actions are those discharged to the $\rightsquigarrow$ relation and all other types of actions will somewhat be visible, in the sense that the structure of equivalence by session will require that two equivalent traces take even the $\xrightarrow{\tau}$ transitions simultaneously, apart from those embedded in $\rightsquigarrow$. This typically makes internal communications visible, as well as the action of unfolding a replication. Most of the incoming definitions will therefore only operate on processes in $\rightsquigarrow$-normal form.

We write $P\!\downarrow$ a $\rightsquigarrow$-normal form of P (and $A\!\downarrow$ its natural extension to an extended process A), which is unique up to bijective renaming of names generated by new instructions. From this we can formalise a notion of *session* of a process, that is intuitively the (possibly infinite) multiset of its parallel subprocesses.

Definition 5.1 (session)
The multiset of *sessions* of a plain process P in $\rightsquigarrow$-normal form is defined by:

$$\mathsf{sessions}(u(x).P) = \{\!\{u(x).P\}\!\} \qquad \mathsf{sessions}(P_1 \mid \cdots \mid P_n) = \bigcup_{i=1}^{n} \mathsf{sessions}(P_i)$$
$$\mathsf{sessions}(\overline{u}\langle v\rangle.P) = \{\!\{\overline{u}\langle v\rangle.P\}\!\} \qquad \mathsf{sessions}(!P) = \bigcup_{i\in\mathbb{N}} \mathsf{sessions}(P\!\downarrow)\varrho_i$$
$$(\varrho_i)_{i\in\mathbb{N}} \text{ fresh renamings of } \mathit{names}(P\!\downarrow) \setminus \mathit{names}(P)$$

This induces a form of "compact" trace that relegates deterministic unobservable transitions to $\rightsquigarrow$-normalisation and uses sessions instead of Rules (Par) and (Repl). We formalise

this using a labelled transition relation $\xrightarrow{\alpha}_{\mathsf{ss}}$ defined by the two rules:

$$A \xrightarrow{\alpha}_{\mathsf{ss}} B \qquad \text{if } A \xrightarrow{\alpha} C \text{ by Rules (In), (Out) or (Comm), and } C\{\!\!\{\} = B \tag{IO}$$

$$(\{\!\!\{P\}\!\!\} \cup \mathcal{P}, \Phi) \xrightarrow{\tau}_{\mathsf{ss}} (\mathsf{sessions}(P) \cup \mathcal{P}, \Phi) \qquad \text{if } |\mathsf{sessions}(P)| > 1 \tag{Sess}$$

Definition 5.2 (session trace)

A *session trace* of an extended process A_0 in $\rightsquigarrow$-normal form is a sequence of transitions $t : A_0 \xrightarrow{\alpha_1}_{\mathsf{ss}} \cdots \xrightarrow{\alpha_n}_{\mathsf{ss}} A_n$, which may be referred to as $t : A_0 \xrightarrow{\alpha_1 \cdots \alpha_n}_{\mathsf{ss}} A_n$. We emphasise in particular that the τ actions are not erased from the word $\mathit{actions}(t) = \alpha_1 \cdots \alpha_n$. The set of session traces of A is written $\mathbb{T}(A)$.

In particular trace equivalence can be characterised in terms of session traces. For that we extend the notion of static equivalence to (session) traces by writing $t \sim t'$ if $t : A \overset{\mathsf{tr}}{\Longrightarrow} B$, $t' : A' \overset{\mathsf{tr}'}{\Longrightarrow} B'$, $B \sim B'$ and $\mathit{actions}(t) = \mathsf{tr}$ and $\mathit{actions}(t') = \mathsf{tr}'$ are identical up to erasure of τ actions. The characterisation is then formalised by the following proposition:

Proposition 5.1 (characterisation of trace equivalence with session traces)

If A and B are two extended processes in $\rightsquigarrow$-normal form, the following points are equivalent:

1. $A \sqsubseteq_t B$
2. $\forall t \in \mathbb{T}(A), \exists t' \in \mathbb{T}(B), t \sim t'$

Twin processes We now formalise the notion of session matchings at the core of the definition of equivalence by session. For that we define a notion of *skeleton* of a process that indicates its toplevel action:

Definition 5.3 (skeleton equivalence)

The *skeleton* of a process starting with an input or an output is defined by $\mathsf{skel}(u(x).P) = \mathsf{in}_u$ and $\mathsf{skel}(\overline{u}\langle v\rangle.P) = \mathsf{out}_u$. Let then $A = (\mathcal{P}, \Phi)$ and $B = (\mathcal{Q}, \Psi)$ be two extended processes, and $S_A = \bigcup_{R \in \mathcal{P}} \mathsf{sessions}(R)$ and $S_B = \bigcup_{R \in \mathcal{Q}} \mathsf{sessions}(R)$ their respective sessions. We say that A and B are *skeleton equivalent*, written $A \approx_{\mathsf{skel}} B$, if there exists a bijection $\pi : S_A \to S_B$ such that for all $R \in S_A$, either $(\mathsf{skel}(R), \mathsf{skel}(\pi(R))) = (\mathsf{in}_u, \mathsf{in}_{u'})$ or $(\mathsf{skel}(R), \mathsf{skel}(\pi(R))) = (\mathsf{out}_u, \mathsf{out}_{u'})$ with, for all recipes ξ, $\xi\Phi_0 =_E u$ *iff* $\xi\Phi_1 =_E u'$.

Intuitively skeleton equivalence simply states that the same actions can be performed at toplevel from A and B, or more precisely by $R \in S_A$ and $\pi(R)$; typically if $\Phi_0 \sim \Phi_1$ this means that u is a public channel (i.e., u is deducible from Φ_0) *iff* u' is public as well, and if they are, they can be deduced by *at least one* same recipe ξ. Relying on this notion we can then consider the following notion of process pairing:

Definition 5.4 ((extended) twin process)

A *twin process* is a pair (P, Q) of plain processes in $\rightsquigarrow$-normal form. An *extended twin process* is a triple $A^2 = (\mathcal{P}^2, \Phi_0, \Phi_1)$ where Φ_0, Φ_1 are two frames of same domain and $\mathcal{P}^2$ is a multiset of twin processes such that for all $(P, Q) \in \mathcal{P}^2$, $(\{\!\!\{P\}\!\!\}, \Phi_0) \approx_{\mathsf{skel}} (\{\!\!\{Q\}\!\!\}, \Phi_1)$.

An extended twin process thus models two regular extended processes, matched together, able to perform the same action, and that can be retrieved by projection using the notations:

$$\mathsf{fst}(A^2) = (\{\!\{P_0 \mid (P_0, P_1) \in \mathcal{P}^2\}\!\}, \Phi_0) \qquad \mathsf{snd}(A^2) = (\{\!\{P_1 \mid (P_0, P_1) \in \mathcal{P}^2\}\!\}, \Phi_1)$$

An operational semantics lifting the transition relation $\xrightarrow{\alpha}_{\mathsf{ss}}$ to twin processes is then defined in Figure 5.3: the new relation $\xrightarrow{\alpha}_{\mathsf{ss}^2}$ intuitively makes two projections follow the same reduction steps using a session-matching mechanism that replaces Rules (Par) and (Repl). We stress that, by definition, a transition $A^2 \xrightarrow{\alpha}_{\mathsf{ss}^2} B^2$ is possible only if the resulting process B^2 verifies the equivalent-skeleton requirement.

$$(\{\!\{(P_0, Q_0), (P_1, Q_1)\}\!\} \cup \mathcal{P}^2, \Phi_0', \Phi_1') \xrightarrow{\alpha}_{\mathsf{ss}^2} (\{\!\{(P_0', Q_0'), (P_1', Q_1')\}\!\} \cup \mathcal{P}^2, \Phi_0, \Phi_1) \qquad (\mathrm{IO}^2)$$
$$\text{if } (\{\!\{P_0, P_1\}\!\}, \Phi_0) \xrightarrow{\alpha}_{\mathsf{ss}} (\{\!\{P_0', P_1'\}\!\}, \Phi_0') \text{ and } (\{\!\{Q_0, Q_1\}\!\}, \Phi_1) \xrightarrow{\tau}_{\mathsf{ss}} (\{\!\{Q_0', Q_1'\}\!\}, \Phi_1') \text{ by Rule (IO)}$$
$$(\{\!\{(P, Q)\}\!\} \cup \mathcal{P}^2, \Phi_0, \Phi_1) \xrightarrow{\tau}_{\mathsf{ss}^2} (\{\!\{(R, \pi(R))\}\!\}_{R \in \mathsf{sessions}(P)} \cup \mathcal{P}^2, \Phi_0, \Phi_1) \qquad (\textsc{Match})$$
$$\text{if } |\mathsf{sessions}(P)| > 1 \text{ and } \pi : \mathsf{sessions}(P) \to \mathsf{sessions}(Q) \text{ bijection}$$

Figure 5.3 Operational semantics for twin processes

▶ *Example 5.1*

If we have two *bounded* processes $P = P_1 \mid \cdots \mid P_n$ and $Q = Q_1 \mid \cdots \mid Q_n$ in $\rightsquigarrow$-normal form, the application of Rule (Match) from (P, Q) (assuming P and Q have equivalent skeletons) can be characterised as follows:

1. if $n \neq p$ the rule is not applicable but there is a contradiction with the fact that P and Q have equivalent skeletons ($\mathsf{sessions}(P)$ and $\mathsf{sessions}(Q)$ are finite but do not have the same number of elements)
2. if $n = p$ an application will result in a $(\{\!\{(P_1, Q_{\pi(1)}), \ldots, (P_n, Q_{\pi(n)})\}\!\}, \varnothing)$ for some $\pi \in S_n$ (S_n being the set of permutations of $[\![1, n]\!]$) such that the skeleton-equivalence requirement is satisfied. For example if we assume that $P_i = c_i(.)P_i'$ and $Q_i = d_i(.)Q_i'$ with $c_i, d_i \in \mathcal{F}_0$, the set of valid permutations π in the above application of the rule are

$$\{\pi \in S_n \mid \forall i \in [\![1, n]\!], c_i = d_{\pi(i)}\}$$

Since P and Q are assumed to have equivalent skeletons, this set is not empty.

If P and Q are not bounded the matching may be more complex. For example let us consider a model of unlinkability in the BAC protocol, inspired by the model presented in Chapter 2 but with an implicit key distribution and no getChallenge messages:

$$P = !\,\mathsf{new}\ k.\,!\,(\mathsf{Reader}'(k) \mid \mathsf{Passport}'(k)) \qquad Q = !\,\mathsf{new}\ k.\,(\mathsf{Reader}'(k) \mid \mathsf{Passport}'(k))$$

with processes of the form $\mathsf{Reader}'(k) = d(x).Q'(k)$ and $\mathsf{Passport}'(k) = \mathsf{new}\ n.\overline{c}\langle n\rangle.P'(k)$ with $c, d \in \mathcal{F}_0$. For now no more details are needed but we will later give a full definition of Reader' and $\mathsf{Passport}'$, as they will serve as a running example throughout this chapter. If E is a multiset, we write $E^\omega = \bigcup_{i \in \mathbb{N}} E$, that is, the multiset obtained by duplicating E an infinite countable number of times. With this notation we have, given a family $(k_i)_{i \in \mathbb{N}}$ of fresh names:

$$\mathsf{sessions}(P) = \mathsf{sessions}(Q)^\omega \qquad \mathsf{sessions}(Q) = \{\mathsf{Reader}'(k_i), \mathsf{Passport}'(k_i) \mid i \in \mathbb{N}\}$$

Since $\mathsf{sessions}(P)$ and $\mathsf{sessions}(Q)$ are infinite countable, there exist bijections of $\mathsf{sessions}(P)$ to $\mathsf{sessions}(Q)$. The ones suitable from an application of Rule (MATCH), that is, those that satisfy the equivalent-skeleton requirement, are the permutations π such that for all $R \in \mathsf{sessions}(P)$,
1 if $R = \mathsf{Reader}'(k_i)$ then there exists $j \in \mathbb{N}$ such that $\pi(R) = \mathsf{Reader}'(k_j)$;
2 if $R = \mathsf{Passport}'(k_i)$ then there exists $j \in \mathbb{N}$ such that $\pi(R) = \mathsf{Passport}'(k_j)$. ◀

Definition 5.5 (twin trace)
We write $\mathbb{T}(A^2)$ the set of traces of the extended twin process A^2 w.r.t. the transition relation $\xrightarrow{\alpha}_{\mathsf{ss}^2}$ (called *twin traces*). In particular, given a twin trace

$$t^2 : A^2 \xrightarrow{\alpha_1}_{\mathsf{ss}^2} A_1^2 \cdots \xrightarrow{\alpha_n}_{\mathsf{ss}^2} A_n^2 \quad \in \mathbb{T}(A^2)\,,$$

we lift the projection functions by writing

$$\begin{aligned} \mathsf{fst}(t^2) : \mathsf{fst}(A^2) \xrightarrow{\alpha_1}_{\mathsf{ss}} \mathsf{fst}(A_1^2) \xrightarrow{\alpha_2}_{\mathsf{ss}} \cdots \xrightarrow{\alpha_n}_{\mathsf{ss}} \mathsf{fst}(A_{n+1}^2) \qquad & \in \mathbb{T}(\mathsf{fst}(A^2)) \\ \mathsf{snd}(t^2) : \mathsf{snd}(A^2) \xrightarrow{\alpha_1}_{\mathsf{ss}} \mathsf{snd}(A_1^2) \xrightarrow{\alpha_2}_{\mathsf{ss}} \cdots \xrightarrow{\alpha_n}_{\mathsf{ss}} \mathsf{snd}(A_{n+1}^2) \qquad & \in \mathbb{T}(\mathsf{snd}(A^2)) \end{aligned}$$

Similarly to session traces, t^2 may be referred to as $t^2 : A^2 \xrightarrow{\alpha_1 \cdots \alpha_n}_{\mathsf{ss}^2} A_n^2$ when the intermediary processes are not relevant in the context.

Equivalence by session Equivalence by session is similar to trace equivalence but only considers the traces of Q matching the structure of the trace of P under study. This structural requirement is simply formalised by considering twin traces of (P, Q).

Definition 5.6 (equivalence by session)
Let P and Q be two skeleton-equivalent plain processes in $\rightsquigarrow$-normal form. We write $P \sqsubseteq_s Q$ when

$$\forall t \in \mathbb{T}(P),\ \exists t^2 \in \mathbb{T}(P, Q),\ t = \mathsf{fst}(t^2) \text{ and } t \sim \mathsf{snd}(t^2)\,.$$

We say that P and Q are *equivalent by session*, written $P \approx_s Q$, when $P \sqsubseteq_s Q$ and $Q \sqsubseteq_s P$.

Although we have designed equivalence by session to increase efficiency of verification procedures, it is also of independent interest. Equivalence by session captures a notion of indistinguishability against an adversary that is able to distinguish actions originated from different protocol sessions. Such an adversarial model may for instance be considered realistic in protocols where servers dynamically allocate a distinct *ephemeral port* to each session. An attacker would therefore observe these ports and always differentiate one session from another. When considering equivalence by session, this allocation mechanism does not need to be explicitly modelled as it is already reflected natively in the definition. On the contrary for trace equivalence, an explicit modelling within the processes would be needed. For example equivalence by session of two protocol sessions operating on a public channel c,

$$!^n\, P(c) \approx_s !^n\, Q(c)$$

can be encoded by relying on dynamically-generated private channels that are revealed to the

attacker. This can be expressed in the applied pi-calclulus by:

$$!^n P_{fresh} \approx_t !^n Q_{fresh} \qquad \text{where } P_{fresh} = \mathsf{new}\ e.\,\overline{c}\langle e\rangle.\,P(e) \text{ and } Q_{fresh} = \mathsf{new}\ e.\,\overline{c}\langle e\rangle.\,Q(e)$$

Such encodings are however incompatible with determinacy as defined for the partial-order reductions of [BDH15]. Our dedicated equivalence offers similar-in-spirit optimisations that are applicable on all processes.

1.3 Comparison to other equivalences

Relation to trace equivalence We first show that equivalence by session is a sound refinement of trace equivalence.

Proposition 5.2 (soundness of equivalence by session)
If $P \approx_s Q$ then $P \approx_t Q$.

This is immediate as $t^2 \in \mathbb{T}(P,Q)$ entails $\mathsf{snd}(t^2) \in \mathbb{T}(Q)$. The converse does not hold in general, meaning that two processes that are not equivalent by session might be trace equivalent. The simplest example is, for $n \in \mathcal{F}_0$,

$$P = \overline{c}\langle n\rangle.\,\overline{c}\langle n\rangle \qquad Q = \overline{c}\langle n\rangle \mid \overline{c}\langle n\rangle$$

We call *false attacks* traces witnessing a violation of equivalence by session, but that can still be matched trace-equivalence-wise (which does not mean that the processes are effectively trace equivalent). In this example even the empty trace is a false attack since the two processes fail to meet the requirement of having identical skeletons. Such extreme configurations are however unlikely to occur in practice: privacy is usually modelled as the equivalence of two protocol instances where some private attributes are changed. In particular the overall structure in parallel processes remains common to both sides. More realistic false attacks may arise when the structural requirements of equivalence by session are too strong, i.e., when matching the trace requires mixing actions from different sessions. Consider for example:

$$P = \overline{s}\langle n\rangle.\,\overline{a}\langle n\rangle \mid s(x).\,\overline{b}\langle n\rangle \qquad Q = \overline{s}\langle n\rangle.\,\overline{b}\langle n\rangle \mid s(x).\,\overline{a}\langle n\rangle$$

with $a, b \in \mathcal{F}_0$ and $s \in \mathcal{N}$. These processes first synchronise on a private channel s using an internal communication and then perform two parallel outputs on public channels a, b. They are easily seen to be trace equivalent. However the skeletons at toplevel constrain the session matchings, i.e., the application of rule (Match). Hence any trace executing an output on a or b is a false attack. This being said, we have a completeness result for determinate processes:

Definition 5.7 (determinate process)
A process P is *action determinate* if it verifies the following two properties:

1. *no private channels:* for all session traces $P \overset{\mathsf{tr}}{\Rightarrow}_{\mathsf{ss}} (\mathcal{P}, \Phi)$, if $R \in \bigcup_{P' \in \mathcal{P}} \mathsf{sessions}(P')$ with $\mathsf{skel}(R) \in \{\mathsf{in}_u, \mathsf{out}_u\}$, u is deducible from Φ;
2. *unambiguous traces:* if $P \overset{\mathsf{tr}}{\Rightarrow}_{\mathsf{ss}} A$, $P \overset{\mathsf{tr}}{\Rightarrow}_{\mathsf{ss}} B$ and $A \sim B$ then $A \approx_t B$.

This notion of determinacy is introduced in [CCD13] and subsumes the notion of *action*-determinacy used in the partial-order reductions of [BDH15]. We discuss in more details in Chapter 7 the different definitions of determinacy in the context of protocol analysis and their consequences on the decidability of equivalence properties [CCD13, BDH15, CKR19, BCK20].

Proposition 5.3 (completeness of equivalence by session for determinate processes)
If P, Q are determinate plain processes such that $P \approx_t Q$ then $P \approx_s Q$.

The core argument is the uniqueness of session matchings, that is, there is always one permutation (up to exchanging trace equivalent processes) that can be chosen when applying the rule (MATCH) to a pair of determinate processes. The proof can be found in Appendix C.

Relation to diff-equivalence ProVerif, Tamarin and Maude-NPA are semi-automated tools that can provide equivalence proofs for an unbounded number of protocol sessions. For that they rely on another refinement of trace equivalence, called diff-equivalence ($\approx_d$). It relies on a similar intuition as equivalence by session, adding (much stronger) structural requirements to proofs. To prove diff-equivalence of P and Q, one first requires that P and Q have *syntactically* the same structure and that they only differ by the data (i.e., the terms) inside the process. Second, any trace of P must be matched in Q by the trace that follows exactly the same control flow. Consider for example

$$P = \overline{c}\langle u\rangle \mid \overline{c}\langle v\rangle \mid R \qquad\qquad Q = \overline{c}\langle u'\rangle \mid \overline{c}\langle v'\rangle \mid R'$$

For P and Q to be diff-equivalent, traces of P starting with $\overline{c}\langle u\rangle$ need to be matched by traces of Q starting with $\overline{c}\langle u'\rangle$. In the original definition of diff-equivalence in [BAF08] the conditional branchings were also required to result into the same control-flow. This condition has however been relaxed within [CB13]: the resulting diff-equivalence can be defined in our formalism as equivalence by session where Rule (MATCH) only performs the identity matching.

Definition 5.8 (diff-equivalence)
We write $\mathbb{T}_d(P, Q)$ the subset of $\mathbb{T}(P, Q)$ by replacing Rule (MATCH) by the two rules

$$(\{\!\{(P_1 \mid \cdots \mid P_n, Q_1 \mid \cdots \mid Q_n)\}\!\} \cup \mathcal{P}^2, \Phi_0, \Phi_1) \xrightarrow{\tau}_{\mathsf{ss}^2} (\{\!\{(P_i, Q_i)\}\!\}_{i=1}^n \cup \mathcal{P}^2, \Phi_0, \Phi_1) \quad (\text{PAR}^2)$$

$$(\{\!\{(!\,P, !\,Q)\}\!\} \cup \mathcal{P}^2, \Phi_0, \Phi_1) \xrightarrow{\tau}_{\mathsf{ss}^2} (\{\!\{(P\{\!\!\downarrow\!\!\}, Q\{\!\!\downarrow\!\!\}), (!\,P, !\,Q)\}\!\} \cup \mathcal{P}^2, \Phi_0, \Phi_1) \quad (\text{REPL}^2)$$

Then we define $P \sqsubseteq_d Q$ as the statement

$$\forall t \in \mathbb{T}(P),\ \exists t^2 \in \mathbb{T}_d(P, Q),\ t = \mathsf{fst}(t^2) \text{ and } t \sim \mathsf{snd}(t^2)\,.$$

We say that P and Q are *diff-equivalent*, written $P \approx_d Q$, when $P \sqsubseteq_d Q$ and $Q \sqsubseteq_d P$.

Proposition 5.4 (relation between diff-equivalence and equivalence by session)
If $P \approx_d Q$ then $P \approx_s Q$.

This follows from the fact that $\mathbb{T}_d(P, Q) \subseteq \mathbb{T}(P, Q)$ by definition. The converse however

does not hold in general, typically,

$$P = \overline{c}\langle a\rangle \mid \overline{c}\langle b\rangle \qquad Q = \overline{c}\langle b\rangle \mid \overline{c}\langle a\rangle \qquad a, b \in \mathcal{F}_0 \text{ distinct}$$

This example is extreme as a pre-processing on parallel operators would make the processes diff-equivalent. Such a pre-processing is however not possible for more involved, real-world examples such as our models of unlinkability in the BAC protocol, or for the simpler example earlier mentioned at the end of Section 1.1. The reason is that the matchings have to be selected dynamically, that is, different session matchings are needed to match different traces.

Relation to labelled bisimilarity Just as equivalence by session, labelled bisimilarity is known to be an intermediate refinement between diff-equivalence and trace equivalence:

Proposition 5.5 (relation between diff, observational, and trace equivalences [CD09])
We have $\approx_d \subseteq \approx_l \subseteq \approx_t$. Besides, $\approx_l$ and $\approx_t$ coincide for determinate processes.

In particular by Proposition 5.3 we obtain that trace equivalence, labelled bisimilarity and equivalence by session coincide for determinate processes. However in general:

Proposition 5.6 (relation between equivalence by session and labelled bisimilarity)
The relations $\approx_s$ and $\approx_l$ are incomparable.

Proof. If we write $P = c(x).c(x)$ and $Q = c(x) \mid c(x)$, then $P \approx_l Q$ but $P \not\approx_s Q$. Besides if $a, b \in \mathcal{F}_0$ are distinct we have $R(a, b, b) \approx_s R(b, a, a)$ and $R(a, b, b) \not\approx_l R(b, a, a)$ where

$$\begin{aligned} R(t_0, t_1, t_2) &= \mathsf{new}\ k_0.\, \mathsf{new}\ k_1.\, \mathsf{new}\ k_2.\ (\overline{c}\langle k_0\rangle \mid \overline{c}\langle k_1\rangle \mid \overline{c}\langle k_2\rangle \mid S(t_0, t_1, t_2)) \\ S(t_0, t_1, t_2) &= c(x).\ \mathsf{if}\ x = k_0\ \mathsf{then}\ \overline{c}\langle t_0\rangle \\ &\qquad\qquad \mathsf{else\ if}\ x = k_1\ \mathsf{then}\ \overline{c}\langle t_1\rangle \\ &\qquad\qquad \mathsf{else\ if}\ x = k_2\ \mathsf{then}\ \overline{c}\langle t_2\rangle \end{aligned}$$

□

Proposition 5.7 (summary of the relations between all equivalences)
If $\approx \in \{\approx_l, \approx_s\}$ then $\approx_d \subsetneq \approx \subsetneq \approx_t$ and, for determinate processes, $\approx = \approx_t$.

2 Formalising optimisations

Section summary

In this section we introduce some theoretical notions that will help presenting all of our optimisations in a uniform and modular way. This includes a notion of *trace refinement* that formalises how to define and compose optimisations, and basic notions and notations of *group theory*.

2.1 Global assumptions

The optimisations we present are specific to the *bounded fragment* of the calculus, in the *private semantics*. For example one of our optimisations consists of always executing public outputs in

priority over other actions, which is unsound in presence or replication due to our formalisation (since Rule (Sess) may make infinitely-many public outputs available) or in the classical semantics (where the public outputs may be used later for an internal communication). We also make a *stability assumption on private channels*, which will ensure the preservation of skeleton equivalence within twin processes through transitions:

Definition 5.9 (stable channels)

A plain process P has *stable channels* if for all traces of P of the form $P \overset{\mathsf{tr}_1}{\Longrightarrow} (\{\!\{u(x).Q\}\!\} \cup \mathcal{P}_1, \Phi_1) \overset{\mathsf{tr}_2}{\Longrightarrow} (\{\!\{u(x).Q\}\!\} \cup \mathcal{P}_2, \Phi_2)$, if u is deducible from Φ_2 then it is deducible from Φ_1.

The notion of stable channels prevents private channels to become public, i.e., if a channel can be used for an internal communication, it will never be used for a public communication later. This does not exclude dynamically generated channels such as in $\mathsf{new}\ e.\,\overline{c}\langle e\rangle.\,P$.

2.2 Trace refinements

Definition We present an abstract notion of *optimisation*, based on trace refinements. This comes with several properties on how to compose them, providing a unified way of presenting different concrete optimisations for the decision of equivalence by session.

Definition 5.10 (optimisation)

An *optimisation* is a pair $\mathbb{O} = (\mathbb{O}^\forall, \mathbb{O}^\exists)$ with $\mathbb{O}^\forall$ a set of traces of extended processes (*universal optimisation*), and $\mathbb{O}^\exists$ a set of traces of extended twin processes (*existential optimisation*).

Intuitively, an optimisation reduces the set of traces that are considered when verifying equivalence: when proving $P \sqsubseteq_s Q$, only traces of $\mathbb{T}(P) \cap \mathbb{O}^\forall$ and $\mathbb{T}(P,Q) \cap \mathbb{O}^\exists$ will be considered. That is, we define the equivalence $\approx_{\mathbb{O}} = \sqsubseteq_{\mathbb{O}} \cap \sqsupseteq_{\mathbb{O}}$ where $P \sqsubseteq_{\mathbb{O}} Q$ means

$$\forall t \in \mathbb{T}(P) \cap \mathbb{O}^\forall,\ \exists t^2 \in \mathbb{T}(P,Q) \cap \mathbb{O}^\exists,\ t = \mathsf{fst}(t^2) \sim \mathsf{snd}(t^2)\,.$$

Typically $\approx_{\mathbb{O}_{\mathsf{all}}}$ is the equivalence by session on the bounded fragment, where $\mathbb{O}_{\mathsf{all}} = (\mathbb{O}^\forall_{\mathsf{all}}, \mathbb{O}^\exists_{\mathsf{all}})$ contains all traces of bounded processes. Also note that for a few of our refinements (Section 4), the definition of $\mathbb{O}^\forall$ may depend on both P and Q. Of course, using refinements may induce different notions of equivalence, hence the need for correctness arguments specific to each layer of optimisation. We specify this as follows: if $\mathbb{O}_\alpha = (\mathbb{O}^\forall_\alpha, \mathbb{O}^\exists_\alpha)$ and $\mathbb{O}_\beta = (\mathbb{O}^\forall_\beta, \mathbb{O}^\exists_\beta)$, $\mathbb{O}_\alpha$ is a *correct refinement* of $\mathbb{O}_\beta$ when

$$\mathbb{O}^\forall_\alpha \subseteq \mathbb{O}^\forall_\beta \quad \text{and} \quad \mathbb{O}^\exists_\alpha \subseteq \mathbb{O}^\exists_\beta \quad \text{and} \quad \approx_{\mathbb{O}_\alpha} = \approx_{\mathbb{O}_\beta}\,.$$

Properties The remainder of this section provides elementary properties useful when constructing, and composing optimisations. First, we can refine optimisations progressively:

Proposition 5.8 (transitivity)

If $\mathbb{O}_1$ is a correct refinement of $\mathbb{O}_2$, and $\mathbb{O}_2$ is a correct refinement of $\mathbb{O}_3$, then $\mathbb{O}_1$ is a correct refinement of $\mathbb{O}_3$.

Moreover, we can prove universal and existantial optimisations correct in a modular way:

Proposition 5.9 (combination)
If $(\mathbb{O}^{\forall}_{opt}, \mathbb{O}^{\exists})$ and $(\mathbb{O}^{\forall}, \mathbb{O}^{\exists}_{opt})$ are both correct refinements of $(\mathbb{O}^{\forall}, \mathbb{O}^{\exists})$, then $(\mathbb{O}^{\forall}_{opt}, \mathbb{O}^{\exists}_{opt})$ is a correct refinement of $(\mathbb{O}^{\forall}, \mathbb{O}^{\exists})$.

Proof. Let $\approx_{\times\times}$, $\approx_{\circ\times}$, $\approx_{\times\circ}$ and $\approx_{\circ\circ}$ denote the equivalences induced by the optimisations $(\mathbb{O}^{\forall}, \mathbb{O}^{\exists})$, $(\mathbb{O}^{\forall}_{opt}, \mathbb{O}^{\exists})$, $(\mathbb{O}^{\forall}, \mathbb{O}^{\exists}_{opt})$ and $(\mathbb{O}^{\forall}_{opt}, \mathbb{O}^{\exists}_{opt})$, respectively. As $\approx_{\circ\times} = \approx_{\times\times} = \approx_{\times\circ}$ by hypothesis, the result follows from the straightforward inclusions $\approx_{\circ\circ} \subseteq \approx_{\circ\times}$ and $\approx_{\times\circ} \subseteq \approx_{\circ\circ}$. □

Relying on this result, we see a universal optimisation $\mathbb{O}^{\forall}$ (resp. existential optimisations $\mathbb{O}^{\exists}$) as the optimisation $(\mathbb{O}^{\forall}, \mathbb{O}^{\exists}_{\mathsf{all}})$ (resp. $(\mathbb{O}^{\forall}_{\mathsf{all}}, \mathbb{O}^{\exists})$). This lightens presentation as we can now meaningfully talk about universal (resp. existential) optimisations being correct refinements of others. Finally, when implementing such optimisations in tools, deciding the membership of a trace in the sets $\mathbb{O}^{\forall}$ or $\mathbb{O}^{\exists}$ may sometimes be inefficient or not effective. In these cases we may implement these optimisations partially, using for example sufficient conditions. The following proposition states that such partial implementations still result into correct refinements.

Proposition 5.10 (partial implementability)
Let us consider the optimisations $\mathbb{O}^{\forall}_{opt} \subseteq \mathbb{O}^{\forall}_{part} \subseteq \mathbb{O}^{\forall}$ and $\mathbb{O}^{\exists}_{opt} \subseteq \mathbb{O}^{\exists}_{part} \subseteq \mathbb{O}^{\exists}$. If $\mathbb{O}^{\forall}_{opt}$ is a correct refinement of $\mathbb{O}^{\forall}$ and $\mathbb{O}^{\exists}_{opt}$ is a correct refinement of $\mathbb{O}^{\exists}$, then $(\mathbb{O}^{\forall}_{part}, \mathbb{O}^{\exists}_{part})$ is a correct refinement of $(\mathbb{O}^{\forall}, \mathbb{O}^{\exists})$.

2.3 Reminders of group theory

Our presentation and proofs can benefit from some group-theoretical terminology recalled here. The reader familiar with elementary results of group theory may skip this section.

Basic notions A *group* G is a set equipped with a binary operation $*$ verifying the following properties; there should exist an element $e \in G$ called the *identity* of G such that

$$\forall g \in G, g * e = e * g = g \qquad \text{(identity)}$$
$$\forall g \in G, \exists g' \in G, g' * g = g * g' = e \qquad \text{(invertibility)}$$
$$\forall g, g' \in G, g * g' \in G \qquad \text{(closure by composition)}$$
$$\forall g_1, g_2, g_3 \in G, (g_1 * g_2) * g_3 = g_1 * (g_2 * g_3) \qquad \text{(associativity)}$$

In particular the identity element is necessarily unique, as well as the inverse of an element g which can therefore be referred to as g^{-1}. By analogy with multiplicative notations, we sometimes write gg' instead of $g * g'$ and g^n instead of $g * \cdots * g$ (n times).

Definition 5.11 (symmetric group)
If E is a set, the set G of all permutations of E equipped with function composition $\circ$ is a group. If $E = [\![1, n]\!]$, G is called the *symmetric group* of order n and is written S_n.

In symmetric groups we often use *cycle notations*: $(a_1\ a_2\ \cdots\ a_p)$ refers to the permutation π such that for all $i \in [\![1, p]\!]$, $\pi(a_i) = a_{(i+1) \bmod p}$, and $\pi(a) = a$ if $a \in dom(\pi) \setminus \{a_1, \ldots, a_p\}$. In particular a 2-cycle $(a\ \ b)$ is called a *transposition*. Another key notion is *subgroups*: we say that G' is a subgroup of G if $G' \subseteq G$ and G and G' are two groups with the same operation. The following results are classical subgroup criteria:

Proposition 5.11 (generated subgroup)

If G is a group and $S \subseteq G$, we write $\langle S \rangle$ the intersection of all subgroups of G containing S. Then $\langle S \rangle$ is a subgroup of G, called the *subgroup of G generated by S*. It is the smallest non-empty subset of G containing S and that is closed under composition and inverse.

Proof. It is immediate that an intersection of subgroups is also a subgroup. Then let S' be a non-empty subset of G containing S that is closed under composition and inverse and let us show that $\langle S \rangle \subseteq S'$. We know that S' contains the identity of G because if we let $x \in S'$, we have by hypothesis $x^{-1} \in S'$ and then $xx^{-1} \in S'$. In particular S' is a subgroup of G, hence $\langle S \rangle \subseteq S'$ by definition. □

Proposition 5.12 (subgroup of finite order)

Let G be a group and S a non-empty, finite subset of G that is closed under composition. Then S is a subgroup of G.

Proof. It sufficies to prove that S is closed by inverse (and it will ensure that the identity of G is in S as well). Since S is closed under composition, if $x \in S$ then $x^n \in S$ for all $n > 1$. Besides S is also finite, hence there exists $i > j$ such that $x^i = x^j$. In particular by multiplying by the inverse of x^j in G, we obtain that x^{i-j-1} is the inverse of x in S. □

Group actions An *action* of a group G on a set X is an application $G \times X \to X$, written using a "dot" notation (the image of (g, x) is written $g.x$). A group action is required to satisfy:

$$\forall x \in X, e.x = x \qquad \text{(identity)}$$
$$\forall g, g' \in G, \forall x \in X, g.(g'.x) = (gg').x \qquad \text{(compatibility)}$$

A typical example that we will consider in this thesis is the action of the symmetric group on words of actions, writing $\pi.(\alpha_1 \cdots \alpha_n) = \alpha_{\pi(1)} \cdots \alpha_{\pi(n)}$. We will use two related notions:

Definition 5.12 (stabiliser)

Let G a group action on X and $\equiv$ an equivalence relation on X compatible with this action, in the sense that for all $g \in G$, if $x \equiv y$ then $g.x \equiv g.y$. If $x \in X$, the *stabiliser* of x quotiented by $\equiv$ is the set $\mathsf{Stab}(x) = \{g \in G \mid g.x \equiv x\}$.

Proposition 5.13 (stabiliser subgroup)

With the notations of the definition, $\mathsf{Stab}(x)$ is a subgroup of G.

Proof. It is immediate that $\mathsf{Stab}(x)$ contains e the identity of G. It is also closed by composition: if $g.x \equiv x$ implies $g'.x \equiv x$, then by hypothesis on $\equiv$ we have $g'.g.x \equiv g'.x \equiv x$, hence $gg' \in \mathsf{Stab}(x)$. The conclusion then follows from Proposition 5.12. □

Definition 5.13 (orbit)

If G is a group action on X and $x \in X$, the *orbit* of x in G is the set $\mathsf{Orb}(x) = \{g.x \mid g \in G\}$.

3 Partial-order reductions

Section summary

We present partial-order reductions for equivalence by session and prove their *correctness* (without restriction to determinate processes). While some are directly inspired from the previous work of [BDH15], several of them are novel (in particular regarding the treatment of *private channels*).

Some partial-order reductions we present in this section are inspired by similar techniques developed for proving trace equivalence of determinate processes [BDH15], although they differ in their technical development to preserve correctness in our more general setting.

3.1 Labels and independence

Labels Partial-order reduction techniques identify commutativity relations in a set of traces and factor out the resulting redundancy. Here we exploit the permutability of concurrent actions without output-input data flow. For that we introduce *labels* to reason about dependencies in the execution:

1. Plain processes P are labelled $[\,P\,]^{\ell}$, with ℓ a word of integers reflecting the position of P within the whole process.
2. Actions α are labelled $[\,\alpha\,]^{L}$ to reflect the label(s) of the process(es) they originate from. That is, L is either a single integer word ℓ (for inputs and outputs) or a pair of such, written $\ell_1 \mid \ell_2$ (for internal communications).

Labels are bootstrapped by the empty word ε and are propagated in session traces by assigning incomparable labels to each new session when applying Rule (Sess), and preserving the label when applying Rule (IO). Formally the relation $\xrightarrow{\alpha}_{\mathsf{ss}}$ is lifted to labelled processes as follows; we recall that we only operate on bounded processes, hence the simpler shape of (Sess):

$$\text{Rule (Sess):} \quad (\{\!|[\,P_1 \mid \cdots \mid P_n\,]^{\ell}|\!\} \cup \mathcal{P}, \Phi) \xrightarrow{[\,\tau\,]^{\ell}}_{\mathsf{ss}} (\{\!|[\,P_i\,]^{\ell.i}|\!\}_{i=1}^{n} \cup \mathcal{P}, \Phi)$$

$$\text{Rule (IO), inputs / outputs:} \quad (\{\!|[\,P\,]^{\ell}|\!\} \cup \mathcal{P}, \Phi) \xrightarrow{[\,\alpha\,]^{\ell}}_{\mathsf{ss}} (\{\!|[\,P'\,]^{\ell}|\!\} \cup \mathcal{P}, \Phi')$$

$$\text{Rule (IO), internal comm.:} \quad (\{\!|[\,P\,]^{\ell}, [\,Q\,]^{\ell'}|\!\} \cup \mathcal{P}, \Phi) \xrightarrow{[\,\tau\,]^{\ell\mid\ell'}}_{\mathsf{ss}} (\{\!|[\,P'\,]^{\ell}, [\,Q'\,]^{\ell'}|\!\} \cup \mathcal{P}, \Phi)$$

In particular, we always implicitly assume the invariant preserved by transitions that extended processes contain labels that are pairwise incomparable w.r.t. the prefix ordering.

Independence Labels are used to materialise flow dependencies. Two actions $\alpha = [\,a\,]^{L}$ and $\alpha' = [\,a'\,]^{L'}$ are said *sequentially dependent* if one of the (one or two) words constituting L, and one of those constituting L', are comparable w.r.t. the prefix ordering. Regarding input-output dependencies, we say that α and α' are *data dependent* when $\{a, a'\} = \{\overline{\xi}\langle \mathsf{ax}\rangle, \xi_c(\xi_t)\}$ with ax appearing in ξ_c or ξ_t. The two notions combine into the following property:

Definition 5.14 (independence)

Two actions α and α' are said to be *independent*, written $\alpha \parallel \alpha'$, when they are sequentially independent and data independent.

We identify some redundancy in a set of traces when, intuitively, swapping adjacent independent actions in a trace has no substantial effect. Still, this is a rather weak notion: for

example the recipe $\mathsf{fst}(\langle n, \mathsf{ax}\rangle)$ is artificially dependent in the axiom ax, which would prevent optimisations from applying. Such spurious dependencies can be erased using the following equivalence notion:

Definition 5.15 (recipe equivalence)

Two input transitions $A \xrightarrow{[\xi_1(\zeta_1)]^\ell}_{\mathsf{ss}} B$ and $A \xrightarrow{[\xi_2(\zeta_2)]^\ell}_{\mathsf{ss}} B$ are said to be *recipe equivalent* when $\xi_1\Phi(A) =_E \xi_2\Phi(A)$ and $\zeta_1\Phi(A) =_E \zeta_2\Phi(A)$. Two traces are recipe equivalent if one can be obtained from the other by replacing some transitions by recipe-equivalent ones.

The rest of this section formalises the intuition that equivalence by session can be studied up to transformation of traces into recipe-equivalent ones, and arbitrary permutation of their independent actions. Technical proofs can be found in Appendix C.

Correctness of por techniques If $\mathsf{tr} = \alpha_1 \cdots \alpha_n$ and π is a permutation of $[\![1, n]\!]$, we write $\pi.\mathsf{tr} = \alpha_{\pi(1)} \cdots \alpha_{\pi(n)}$. This is an action of S_n the symmetric group of order n on the set of action words of length n. We say that π *permutes independent actions* of tr if either $\pi = id$, or $\pi = \pi_0 \circ (i \;\; i{+}1)$ with $\alpha_i \parallel \alpha_{i+1}$ and π_0 permutes independent actions of $(i \;\; i{+}1).\mathsf{tr}$. Although not a subgroup of S_n in a strict sense, permutations of independent actions have some sorts of group-like properties:

Proposition 5.14 (inversion and composition)

If π permutes independent actions of tr, π^{-1} permutes independent actions of $\pi.\mathsf{tr}$. Moreover if π' permutes independent actions of $\pi.\mathsf{tr}$, $\pi' \circ \pi$ permutes independent actions of tr.

We will use these two properties implicitly in most of our proofs. But more importantly, the action of the symmetric group can be uplifted to an action on traces:

Proposition 5.15 (trace permutability)

If $t : A \overset{\mathsf{tr}}{\Rightarrow}_{\mathsf{ss}} B$ and π permutes independent actions of tr, then $A \overset{\pi.\mathsf{tr}}{\Longrightarrow}_{\mathsf{ss}} B$. This trace is unique if we take labels into account, and will be referred as $\pi.t$.

Together with recipe equivalence, this is the core notion for defining partial-order reductions. We gather them into $\equiv_{\mathsf{por}}$ the smallest equivalence relation over traces containing recipe equivalence and such that $t \equiv_{\mathsf{por}} \pi.t$ for all permutations π of independent actions of t. The key result below justifies that any optimisation that discards traces t when there exists another trace $t' \equiv_{\mathsf{por}} t$ is correct. The statement is even slightly more general by allowing to discard t if t' is equivalent to an extension t_{ext} of t.

Theorem 5.16 *(correctness of por)*

Let $\mathbb{O}_1^\forall \subseteq \mathbb{O}_2^\forall$ be universal optimisations. We assume that for all $t \in \mathbb{O}_2^\forall$, there exists t_{ext} such that t is a prefix of t_{ext}, and $t' \in \mathbb{O}_1^\forall$ such that $t' \equiv_{\mathsf{por}} t_{ext}$. Then $\mathbb{O}_1^\forall$ is a correct refinement of $\mathbb{O}_2^\forall$.

3.2 Compression optimisations

We first present a compression of traces into blocks of actions of a same type (inputs, outputs and parallel, or internal communications) by exploiting Theorem 5.16. The optimisation is a generalisation of the compressed traces of [BDH15]. We formalise this idea by using reduction strategies based on *polarity* patterns.

Polarities and phases We assign polarities to processes depending on their toplevel actions: public inputs are positive (+1), public outputs and parallels are overwhelmingly negative ($-\infty$), and others are null.

$$\begin{array}{rrl} \mathit{polar}_\Phi(u(x).P) = 1 & \mathit{polar}_\Phi(\overline{u}\langle v\rangle.P) = -\infty & u \text{ deducible from } \Phi \\ \mathit{polar}_\Phi(u(x).P) = 0 & \mathit{polar}_\Phi(\overline{u}\langle v\rangle.P) = 0 & u \text{ not deducible from } \Phi \\ \mathit{polar}_\Phi(0) = 0 & \mathit{polar}_\Phi(P \mid Q) = -\infty & \end{array}$$

Note that the assumption of channel stability ensures executing an instruction, thus potentially increasing the frame, does not change the polarity of the other processes in parallel. This notion is lifted to extended processes by summing:

$$\mathit{polar}(\mathcal{P}, \Phi) = \sum_{R \in \mathcal{P}} \mathit{polar}_\Phi(R)\,.$$

In particular, extended processes containing an executable parallel operator or public output has polarity $-\infty$, and executing public inputs makes polarity nonincreasing. We then identify the trace patterns at the core of our partial-order reductions. We say that a session trace

$$t : A_0 \xrightarrow{[a_1]^{L_1}}_{\mathsf{ss}} \cdots \xrightarrow{[a_n]^{L_n}}_{\mathsf{ss}} A_n$$

- is a *negative phase* when all transitions are outputs or parallels, and $\mathit{polar}(A_n) \neq -\infty$.
- is a *null phase* when $\mathit{polar}(A_0) \geqslant 0$, $n = 1$ and the transition is an internal communication.
- is a *positive phase* when $\mathit{polar}(A_0) > 0$, all transitions are inputs, all labels L_i are identical, and $\mathit{polar}(A_0) > \mathit{polar}(A_n)$.

Rephrasing, a negative phase executes all available outputs and parallels, a null phase is one internal communication, and a positive phase executes a maximal non-empty chain of inputs.

Basic compression The first optimisation is to only consider traces that can be decomposed into phases. Formally we write $\mathbb{O}^\forall_{\mathsf{c},b}$ the set of traces of the form

$$t : b_0^- \cdot b_1^+ \cdot b_1^- \cdot b_2^+ \cdot b_2^- \cdots b_n^+ \cdot b_n^-$$

where each b_i^+ is a positive or null phase, and each b_i^- is a negative phase. We show in Appendix C that any maximal trace can be decomposed this way after application of a well-chosen permutation of independent actions. Hence by Theorem 5.16:

Proposition 5.17 (correctness of basic compression)
$\mathbb{O}^\forall_{\mathsf{c},b}$ is a correct refinement of $\mathbb{O}^\forall_{\mathsf{all}}$.

▶ *Example 5.2: running example*
Let us introduce a running toy example to illustrate what kind of traces are discarded by the different optimisations in this chapter. This will give an intuition of why they often reduce the number of symbolic traces by an exponential factor.

Let us consider again the example of the BAC protocol with an implicit key distribution and no getChallenge messages. For that we define the two processes

$$\begin{aligned}\mathsf{Passport}'(k) = \mathsf{new}\ n.\ \overline{c}\langle n\rangle.\ c(x).\\ \mathsf{if}\ \mathsf{rsdec}(x,k) = n\ \mathsf{then}\ \overline{c}\langle\mathsf{ok}\rangle\\ \mathsf{else}\ \overline{c}\langle\mathsf{error}\rangle\end{aligned} \qquad \begin{aligned}\mathsf{Reader}'(k) = d(x).\\ \mathsf{new}\ r.\ \overline{d}\langle\mathsf{rsenc}(x,r,k)\rangle\end{aligned}$$

We study the traces of the bounded process $\mathsf{System} = !^n\,\mathsf{new}\ k.\ !^p\,\mathsf{Reader}'(k) \mid !^p\,\mathsf{Passport}'(k)$. This first optimisation imposes to follow a reduction strategy executing public outputs in priority; typically all of the np outputs of the various instances of $\mathsf{Passport}'$ will be executed first, without interleaving actions from Reader'. Moreover, after these outputs have been executed, the remaining two actions of each instance of $\mathsf{Passport}'$ and Reader' are executed in one bloc without interleaving other concurrent actions. All only considering traces respecting basic compression reduces the number of interleaving by an exponential factor in np. ◀

Determinism of negative phases Negative phases are non-deterministic by essence, but the underlying combinatorial explosion is artificial in that most of the actions within negative phases are independent: we show that they can actually be executed purely deterministically. We fix an arbitrary total ordering $\preccurlyeq$ on labelled actions. A negative phase b^-, with $\mathit{actions}(b^-) = \alpha_1 \cdots \alpha_n$, is said to be *consistent* when for all $i < n$ such that $\alpha_i \parallel \alpha_{i+1}$, we have $\alpha_i \preccurlyeq \alpha_{i+1}$. We write $\mathbb{O}^{\forall}_{\mathsf{c}}$ the subset of $\mathbb{O}^{\forall}_{\mathsf{c},b}$ of traces whose negative phases are all consistent.

Proposition 5.18 (correctness of compression)
$\mathbb{O}^{\forall}_{\mathsf{c}}$ is a correct refinement of $\mathbb{O}^{\forall}_{\mathsf{c},b}$.

Proof. By Theorem 5.16, it sufficies to prove that for all negative phases b^-, there exists π permuting independent actions of b^- such that $\pi.b^-$ is consistent. This follows from a well-founded induction on $\mathit{actions}(b^-)$ with respect to the lexicographic extension of $\preccurlyeq$ on words of actions. □

▶ *Example 5.3*
Consider again the process System introduced in the running example (Example 5.2), this optimisation fixes the order of the np initial outputs of the trace, reducing again the number of interleavings by an exponential factor. ◀

Blocks We have established that, to prove equivalences by session, it is sufficient to consider traces of a certain shape, namely, traces alternating between nonnegative and negative phases. The rest of our partial-order reductions reason at the granularity of these phases, by reordering so-called *blocks*. A block is a positive or null phase followed by a negative phase. Any trace of $\mathbb{O}^{\forall}_{\mathsf{c}}$ is therefore composed of an initial negative phase and a sequence of blocks. Two blocks b and b' are said *independent*, written $b \parallel b'$, if all actions of the former are independent of all actions of the latter. Analogously to actions, we refer to permutations π permuting independent blocks of traces of $\mathbb{O}^{\forall}_{\mathsf{c}}$. The link between the two notions is given by:

Proposition 5.19 (block-to-action permutation conversion)
Let $t : b_p \cdot \cdots \cdot b_n$ a sequence of blocks, $\mathsf{tr}_i = \mathit{actions}(b_i)$. If π permutes independent blocks of $\mathsf{tr} = \mathit{actions}(t)$, then there is π' permuting independent actions of tr s.t.

$$\pi'.\mathsf{tr} = \pi.\mathsf{tr} = \mathsf{tr}_{\pi(p)} \cdot \cdots \cdot \mathsf{tr}_{\pi(n)}\,.$$

Note in particular the following corollary of Theorem 5.16 that will be at the core of the results of the next sections, where $\equiv_{\mathsf{b\text{-}por}}$ is the analogue of $\equiv_{\mathsf{por}}$ where permutation of independent actions is replaced by permutation of independent blocks:

Theorem 5.20 *(correctness of por: block version)*
Let $\mathbb{O}_1^\forall \subseteq \mathbb{O}_2^\forall \subseteq \mathbb{O}_\mathsf{c}^\forall$. We assume that for all $t \in \mathbb{O}_2^\forall$, there exists $t_{ext} \in \mathbb{O}_\mathsf{c}^\forall$ such that t is a prefix of t_{ext}, and $t' \in \mathbb{O}_1^\forall$ such that $t' \equiv_{\mathsf{b\text{-}por}} t_{ext}$. Then $\mathbb{O}_1^\forall$ is a correct refinement of $\mathbb{O}_2^\forall$.

3.3 Improper positive phases

We now introduce *improper blocks*: intuitively, they conclude the execution of a process but do not bring new knowledge to the attacker. Such blocks can always be relegated to the end of traces because they are not essential to execute other blocks. This takes inspiration from the improper blocks of [BDH15] despite some technical differences.

Definition 5.16 (improper block)
We say that a block $b : (\mathcal{P}, \Phi) \overset{\mathsf{tr}}{\Rightarrow}_{\mathsf{ss}} (\mathcal{Q}, \Phi \cup \{\mathsf{ax}_1 \mapsto t_1, \ldots, \mathsf{ax}_n \mapsto t_n\})$ is *improper* if
1 it starts with an input, i.e., tr is of the form $\xi_c(\xi_t) \cdot \mathsf{tr}'$; and
2 all labels appearing in tr do not appear in $\mathcal{Q}$, except maybe on null processes; and
3 for all $i \in [\![1, n]\!]$, t_i is deducible from Φ.

Formally we write $\mathbb{O}_{\mathsf{c+i}}^\forall$ the subset of $\mathbb{O}_\mathsf{c}^\forall$ of traces not containing an improper block followed by a proper block.

▶ *Example 5.4*
Let us consider again the running example System. The notion takes inspiration, but generalises, the notion of improper blocks used in [BDH15, CKR19] that requires $n = 0$. Our more general notion of improper block captures for example
1 the block of any $\mathsf{Passport}'(k)$, since the error codes $\mathsf{error}, \mathsf{ok}$ are public values;
2 the block of $\mathsf{Reader}'(k)$ assuming a previous instance of $\mathsf{Reader}'(k)$ has already been executed with the same input x. ◀

Let us comment on the first item of the above definition discarding improper blocks starting with an internal communication: although the optimisation would have been correct without this restriction, this is not the case anymore when combined with those of the next sections. They advance the execution of some blocks starting with an internal communication: we shall therefore ensure that such blocks are never deemed improper since, as such, they would also have to be delayed, yielding a deadlock. The following results, proved in Appendix C, justify the correctness of the refinement.

Proposition 5.21 (correctness criterion)
For all $t \in \mathbb{O}^{\forall}_{\mathsf{c}}$, there exists $t' \equiv_{\mathsf{b\text{-}por}} t$ such that $t' \in \mathbb{O}^{\forall}_{\mathsf{c+i}}$ and t' has the same number of improper blocks t.

The fact that t' has the same number of improper blocks as t is not necessary to apply Proposition 5.20, but it will help establishing further optimisations on improper blocks in the next section.

Corollary 5.22 (correctness of improper-block delay)
$\mathbb{O}^{\forall}_{\mathsf{c+i}}$ is a correct refinement of $\mathbb{O}^{\forall}_{\mathsf{c}}$.

Note that when restricting the definition to $n = 0$, we obtain a weaker optimisation than the one presented in [BDH15] for determinate processes. The latter indeed considers traces with at most one improper block. This, however, relies on determinate-specific arguments that are unsound for equivalence by session in general.

3.4 Stabilising proper blocks

We introduce in this section a refinement $\mathbb{O}^{\forall}_{\mathsf{c+i}^*} \subseteq \mathbb{O}^{\forall}_{\mathsf{c+i}}$ of improper blocks. It is not included in DeepSec as we found it quite hard to implement; it should rather be seen as a convenient proof tool in the sense that the correctness of refinements of $\mathbb{O}^{\forall}_{\mathsf{c+i}^*}$ is easier to prove than the correctness of refinements of $\mathbb{O}^{\forall} \cap \mathbb{O}^{\forall}_{\mathsf{c+i}}$. Intuitively, it solves the problem that properness is not stable by permutation of independent blocks. For example consider processes built from signatures $\mathcal{F} = \{\mathsf{h}/1,\ a, c \in \mathcal{F}_0,\ k \in \mathcal{N}$, and

$$P = c(x).\overline{c}\langle \mathsf{h}(k)\rangle \mid c(x).\overline{c}\langle k\rangle$$

The traces of P can contain two blocks, one outputting $\mathsf{h}(k)$ and one outputting k: the former may be improper or not depending on whether it is executed second or first. This highlights that permutations of independent, *proper* blocks of a trace $t \in \mathbb{O}^{\forall}_{\mathsf{c+i}}$ may lead to a trace $\pi.t \notin \mathbb{O}^{\forall}_{\mathsf{c+i}}$. To get around this issue we let $\mathbb{O}^{\forall}_{\mathsf{c+i}^*}$ be the set of traces of the form $u \cdot v$ where

- $u, v \in \mathbb{O}^{\forall}_{\mathsf{c}}$;
- for all π permuting independent blocks of u, $\pi.u$ does not contain any improper blocks;
- v only contains improper blocks.

▶ *Example 5.5*
If we consider the processes of the running example, if we let

$$S = \mathsf{new}\ k.(c(x).\overline{c}\langle k\rangle \mid \mathsf{Passport}'(k) \mid \mathsf{Reader}'(k))$$

then executing $\mathsf{Reader}'(k)$ results in a proper block. However it becomes improper if we execute the block $c(x).\overline{c}\langle k\rangle$ first, and the optimisation will therefore only consider traces that do so, discarding one more interleeaving in this example. ◀

In particular this optimisation has the important property of being stable under permutation of proper blocks:

Proposition 5.23 (stability of the refinement under permutation)
Let $t \in \mathbb{O}^{\forall}_{\mathsf{c+i}^*}$ with $t = u \cdot v$ and u containing only proper blocks and v, only improper blocks. Then for all π permuting independent blocks of u, $(\pi.u \cdot v) \in \mathbb{O}^{\forall}_{\mathsf{c+i}^*}$.

We then prove the correctness of this optimisation.

Proposition 5.24 (correctness criterion)
For all $t \in \mathbb{O}^{\forall}_{\mathsf{c+i}}$, there exists $t' \equiv_{\mathsf{b\text{-}por}} t$ such that $t' \in \mathbb{O}^{\forall}_{\mathsf{c+i}^*}$.

Proof. We proceed by induction on the number of proper blocks of t. If $t \in \mathbb{O}^{\forall}_1$ (which subsumes the case where t has no proper blocks) it sufficies to choose $t' = t$. Otherwise let us write $t = u \cdot v$ with u containing only proper blocks and v improper blocks, and let π permuting independent blocks of u such that $\pi.u$ contains an improper block. By Proposition 5.21 there exists $u' \equiv_{\mathsf{b\text{-}por}} \pi.u$ such that $u' \in \mathbb{O}^{\forall}_{\mathsf{c+i}}$ and u' contains the same number of improper blocks as $\pi.u$. In particular if we write $s = u' \cdot v$, we have $s \in \mathbb{O}^{\forall}_{\mathsf{c+i}}$, s has less proper blocks than t, and $s \equiv_{\mathsf{b\text{-}por}} t$. Hence the conclusion by induction hypothesis applied to s. □

Corollary 5.25 (correctness of stabilised improper blocks)
$\mathbb{O}^{\forall}_{\mathsf{c+i}^*}$ is a correct refinement of $\mathbb{O}^{\forall}_{\mathsf{c+i}}$.

3.5 High-priority null phases

We introduce a condition for prioritising the execution of some internal communications (whose correctness therefore heavily relies on the assumption of stable channels). Consider for example

$$c(x).P_1 \mid \overline{d}\langle u\rangle.P_2 \mid \overline{d}\langle v\rangle.P_3 \mid d(x).P_4$$

where $c \in \mathcal{F}_0$ and $d \in \mathcal{N}$. Assuming that the channel d does not appear in P_1, all (maximal) traces will contain an internal communication on d between the rightmost three processes, regardless of the potential prior execution of $c(x).P_1$. Our optimisation will typically execute such internal communications in priority.

Definition 5.17 (high-priority channel)
Let $A = (\{\!\{[\,P_1\,]^{\ell_1}, \ldots, [\,P_n\,]^{\ell_n}\}\!\}, \Phi)$ be an extended process with $polar(A) \geqslant 0$. If $d \in \mathcal{N}$ is a private channel, we write $\mathbb{L}_A(d) \subseteq \{\ell_1, \ldots, \ell_n\}$ the set of labels ℓ_i such that P_i starts with an input or output on d. When an internal communication is effectively possible on $d \in \mathcal{N}$, we say that d is *high-priority* (in A) if for all session traces $A \overset{\mathsf{tr}}{\Rightarrow}_{\mathsf{ss}} B$ such that no labels of $\mathbb{L}_A(d)$ appear in tr, we have $\mathbb{L}_A(d) = \mathbb{L}_B(d)$.

This formalises the property that, in all traces of A, the first internal communication on d needs to be between an input and an output that were already available in A. By extension, a transition is high-priority when it is an internal communication on the minimal high-priority channel w.r.t. an arbitrary total ordering $\preccurlyeq_{ch}$ on channels. Rephrasing, it is a null phase on one deterministically-chosen, high-priority channel.

We define $\mathbb{O}^{\forall}_{0}$ to be the subset of traces of $\mathbb{O}^{\forall}_{\mathsf{c}}$ that do not contain non-high-priority transitions from processes a high-priority transition is possible from. To combine it with the previous optimisations we let $\mathbb{O}^{\forall}_{\mathsf{c+i^*+0}}$ be the set of traces of the form

$$b^{-} \cdot t_p \cdot t_i \ \in \mathbb{O}^{\forall}_{\mathsf{c+i^*}}$$

where b^{-} is a negative phase, $t_p \in \mathbb{O}^{\forall}_{0}$ only contains proper blocks, and $t_i \in \mathbb{O}^{\forall}_{\mathsf{c}}$ only contains improper blocks. The following propositions state the correctness of this optimisation; their proofs are detailed in Appendix C and rely as usual on Proposition 5.20.

Proposition 5.26 (correctness criterion)
Let $t = u \cdot v$ a maximal trace, where $u, v \in \mathbb{O}^{\forall}_{\mathsf{c}}$ and v does not contain any high-priority transitions. Then there exists π permuting independent blocks of u such that $\pi.u \in \mathbb{O}^{\forall}_{0}$.

Corollary 5.27 (correctness of high-priority transitions)
$\mathbb{O}^{\forall}_{\mathsf{c+i^*+0}}$ is a correct refinement of $\mathbb{O}^{\forall}_{\mathsf{c+i^*}}$.

3.6 Reduction of independent blocks

Finally, as sequences of independent blocks can be permuted arbitrarily, we define an optimisation that fixes their order. For that we let $\preccurlyeq$ be an ordering on words of actions such that two words of independent actions are always strictly comparable ($\prec$). We then define a predicate $\mathsf{Minimal}(t, b)$, ensuring that it is not possible to obtain an action word lexicographically-smaller than $\mathit{actions}(t \cdot b)$ by inserting b inside t using permutations of independent blocks.

Definition 5.18 (lexicographic minimality)
We say that b *can follow* b_n when either of the following conditions is met:
(a) $\mathit{actions}(b_n) \prec \mathit{actions}(b)$; or
(b) b_n starts with a high-priority transition but not b; or
(c) both b_n and b start with a high-priority transition, but on different private channels.
Given some blocks $b_1, \ldots, b_n, b$ we then write $\mathsf{Minimal}(b_1 \cdots b_n, b)$ when either 1 $n = 0$; or 2 $\neg(b_n \parallel b)$; or 3 b can follow b_n and $\mathsf{Minimal}(b_1 \cdots b_{n-1}, b)$.

Intuitively, we say that a block b can follow an other block b_n when $\mathit{actions}(b_n) \prec \mathit{actions}(b)$, but the test may be ignored in some cases to ensure compatibility with prior optimisations. We then define $\mathbb{O}^{\forall}_{\mathsf{r}}$ to be the smallest subset of $\mathbb{O}^{\forall}_{\mathsf{c}}$ containing the empty trace, and such that $t \in \mathbb{O}^{\forall}_{\mathsf{r}}$ and $\mathsf{Minimal}(t, b)$ implies $t \cdot b \in \mathbb{O}^{\forall}_{\mathsf{r}}$. To account for improper blocks, we let $\mathbb{O}^{\forall}_{\mathsf{por}}$ the set of traces

$$b^{-} \cdot t_p \cdot t_i \ \in \mathbb{O}^{\forall}_{\mathsf{c+i^*+0}}$$

where b^{-} is a negative phase, $t_p \in \mathbb{O}^{\forall}_{\mathsf{r}}$ only contains proper blocks, and $t_i \in \mathbb{O}^{\forall}_{\mathsf{r}}$ only contains improper blocks.

▶ *Example 5.6*
Consider again the running example System. This process contains np instances of the process $\mathsf{Passport}'(k)$ for various values of k. Let us number them with labels from 1 to np. After

executing the np initial outputs (one for each process), all remaining actions of these processes form np improper blocks that are therefore independent: this optimisation thus allows us to execute them in increasing label order. ◀

The correctness of this optimisation is stated below and proved in Appendix C.

Proposition 5.28 (correctness criterion)
For all maximal traces $t \in \mathbb{O}^{\forall}_{\mathsf{c+i^*+0}}$, there exists π permuting independent blocks of t such that $\pi.t \in \mathbb{O}^{\forall}_{\mathsf{por}}$. Besides $\pi.t \prec_{lex} t$ if $t \notin \mathbb{O}^{\forall}_{\mathsf{por}}$, with $\preccurlyeq_{lex}$ the lexicographic extension of $\preccurlyeq$.

The decreasing argument w.r.t. $\preccurlyeq_{lex}$ is not necessary to obtain correctness, but will be important to prove the compatibility with the reduction by symmetry of the next section.

Corollary 5.29 (correctness of lexicographic reduction)
$\mathbb{O}^{\forall}_{\mathsf{por}}$ is a correct refinement of $\mathbb{O}^{\forall}_{\mathsf{c+i^*+0}}$.

4 Reductions by symmetry

> ***Section summary***
> In practice we analyse processes that consist of *several parallel copies of a same session*, which induces a lot of redundancy in the trace space. We develop and prove correct some optimisations that factor this redundancy out for equivalence by session.

Symmetries often appear during practical verification: verifying multiple sessions of a same protocol results into parallel copies of almost identical processes. We first provide a group-theoretical characterisation of internal process redundancy and then design two optimisations. In spirit, this approach shares some similarities with classical work in model checking modelling symmetries by the group of the automorphisms of the system to be analysed [ES96].

4.1 Structural equivalence

We exhibit an equivalence relation identifying processes with an identical structure and whose data are equivalent w.r.t. the equational theory and alpha-renaming of private names. This will be the basis of our symmetry-based refinements. We define the relation $\equiv_{\mathsf{ac}}$ on plain processes as the smallest equivalence relation such that

$$P \mid Q \equiv_{\mathsf{ac}} Q \mid P \qquad\qquad (P \mid Q) \mid R \equiv_{\mathsf{ac}} P \mid (Q \mid R)$$

and that is closed under context (that is, composition of equivalent processes with either a same process in parallel, or an input, output, or conditional instruction at toplevel). To account for the equational theory, we extend it to $\equiv_E$ defined by the inference rule:

$$\frac{\sigma, \sigma' \text{ substitutions} \qquad P \equiv_{\mathsf{ac}} Q \qquad \sigma =_E \sigma'}{P\sigma \equiv_E Q\sigma'}$$

Moreover we extend the relation to capture alpha equivalence of private names and channels; intuitively, two agents executing the same protocol are behaving similarly even though they use

their own session nonces. Formally we define *structural equivalence* $\equiv$ on extended processes by the following inference rule

$$\frac{\forall i, P_i \equiv_E Q_i \qquad \varrho \text{ permutation of } \mathcal{N}}{(\{\!\{[\,P_1\,]^{\ell_1}, \ldots, [\,P_n\,]^{\ell_n}\}\!\}, \Phi) \equiv (\{\!\{[\,Q_1\,]^{\ell_1}, \ldots, [\,Q_n\,]^{\ell_n}\}\!\}, \Phi)\ \varrho}$$

4.2 Group actions and process redundancy

Let a labelled extended process (as defined in Section 3.1) and a labelled extended twin process

$$A = (\{\!\{[\,P_1\,]^{\ell_1}, \ldots, [\,P_n\,]^{\ell_n}\}\!\}, \Phi) \qquad \ell_1 < \cdots < \ell_n$$
$$A^2 = (\{\!\{([\,P_1\,]^{\ell_1}, Q_1), \ldots, ([\,P_n\,]^{\ell_n}, Q_n)\}\!\}, \Phi_0, \Phi_1)$$

where $<$ is an arbitrary total ordering on labels; the roles of these labels and this ordering is only to fix the position of processes in the multiset. They will often be omitted for succinctness, assuming an implicit ordering.

Definition 5.19 (action of the symmetric group on processes)

Using the above notations, if $\pi \in S_n$ we define the following two group actions of S_n:

$$\pi.A = (\{\!\{[\,P_{\pi(1)}\,]^{\ell_1}, \ldots, [\,P_{\pi(n)}\,]^{\ell_n}\}\!\}, \Phi)$$
$$\pi.A^2 = (\{\!\{([\,P_{\pi(1)}\,]^{\ell_1}, Q_1), \ldots, ([\,P_{\pi(n)}\,]^{\ell_n}, Q_n)\}\!\}, \Phi_0, \Phi_1)$$

The notation $\pi.A^2$ is only well defined if for all i, $P_{\pi(i)}$ and Q_i have equivalent skeletons w.r.t. Φ_0 and Φ_1; or, equivalently, if Q_i and $Q_{\pi(i)}$ have equivalent skeletons w.r.t. Φ_1.

Process redundancy within an extended process is then simply captured by the stabiliser:

$$\mathsf{Stab}(A) = \{\pi \in S_n \mid \pi.A \equiv A\}\,.$$

▶ *Example 5.7*

$\mathsf{Stab}(\{\!\{P, \ldots, P\}\!\}, \Phi) = S_n$ models the case where all parallel subprocesses are identical. On the contrary, the case where $\mathsf{Stab}(A) = \{id\}$ models that there is no redundancy at all between parallel processes. Intermediate examples model partial symmetries: the larger the stabiliser, the more redundancy. For example $\mathsf{Stab}(\{\!\{P, P, Q, Q, Q\}\!\}, \Phi)$ will contain at least the subgroup of S_n generated by the transpositions (1 2), (3 4) and (3 5). ◀

We also extend the action of the symmetric group to capture symmetries *up to channel renaming*. Indeed, permuting public channels does not affect the executability of processes in the private semantics; therefore two processes that are structurally-equivalent but differ on their public channels still have a similar set of traces.

Definition 5.20 (action of the channel-permutation group on processes)

The *channel-permutation group* of A is the group S_A^{ch} of permutations of ground terms deducible from $\Phi(A)$. It acts on extended processes as follows; if $\varrho \in S_A^{ch}$ we obtain $A.\varrho$ by replacing all ground channels u of A deducible from $\Phi(A)$ by $\varrho(u)$.

Typically to prove the equivalence of $P(c) \mid P(d)$ and $Q(c) \mid Q(d)$ where $P(x), Q(x)$ are processes operating on a single channel x, matching a trace starting with an action of either $P(c)$ or $P(d)$ results into a similar analysis, provided the channels c and d are known to the attacker. The symmetry will typically be expressed by the fact that the pair of permutations $((1\ \ 2),(c\ \ d))$ is simultaneously in $\mathsf{Stab}_{ch}(P(c) \mid P(d))$ and $\mathsf{Stab}_{ch}(Q(c) \mid Q(d))$, where:

$$\mathsf{Stab}_{ch}(A) = \{(\pi, \varrho) \in S_n \times S_A^{ch} \mid \pi.A.\varrho^{-1} \equiv A\}\,.$$

4.3 Universal symmetry optimisations

We first present a universal optimisation that reduces the number of possibilities when applying rules (In) and (Comm) at the start of a nonnegative phase. It captures the idea that, when considering the traces of several parallel protocol sessions, starting the trace by an action from one session or an other does not make a substantial difference, even when they use distinct public channels. To formalise this idea, let us consider a labelled trace $t \in \mathbb{O}_{\mathsf{c}}^{\forall}$:

$$t : [\,P\,]^{\varepsilon} \overset{\mathsf{tr}}{\Rightarrow} (\{\!|\,[\,P_i\,]^{\ell_i}\,|\!\}_{i=1}^{n}, \Phi_0) = A\,.$$

The goal is to exhibit conditions discarding some possible transitions directly following t. For that we define two subgroup Sym_1 and Sym_2 of the symmtric group to characterise symmetries. They capture an intuition that the symmetries in P shall be reflected in one way or an other in $\mathbb{T}(P,Q)$ to be exploited.

Symmetry by matching We first define a notion of symmetry within P that is required to be reflected in the matching with Q. We let the group $\mathsf{Sym}_1 = \mathsf{Sym}_1^P \cap \mathsf{Sym}_1^Q$ where

$$\mathsf{Sym}_1^P = \mathsf{Stab}(A) \qquad \mathsf{Sym}_1^Q = \bigcap_{\substack{t^2:(P,Q)\overset{\mathsf{tr}}{\Rightarrow}A^2 \\ t=\mathsf{fst}(t^2)\sim\mathsf{snd}(t^2)}} \{\pi \in S_n \mid s^2 : (P,Q) \overset{\mathsf{tr}}{\Rightarrow}_{\mathsf{ss}} \pi.A^2, \mathsf{fst}(s^2) = t\}$$

Intuitively $\pi \in \mathsf{Sym}_1^P$ means that for all a, P_a and $P_{\pi(a)}$ have the same traces, and $\pi \in \mathsf{Sym}_1^Q$ means that they can be matched by the same sessions of Q. In particular if $\pi \in \mathsf{Sym}_1$, executing first an action from P_a or $P_{\pi(a)}$ results into symmetric equivalence proofs.

Simultaneous symmetry We now define a notion of symmetry capturing redundancy occurring at the same time in P and Q, up to bijective renaming of public channels. We let $\mathsf{Sym}_2 = \{\pi \in S_n \mid \exists \varrho, (\pi, \varrho) \in \mathsf{Sym}_2^P \cap \mathsf{Sym}_2^Q\}$, where

$$\mathsf{Sym}_2^P = \mathsf{Stab}_{ch}(A) \qquad \mathsf{Sym}_2^Q = \bigcap_{\substack{t^2:(P,Q)\overset{\mathsf{tr}}{\Rightarrow}_{\mathsf{ss}}A^2 \\ t=\mathsf{fst}(t^2)\sim\mathsf{snd}(t^2)}} \mathsf{Stab}_{ch}(\mathsf{snd}(A^2))$$

Intuitively $\pi \in \mathsf{Sym}_2^P$ means that for all a, P_a and $P_{\pi(a)}$ have the same traces, and $\pi \in \mathsf{Sym}_2^Q$ means that all matching traces of Q have this symmetry as well. Hence, like Sym_1, $\pi \in \mathsf{Sym}_2$ captures a process symmetry reflected in the equivalence proof.

The optimisation: for input transitions In the following, Sym is the group generated by Sym_1 and Sym_2. We write $I \subseteq [\![1, n]\!]$ for the set of indexes i such that an input transition is applicable from P_i in $\mathbb{O}^{\forall}_{\mathsf{c+i^*+0}}$. If $i \in I$ we consider the orbit of i w.r.t. the action of Sym:

$$\mathsf{Orb}(i) = \{\pi(i) \mid \pi \in \mathsf{Sym}\} \cap I\,.$$

Intuitively starting a trace of A by an action of P_i or P_j results into a similar analysis if i and j are on the same orbit. Thus a correct optimisation is to consider only one index per orbit when listing the possible transitions from A. Yet this representant should be chosen carefully for compatibility with prior optimisations, namely $\mathbb{O}^{\forall}_{\mathsf{por}}$ that discards traces that are not minimal w.r.t. an ordering $\preccurlyeq$ on blocks. We should pick a representant i such that the execution of P_i induces a minimal block w.r.t. $\preccurlyeq$. For that we assume that the ordering $\preccurlyeq$ is entirely determined by the label of the first action of the block. Hence the following extension of $\preccurlyeq$ to I is well-defined:

$$i \preccurlyeq j \quad \Longleftrightarrow \quad \mathit{actions}(b_i) \preccurlyeq \mathit{actions}(b_j)$$

where b_i (resp. b_j) is an arbitrary block starting with an input from P_i (resp. P_j). An input transition on process P_a following the trace t is said *well-formed* if a is minimal w.r.t. $\preccurlyeq$ within $\mathsf{Orb}(a)$.

The optimisation: for internal communications The optimisation is the same as the one above, except that the symmetries should apply to the input and output processes at the same time. We write $IO \subseteq [\![1, n]\!]^2$ the set of pairs of indexes (i, j) such that an internal communication is possible between (P_i, P_j) in $\mathbb{O}^{\forall}_{\mathsf{c+i^*+0}}$, where the input is at the start of P_i. If $(i, j) \in IO$ we consider

$$\mathsf{Orb}(i, j) = \{(\pi(i), \pi(j)) \mid \pi \in \mathsf{Sym}\} \cap IO\,.$$

Intuitively starting a trace of A by an internal communication between processes P_i, P_j or P_k, P_l results into a similar analysis if (i, j) and (k, l) are on the same orbit. Then similarly to inputs we extend the ordering $\preccurlyeq$ on blocks to IO by writing

$$(i, j) \preccurlyeq (k, l) \quad \Longleftrightarrow \quad \mathit{actions}(b) \preccurlyeq \mathit{actions}(b')$$

where b (resp. b') is the block starting with the internal communication between P_i and P_j (resp. P_k and P_l). We thus qualify an internal communication between P_a and P_b following the trace t as *well-formed* if (a, b) is minimal w.r.t. $\preccurlyeq$ within $\mathsf{Orb}(a, b)$.

Correctness We say that a trace is well-formed when all its transitions (In) and (Comm) at the start of nonnegative phases are well-formed. We then define the optimisation $\mathbb{O}^{\forall}_{\mathsf{sym}}$ as the set of well-formed traces of $\mathbb{O}^{\forall}_{\mathsf{c+i^*+0}}$. Its correctness is proved in Appendix C.

Proposition 5.30

$\mathbb{O}^{\forall}_{\mathsf{por}} \cap \mathbb{O}^{\forall}_{\mathsf{sym}}$ is a correct refinement of $\mathbb{O}^{\forall}_{\mathsf{por}}$.

4.4 Existential symmetry optimisation

The goal of this optimisation is to exploit symmetries when applying the matching rule: when several processes are structurally equivalent then we do not need to consider redundant matchings. For instance, suppose that we need to match $P_1 \mid P_2$ with $Q \mid Q$. Just considering the identity permutation would be sufficient, and the permutation $(1\ \ 2)$ should be considered as redundant. Formally, let us consider an instance of the rule (MATCH)

$$(\{\!\{(P,Q)\}\!\} \cup \mathcal{P}^2, \Phi_0, \Phi_1) \xrightarrow{\tau}_{\mathsf{ss}^2} (\{\!\{(P_i, Q_{\pi(i)})\}\!\}_{i=1}^n \cup \mathcal{P}^2, \Phi_0, \Phi_1) = \pi^{-1}.A^2$$

with $P = P_1 \mid \cdots \mid P_n$, $Q = Q_1 \mid \cdots \mid Q_n$ and $\pi \in S_p$ where $p = n + |\mathcal{P}^2|$. We only consider S_p instead of the usual S_n for convenience as it will ease the formalisation of the optimisation, in particular the definition of the following relation on S_p:

$$\pi \sim \pi' \quad \iff \quad \exists u \in \mathsf{Stab}(\mathsf{snd}(A^2)),\ \pi' = \pi \circ u\,.$$

Proposition 5.31

$\sim$ is an equivalence relation on S_p.

Proof. We first observe that $A \equiv B$ implies $\pi.A \equiv \pi.B$, in particular $\mathsf{Stab}(\mathsf{snd}(A^2))$ is a group by Proposition 5.13. The conclusion is then immediate (reflexivity: a group of permutations contains the identity; symmetry: a group is closed by inverse; transitivity: a group is closed by composition). □

We say that an instance of rule (MATCH) is *well-formed* when the underlying permutation π is minimal within its equivalence class for $\sim$, w.r.t. an arbitrary total ordering on permutations. We denote by $\mathbb{O}^{\exists}_{\mathsf{sym}}$ the set of traces of extended twin processes whose instances of rule (MATCH) are all well-formed. The correctness of this optimisation is stated below and proved in Appendix C.

Proposition 5.32

$\mathbb{O}^{\exists}_{\mathsf{sym}}$ is a correct refinement of $\mathbb{O}^{\exists}_{\mathsf{all}}$.

5 Symbolic analysis and implementation

Section summary

We show that the construction of the partition tree (Chapter 4) can be adapted to equivalence by session and its optimisations. We then compare the performances of the modified algorithm with the procedure for trace equivalence, in terms of expressivity and verification time.

5.1 Symbolic matching

Subprocess matchings All optimisations aside, the essence of equivalence by session is to add structural requirements on top of the decision procedure for trace equivalence. As such we build on the procedure described in Chapter 3 to generate a partition tree. To make

the integration easier, we used an alternative characterisation of equivalence by session that is closer to trace equivalence. In essence it expresses the structural constraints imposed by twin processes as explicit bijections between labels that we call *session matchings*. A precise definition is given in Appendix C, with a proof that this is equivalent to the twin-process-based definition of equivalence.

In practice, our implementation consists of keeping track of these session matchings into the nodes of the partition tree generated by DeepSec. For obvious resource reasons, doing so by naively enumerating all subprocess permutation is prohibitive, hence the need for designing a symbolic treatment of these structural constraints (see example below). Our approach is to carry out a symbolic representation of all possible process matchings in each node of the partition tree. The set of all these bijections is then updated at each new symbolic transitions in the partition tree, among others to satisfy the requirement that matched subprocesses should have the same skeleton.

▶ *Example 5.8*
Consider two initial processes

$$P = c(x).P_0 \mid c(x).P_1 \mid \overline{c}\langle u\rangle.P_2 \qquad\qquad Q = c(x).Q_0 \mid \overline{c}\langle u'\rangle.Q_1 \mid c(x).Q_2\,.$$

In the root of the partition tree, P and Q will be labeled by 0, i.e., the root will contain the two symbolic processes

$$(\{[\,P\,]^0\}, \varnothing, \varnothing) \qquad\qquad (\{[\,Q\,]^0\}, \varnothing, \varnothing)\,.$$

There is only a single bijection between their labels, i.e., the identity $0 \mapsto 0$. Upon receiving this initial node, DeepSec applies the symbolic transition corresponding to our rule (Par), hence generating the two symbolic processes

$$\begin{array}{l}(\{\!\{[\,c(x).P_0\,]^{0.1}\,;\ [\,c(x).P_1\,]^{0.2}\ ;\ [\,\overline{c}\langle u\rangle.P_2\,]^{0.3}\}\!\}, \varnothing, \varnothing)\\ (\{\!\{[\,c(x).Q_0\,]^{0.1}\,;\ [\,\overline{c}\langle u'\rangle.Q_1\,]^{0.2}\,;\ [\,c(x).Q_2\,]^{0.3}\}\!\}, \varnothing, \varnothing)\end{array}$$

There are then only two possible bijection of labels that respect the skeleton requirement of twin processes:

$$\begin{array}{lcl} 0.1 \mapsto 0.1 & & 0.1 \mapsto 0.3 \\ 0.2 \mapsto 0.3 & \text{and} & 0.2 \mapsto 0.1 \\ 0.3 \mapsto 0.2 & & 0.3 \mapsto 0.2 \end{array}$$ ◀

These bijections are kept within the node of the partition tree and updated along side the other transformation rules of DeepSec. For obvious performance reasons, we cannot represent them by a naive enumeration of all process permutations. Fortunately, the skeleton requirement ensures an invariant that the set S of session matchings between two processes A and B is always of the form

$$S = \{\pi \mid \forall i, \forall \ell \in C_i, \pi(\ell) \in D_i\}$$

where the sets $C_1, \ldots, C_n$ form a partition of the labels of A and $D_1, \ldots, D_n$ a partition of the labels of B. In particular, S can succinctly be stored as a simple association list of equivalence classes.

Decision of equivalence Finally, as our trace refinements depend on two sets $\mathbb{O}^{\forall}$ and $\mathbb{O}^{\exists}$, we annotate each symbolic process in the node by $\forall$, $\exists$ or $\forall\exists$ tags. They mark whether the trace from the root of the partition tree to the tagged process is determined to be in $\mathbb{O}^{\forall}$, $\mathbb{O}^{\exists}$ or both respectively. For instance, the two initial symbolic processes in the root of the partition tree are labeled by $\forall\exists$. We also provide a decision procedure for inclusion by session $\sqsubseteq_s$ that consists of tagging one of the initial processes as $\forall$ and the other one as $\exists$.

The final decision criterion for equivalence by session is then strenghtened compared to the one for trace equivalence (recall Chapter 3, Theorem 3.3). For equivalence to hold, not only each node of the partition tree should contain at least one process originated from P and one process originated from Q, but each of them that has the tag $\forall$ should be paired with at least one other process of the node with the tag $\exists$.

5.2 Integration

From a high-level of abstraction, the integration of the universal optimisations described in Sections 3 and 4 prune some branches of the partition tree—those that abstract traces that do not belong to $\mathbb{O}^{\forall}_{\mathsf{por}} \cap \mathbb{O}^{\forall}_{\mathsf{sym}}$. For instance in Section 3.2, we showed that to prove equivalence by session, we can always perform non-input actions in priority. Therefore on a process $\overline{c}\langle u\rangle.P \mid c(x).Q$, we prevent DeepSec from generating a node corresponding to the execution of the input due to the presence of the output.

The integration of other optimisations is more technical in a symbolic setting, in particular the *lexicographic reduction* $\mathbb{O}^{\forall}_{\mathsf{c+r}}$ described in Section 3.6. Remember that it discards traces that do not satisfy the predicate Minimal, that identifies lexicographically-minimal traces among those obtained by permutation of independent blocks. Unfortunately, the definition of independence (Definition 5.14) is only defined for ground actions—and not their symbolic counterpart, that intuitively abstracts a set of ground actions. A branch may therefore be removed only if *all* its solutions violate the predicate Minimal. However, by Proposition 5.10, it is correct to only partially implement such optimisations.

In practice Based on the high-level description of the previous section, we extended the implementation of DeepSec to decide equivalence by session of P and Q. Upon completing an analysis, two cases can arise:

1. The two processes are proved equivalent by session. Then they are also trace equivalent by Proposition 5.2.
2. The two processes are not equivalent by session and DeepSec returns an attack trace t, say, in P, as a result.

In the second case, when using equivalence by session as a heuristic for trace equivalence, the conclusion is not straightforward. As discussed in Section 1.3, the witness trace t may not violate trace equivalence (false attack). We integrated a simple test to our prototype, that checks whether this is the case or not. For that we leverage the internal procedure of DeepSec by, intuitively, restricting the generation of the partition tree for checking $P \sqsubseteq_t Q$ to the unique branch corresponding to the trace t.

If this trace t appears to violate trace equivalence, which is the case for example in our analysis of two sessions of the BAC protocol, we naturally conclude that $P \not\approx_t Q$. Otherwise, the false attack may guide us to discover a real attack: our analysis of session equivalence

consider traces with a specific shape (see Sections 3 and 4). Thus, we implemented a simple heuristic that, whenever a false attack is discovered, also checks whether different permutations of actions of this false attack could lead to a true attack. For instance, this heuristic allowed us to disprove trace equivalence in some analyses of $n \geqslant 3$ sessions of BAC. When our heuristic cannot discover a true attack, the result is not conclusive: the processes may well be trace equivalent or not. We leave as future work the design of a complete decision procedure for trace equivalence that builds on a preliminary analysis of equivalence by session.

Experimental setting We report experiments (Figure 5.4) comparing the scope and efficiency of the following two approaches for proving trace equivalence:

1. The original version of DeepSec as a baseline;
2. The analysis leveraging our contributions (preliminary analysis of equivalence by session, test of false attack if it fails, and then the heuristic attempting to reconstruct a true attack).

Benchmarks were carried out on 20 Intel Xeon 3.10GHz cores, with 50 Gb of memory. The result table using the following graphical conventions for the result of the analysis:

- ✓ analysis terminates and equivalence holds
- ↯ analysis terminates and a *true* attack is found
- ⏱ analysis aborted due to timeout (12 hours)
- ✗ unable to conclude whether trace equivalence holds (false attack)

Let us comment the results of Figure 5.4. First, the analysis of Helios for two voters and an honest ballot box only benefits from an underwhelming gain in verification time when exploiting process symmetries. This is however not surprising: with an honest ballot box, when a voter casts their vote, the protocol's execution can only carry on once the ballot box has effecitvely received the ballot in question, which greatly already limits the combinatorial explosion due to concurrency. A slightly better improvement can be observed when increasing the number of voters, allowing to scale to 10 ballots instead of 9 for 3 voters. However the real improvement appears when considering a dishonest ballot box: due to the ability of the adversary to permute or drop ballot, an effective combinatorial explosion arise, giving the opportunity to exploit proof symmetries. When the baseline approach fails to conclude in 12h for 4 ballots, using equivalence by session only requires 2s.

The case of BAC is also interesting: analysing the model as defined in Chapter 2 results in a false attack. This is due to the initial key distribution that restricts the possible matchings, making it impossible to conclude a proof of equivalence by session. However, since the key distribution only fixes a bijection between readers and passports using unobservable communications, we could consider a model of unlinkability where this distribution is implicit (that is, each reader is defined by a process that is already paired with a passport). Using this equivalent model, equivalence by session can be proved up to 5 sessions and significantly outperforms the baseline approach. We included in the result table some experiments for each variant studied in Chapter 2 (honest sessions, arbitrary sessions, and fixed scenarios). Note in particular that in the cases of 3 or more fixed sessions, the attacks found when using equivalence by session were false attacks; true attacks could however be retrieved by using our simple heuristic.

Protocol + roles			baseline	eq. session
Helios ZKP with revote (BPRIV ballot privacy)	**2** voters + honest ballot box	**3** ballots	✓ <1s	✓ <1s
		8 ballots	✓ 44s	✓ 38s
		9 ballots	✓ 4m 33s	✓ 4m 21s
		10 ballots	✓ 10m 44s	✓ 28m 53s
		11 ballots	✓ 3h 21m	✓ 3h 2m
	3 voters + honest ballot box	**3** ballots	✓ <1s	✓ <1s
		7 ballots	✓ 1m 25s	✓ 50s
		8 ballots	✓ 11m 52s	✓ 6m 57s
		9 ballots	✓ 1h 40m	✓ 1h
		10 ballots	⏱	✓ 8h 19m
	2 voters + ∼dishonest ballot box	**2** ballots	✓ <1s	✓ <1s
		3 ballots	✓ 1m 3s	✓ 2s
		4 ballots	⏱	✓ 2s
		5 ballots		✓ 11s
		6 ballots		✓ 6m 21s
		7 ballots		✓ 4h 18s
BAC — vanilla	only honest sessions	**2** sessions	✓ 13m 12s	✗ <1s
		3 sessions	⏱	✗ <1s
BAC — implicit key distribution	only honest sessions	**2** sessions	✓ 9m 24s	✓ <1s
		3 sessions	⏱	✓ 1s
		4 sessions		✓ 18s
		5 sessions		✓ 1h 42m
	arbitrary sessions	**2** sessions	✓ 11h 40m	✓ <1s
		3 sessions	⏱	✓ 2s
		4 sessions		✓ 24m 9s
	2 fixed honest sessions	**2** identical	↯ <1s	↯ <1s
	3 fixed honest sessions	**2** identical + **1** fresh	⏱	↯ <1s
	4 fixed honest sessions	**3** identical + **1** fresh		↯ <1s
		2 identical + **2** fresh		✓ 19s
	5 fixed honest sessions	**4** identical + **1** fresh		↯ 2s
		3 identical + **2** fresh		↯ 10m 45s
		2 identical + **3** fresh		✓ 2h 52m

Figure 5.4 Improvement of performances when using equivalence by session (20 cores)

Part III

A guide to the complexity of equivalences

Introduction

The theoretical boundaries of the problem

In Part II wee have designed and evaluated the performances of an algorithm for proving trace equivalence and of some optimisations. In Part III we will focus more on the theoretical understanding of the problem.

As mentioned in the previous introductions, there exist many decidability results for reachability properties in symbolic models. We argue that the development of mature automated tools such as ProVerif or Tamarin has been eventually possible due to such more theoretical work, by permitting to understand the precise limits and possibilities of the problem in terms of decidability and complexity. Although the tool support for equivalence properties is steadily improving as discussed in Part II, our theoretical understanding of equivalence is still not at the level of reachability's. In Part III we give a more detailed snapshot of the state of the art.

1 Chapter 6 is a technical chapter were we prove the termination of the algorithm introduced in Part II for deciding equivalences, and study the theoretical complexity of the problem. More precisely we prove that the problems of deciding trace equivalence, labelled bisimilarity and equivalence by session are *coNEXP complete* for bounded processes and constructor-destructor subterm convergent theories. Interestingly, the worst-case complexity of equivalence by session is therefore the same as trace equivalence, which may be surprising due to the practical differences in performances between observed in Chapter 5.

2 In Chapter 7 we include this result in a more complete overview of the state of the art by surveying complexity results for the decidability of equivalences in the bounded fragment for various classes of theories. In particular we found during this survey several classes of processes and theories whose complexity had not been investigated before, or where existing complexity results could be improved; this survey thus includes several new or streghtened results. Moreover, we identified some variations in how the problems were stated across the literature. The most relevant variations are

   a the operational semantics used in the applied pi calculus, more precisely the treatment of internal communications (classical or private semantics, see Chapter 1);

   b whether the theory is considered as a constant of the problem.

   Although the complexity results we provide seem to be relatively stable when changing semantics, considering fixed theories or not has an impact on some complexity results. Note that Delaune and Hirschi [DH17] also survey symbolic methods for verifying equivalence properties; they mainly discuss tool support whereas we focus on computational complexity.

   ▶ A summary table of all complexity results can be found in Table 7.1, p.182.

Chapter 6:

Complexity analysis of DeepSec

Summary.

Technical Chapter. In this chapter we prove the *termination* of the constraint solving procedure used by DeepSec to generate partition trees (Chapter 4). This shows that, in addition of being correct, the overall verification algorithm carried out by the DeepSec tool (Chapter 3, Algorithm 1) is guaranteed to terminate. Besides we prove that, whenever equivalence does not hold, a *witness of non-equivalence of exponential size* can be extracted from the generated partition tree, thus justifying that trace equivalence, labelled bisimilarity and equivalence by session are decidable in coNEXP. We then show that this bound is tight, i.e., that the three problems are actually *coNEXP complete*.

Preliminary word

Organisation of the chapter In Chapter 3 we introduced the notion of partition tree and showed how trace equivalence and labelled bisimilarity could be decided assuming such a tree had been computed (procedure adapted to equivalence by session in Chapter 5). Then we provided a tree-generation procedure in Chapter 4 for arbitrary constructor-destructor subterm convergent theories (Algorithm 1). The structure of the current chapter is the following:

1. In Section 1 we prove that Algorithm 1 uses a *finite number of constraint-solving rules* to compute each branch of the partition tree. This proves the three equivalences to be decidable.
2. For complexity purposes we then refine this result in Section 2: we prove that Algorithm 1 applies at most an *exponential number of rules* and that the nodes of the resulting partition tree have *most general solutions of exponential (DAG) size*.
3. Relying on these bounds, we show that two processes are not equivalent *iff* there exists a *non-equivalence witness of exponential size* (as defined in Chapter 3). This shows the three equivalences to be decidable in coNEXP time.
4. Finally we show in Section 4 that the equivalences are *coNEXP hard* as well. All in all:

> ***Theorem 6.1*** *(complexity of equivalences)*
>
> The decision problems TraceEquiv, Bisimilarity and SessEquiv are coNEXP complete for bounded processes and constructor-destructor subterm convergent theories.

Notations We also introduce some notations that will be used in most sections of this chapter. We recall that we study complexity w.r.t. the DAG size of terms (which provides stronger results compared to complexity bounds w.r.t. the tree size of terms); in particular the DAG size of a substitution σ is $|\sigma|_{\mathsf{dag}} = |subterms(img(\sigma))|$, hence the many occurrences of subterm sets below. Given an extended constraint system $\mathcal{C}^e = (\Phi, \mathsf{D}, \mathsf{E}^1, \mathsf{E}^2, \mathsf{K}, \mathsf{F})$ we write

$$\begin{aligned}
\mu^1 &= mgu(\mathsf{E}^1) && \text{(first-order mgu)}\\
\mu^2 &= mgu(\mathsf{E}^2) && \text{(second-order mgu)}\\
T^1 &= subterms^1(img(\Phi\mu^1), img(\mu^1), \mathsf{K}\mu^1, \mathsf{D}\mu^1) && \text{(first-order terms)}\\
T^2 &= subterms^2(img(\mu^2), \mathsf{K}, \mathsf{D}) && \text{(second-order terms)}\\
used^2 &= subterms_{\mathsf{c}}(img(\mu^2),\ \mathsf{K}\cup\mathsf{D}) \cup vars^2(\mathsf{D}) && \text{(solution recipes)}
\end{aligned}$$

When the extended constraint system is not clear from context we write explicitly $\mu^1(\mathcal{C}^e)$, $\mu^2(\mathcal{C}^e)$, $T^1(\mathcal{C}^e)$,... Intuitively μ^i are the mgu's of the equations of E^i and we recall in particular that $mgs(\mathcal{C}^e) = \{\mu^2\}$ when $\mathcal{C}^e$ is solved (Chapter 4, Section 2.4, Proposition 4.1). The other notations assume $\mu^1 \neq \bot$ (if $\mu^1 = \bot$, $\mathcal{C}^e$ will be discarded by the normalisation rule (MGS-Unsat) anyway). The sets T^1 and T^2 respectively represent the first-order and second-order terms appearing in the system, while $used^2 \subseteq T^2$ models the set of recipes used to build the solution of $\mathcal{C}^e$ (i.e., μ^2) from $\mathsf{K}\cup\mathsf{D}$. We recall that it is the same set as the one used when defining the constraint-solving rules for most general solutions (Chapter 4, Section 2.2).

▶ *Remark: uniformity of second-order terms across components*
Due to an invariant of the procedure (Inv_{str} formalised in Appendix B, Section 1), we know that all extended constraint systems in a component Γ have the same second-order structure. Here this means that $\mu^2(\mathcal{C}^e_1) = \mu^2(\mathcal{C}^e_2)$ and $T^2(\mathcal{C}^e_1) = T^2(\mathcal{C}^e_2)$ for any $(\mathcal{P}_1, \mathcal{C}_1, \mathcal{C}^e_1), (\mathcal{P}_2, \mathcal{C}_2, \mathcal{C}^e_2) \in \Gamma$. For this reason we may write $\mu^2(\Gamma)$ or $T^2(\Gamma)$ instead of $\mu^2(\mathcal{C}^e)$ or $T^2(\mathcal{C}^e)$ for some arbitrary $(\mathcal{P}, \mathcal{C}, \mathcal{C}^e) \in \Gamma$. ◀

1 Termination of the constraint solving

Section summary
As we showed in Chapters 3 and 4, our algorithm for producing equivalence proofs constructs a partition tree using a constraint solving procedure. We prove here that it terminates and give some preliminary complexity arguments.

1.1 Termination of the computation of most general solutions

We first study the termination of the procedure for computing most general solutions of Chapter 4, Section 2. The proof mostly relies on the following measure that characterises the set of first-order terms of an extended constraint system $\mathcal{C}^e$ that are not used in $mgs(\mathcal{C}^e)$, that is, that are not deduced by any recipe $\xi \in used^2(\mathcal{C}^e)$:

$$unused^1(\mathcal{C}^e) = \{t \in T^1 \mid t \notin \mathcal{X}^1 \wedge \forall \xi \in used^2(\mathcal{C}^e) \setminus \mathcal{X}^2,\ (\xi, t) \notin \mathsf{Conseq}(\mathsf{K}\mu^1 \cup \mathsf{D}\mu^1)\}$$

The simplification rules for mgs' do not affect this value (except maybe if $\mathcal{C}^e$ is replaced by $\bot$ by (MGS-Unsat)). The application of (MGS-Conseq) will ensure that $unused^1$ is at

least non-increasing, while (MGS-Res) and (MGS-Cons) make it strictly decreasing. We summarise this as the following proposition, proved in Appendix D; we recall that, similarly to the correctness arguments in Chapter 4, the statements makes reference to some procedure invariants formalised in Appendix D:

Proposition 6.2 (decrease of unused first-order terms during constraint solving)
Let $\mathcal{C}^e$ be an extended constraint system such that $\mathcal{C}^e\{ = \mathcal{C}^e$ and the invariants $\mathsf{Inv}_{wf}(\mathcal{C}^e)$ and $\mathsf{Inv}_{sound}(\mathcal{C}^e)$ hold. Then let $\mathcal{C}^e \xrightarrow{\Sigma}\{ \mathcal{C}^{e\prime} \neq \bot$. If this transition is derived with:
1 Rule (MGS-Conseq): $|unused^1(\mathcal{C}^{e\prime})| \leqslant |unused^1(\mathcal{C}^e)|$
2 Rules (MGS-Res) or (MGS-Cons): $|unused^1(\mathcal{C}^{e\prime})| < |unused^1(\mathcal{C}^e)|$

Using this proposition we can then easily prove the computation of the set of most general solutions to be terminating, and actually to give an upper bound on its cardinality:

Theorem 6.3 *(termination for most general solutions)*
There exist no infinite sequences of transitions w.r.t. $\rightarrow\{$. Besides if $\mathcal{C}^e$ is an extended constraint system such that the invariants $\mathsf{Inv}_{wf}(\mathcal{C}^e)$ and $\mathsf{Inv}_{sound}(\mathcal{C}^e)$ hold, we have

$$|mgs(\mathcal{C}^e)| \leqslant (|\mathsf{K}(\mathcal{C}^e)| + 1)^{|unused^1(\mathcal{C}^e)|}$$

Proof. First of all we observe that consecutive applications of Rule (MGS-Conseq) are terminating, since applying this rule strictly decrease the cardinality of the set $m(\mathcal{C}^e)$ of parameters (ξ, ζ) the rule can be applied with. Combining this with Proposition 6.2 we obtain that if $\mathcal{C}^e \xrightarrow{\Sigma}\{ \mathcal{C}^{e\prime} \neq \bot$ then $\mathcal{C}^e < \mathcal{C}^{e\prime}$ w.r.t. the lexicographic composition of $unused^1$ and m.

Besides consecutive applications of Rule (MGS-Conseq) are also confluent by unicity of mgu's. For the same reason the applications of Rules (MGS-Res) or (MGS-Cons) can be performed on one deterministically-chosen deduction fact $(X \vdash^? u) \in \mathsf{D}$. We therefore obtain $mgs(\mathcal{C}^e) = \{\Sigma_{|vars^2(\mathcal{C}^e)} \mid \mathcal{C}^e \xRightarrow{\Sigma}\{ \mathcal{C}^{e\prime}$ normalised, $\mathcal{C}^{e\prime}$ solved$\}$ where a reduction $\mathcal{C}^e \xRightarrow{\Sigma}\{ \mathcal{C}^{e\prime}$ is said to be *normalised* when all applications of Rule (MGS-Conseq) and the choice of deduction facts in Rules (MGS-Res) or (MGS-Cons) are done in a fixed, deterministic way. Since for any $\mathcal{C}^e$, there are at most $|\mathsf{K}(\mathcal{C}^e)|$ normalised applications of Rule (MGS-Res) and 1 normalised application of Rule (MGS-Cons), we deduce by Proposition 6.2 that $|mgs(\mathcal{C}^e)| \leqslant (|\mathsf{K}(\mathcal{C}^e)| + 1)^{|unused^1(\mathcal{C}^e)|}$. □

1.2 Termination of the computation of partition trees

To bound the number of rule applications in Algorithm 1 we define a well-founded measure that decreases after each case-distinction, simplification, normalisation and vector-simplification rules. More precisely the rule applications are always of the form

$$\mathbb{S} \cup \{\Gamma\} \rightarrow \mathbb{S} \cup \{\Gamma_1, \ldots, \Gamma_p\} \qquad\qquad p \in \{1, 2\}$$

and we show that for all $i \in [\![1, p]\!]$, $\Gamma > \Gamma_p$ w.r.t. to a well-founded measure on components (under the invariants of the procedure defined in Appendix B). This therefore bounds the number of rule applications to compute a given branch of the partition tree. The measure in question is a tuple of 10 integer components that is ordered w.r.t. the lexicographic ordering.

Measure 1: sizes of the processes As first element of the measure, we compute a maximum on the sizes of the processes in the multisets $\mathcal{P}$, that is,

$$M_1(\Gamma) = \max_{(\mathcal{P},\mathcal{C},\mathcal{C}^e)\in\Gamma} \sum_{R\in\mathcal{P}} |R|_{\mathsf{dag}} \qquad \text{(Meas. 1)}$$

Notice that this stays unchanged for any simplification or case distinction rules but strictly decreases when applying the extended symbolic transitions.

Measure 2: Number of non-deducibility facts The second element of the measure is the number of non-deducibility facts $\forall X.X \not\vdash^? u$ appearing in D. This types of formulas can only be added by symbolic rules (that already decrease the first component of the measure) and strictly decrease when Rule (Chan) is applied.

$$M_2(\Gamma) = \sum_{S\in\Gamma} |\{(\forall X.\, X \not\vdash^? u) \in \mathsf{D}(S)\}| \qquad \text{(Meas. 2)}$$

Measure 3: Number of constraint systems The third element of the measure considers the number of extended symbolic processes in the set, i.e., $|\Gamma|$, that may increase only when applying a symbolic transition; however it strictly decreases when applying the simplification rules (Vect-Split) and (Vect-rm-Unsat). Moreover it also strictly decreases for the positive branch of Rule (Sat) when applied with the case 3 of its application conditions. In such a case, we consider a disequation ψ and a mgs Σ of $\mathcal{C}^e_j$ that does not satisfy ψ, which will lead to at least one $S \in \Gamma$ being discarded by the simplification rule (Vect-rm-Unsat).

$$M_3(\Gamma) = |\Gamma| \qquad \text{(Meas. 3)}$$

Measure 4: Number of terms not consequence Given $\mathcal{C}^e$ an extended symbolic constraint system, let us consider the following set representing the set of terms that are not consequence of $\mathsf{K}(\mathcal{C}^e)$ and $\mathsf{D}(\mathcal{C}^e)$:

$$\mathsf{set}_{\mathsf{K}}(\mathcal{C}^e) = \{t \in T^1 \mid \forall \xi, (\xi, t) \notin \mathsf{Conseq}(\mathsf{K}(\mathcal{C}^e) \cup \mathsf{D}(\mathcal{C}^e))\}$$

Typically it corresponds to the terms that are not deducible by the attacker but could potentially be (because the knowledge base is not saturated yet). In fact, when the simplification Rule (Vect-add-Conseq) is applied, i.e., when a deduction fact $\xi \vdash^? u$ is added to $\mathsf{K}(\mathcal{C}^e)$, the term u is necessarily a subterm of the frame by the invariant $\mathsf{Inv}_{wf}(\mathcal{C}^e)$. Moreover by definition of Rule (Vect-add-Conseq) we know that u is not already consequence, meaning that the size of $\mathsf{set}_{\mathsf{K}}(\mathcal{C}^e)$ will strictly decrease. Finally the case distinction rules never increase the number of elements of $\mathsf{set}_{\mathsf{K}}(\mathcal{C}^e)$: indeed they all consist of applying substitutions Σ that are most general solutions of some systems having $\mathsf{K}(\mathcal{C}^e)$ as their knowledge base, hence their first-order terms are consequence by K-basis. All in all we choose the following component:

$$M_4(\Gamma) = \min_{(\mathcal{P},\mathcal{C},\mathcal{C}^e)\in\Gamma} |\mathsf{set}_{\mathsf{K}}(\mathcal{C}^e)| \qquad \text{(Meas. 4)}$$

Measure 5: Number of unsolved extended constraint systems We recall that the aim of Rule (Sat), case 1 of its application conditions, is to put extended constraint systems in solved form (that is, in a form where they trivially have μ^2 as a unique mgs). If

$$M_5(\Gamma) = |\{(\mathcal{P}, \mathcal{C}, \mathcal{C}^e) \in \Gamma \mid \mathcal{C}^e \text{ unsolved}\}| \qquad \text{(Meas. 5)}$$

then this measure is strictly decreasing when applying the rule in question. Once a system has a unique mgs, intantiating its second variables does not change this fact and the other case distinction rules are therefore non-increasing w.r.t. this measure.

Measure 6: Applicability of Rule REW The next element represents the number of applications of Rule (Rew) that are still possible. Typically, we consider all the parameters of the rule (Rew) (the deduction facts from K, the rewrite rule, etc...) on which the rule would be applied with a most general solutions that does not already corresponds to a deduction fact in F. If $\mathcal{C}^e$ is an extended constraint system we therefore consider $\mathsf{set}_{\text{Rew}}(\mathcal{C}^e)$ the set of tuples $(\psi, \ell \to r, p, \psi_0, \Sigma)$ that satisfy all the application conditions of Rule (Rew), and

$$M_6(\Gamma) = \sum_{(\mathcal{P},\mathcal{C},\mathcal{C}^e)\in\Gamma} |\mathsf{set}_{\text{Rew}}(\mathcal{C}^e)| \qquad \text{(Meas. 6)}$$

By definition $|\mathsf{set}_{\text{Rew}}(\mathcal{C}^e)|$ strictly decreases for at least one $(\mathcal{P}, \mathcal{C}, \mathcal{C}^e) \in \Gamma$ (and non-increasing for the others) when applying Rule (Rew). Then let $(\mathcal{P}, \mathcal{C}, \mathcal{C}^e) \in \Gamma$: the other case distinction rules (Sat) and (Eq) do not increase $|\mathsf{set}_{\text{Rew}}(\mathcal{C}^e)|$. Indeed if we consider one of their applications $\mathcal{C}^e \xrightarrow{\Sigma'}\!\!\!\!\!\!\{\ \mathcal{C}^e{:}\Sigma'$, we have

$$\mathsf{set}_{\text{Rew}}(\mathcal{C}^e{:}\Sigma') = \{(\psi\Sigma', \ell \to r, p, \psi_0, \Sigma\Sigma') \mid (\psi, \ell \to r, p, \psi_0, \Sigma) \in \mathsf{set}_{\text{Rew}}(\mathcal{C}^e)\}\,.$$

Note however that $|\mathsf{set}_{\text{Rew}}(\mathcal{C}^e)|$ may increase by application of Rule (Vect-add-Conseq) since $|\mathsf{K}(\mathcal{C}^e)|$ will increase; yet this rule is already decreasing by the component M_4.

Measure 7: Number of unsolved deduction formulas We recall that Rule (Sat), case 2 of its application conditions, applies a most general solution to remove the hypotheses of one formula $\psi \in \mathsf{F}(\mathcal{C}^e)$ for some $(\mathcal{P}, \mathcal{C}, \mathcal{C}^e) \in \Gamma$ (the formula becomes solved in the positive branch, and is removed by (Norm-Formula) in the negative branch). This rule application is therefore strictly decreasing w.r.t. the measure

$$M_7(\Gamma) = |\{\psi \in \mathsf{F}(\mathcal{C}^e) \mid (\mathcal{P}, \mathcal{C}, \mathcal{C}^e) \in \Gamma, \psi \text{ unsolved deduction formula}\}| \qquad \text{(Meas. 7)}$$

We only consider deduction formulas ψ for this component. In particular the only rule that may increase this measure (i.e., generate unsolved deduction formulas) is (Rew) which is already decreasing w.r.t. the previous component of the measure.

Measure 8: Applicability of Rule EQ Similarly to the analogue component for Rule (Rew), we now define the next component that bounds the maximal number of possible applications of Rule (Eq). The application conditions stipulate that it can be applied either

1. on two deduction facts of $\mathsf{K}(\mathcal{C}_i^e)$, or
2. on one deduction fact of $\mathsf{K}(\mathcal{C}_i^e)$ in combination with a construction function symbol.

Even if the application conditions also consider a mgs Σ, the number of applications of Rule (Eq) will not depend on their number; this is intuitively because after applying the rule with one arbitrary mgs Σ, the conditions forbid any later applications with identical parameters except Σ. Formally consider for example the case 1 (case (2) follows the same reasoning). The rule is applied on two deduction facts $(\xi_1 \vdash^? u_1), (\xi_2 \vdash^? u_2) \in \mathsf{K}(\mathcal{C}^e)$. Thus, an equality formula with $\xi_1\Sigma =^?_f \xi_2\Sigma$ as head will be added in $\mathsf{F}(\mathcal{C}^e{:}\Sigma)$. However, in further applications of the rule, the condition that "for all $(\varphi \Rightarrow H) \in \mathsf{F}(\mathcal{C}^e{:}\Sigma)$, $H \neq (\xi_1 =^?_f \xi_2)$" will prevent a new application with the same (up to instantiation of Σ) deductions facts from $\mathsf{K}(\mathcal{C}^e{:}\Sigma)$.

We therefore conclude that the rule (Eq) can be applied only once per pair of deduction facts in K and once per deduction fact in K and function symbol in $\mathcal{F}_\mathsf{c}$. If $\mathcal{C}^e$ is an extended constraint system we therefore consider $\mathsf{set}_{\mathrm{Eq}}(\mathcal{C}^e)$ the set of pairs $(\psi, \psi') \in \mathsf{K}(\mathcal{C}^e)^2$ or $(\psi, \mathsf{f}) \in \mathsf{K} \times \mathcal{F}_\mathsf{c}$ that satisfy all the application conditions of the rule (Eq), and

$$M_8(\Gamma) = \sum_{(\mathcal{P},\mathcal{C},\mathcal{C}^e)\in\Gamma} |\mathsf{set}_{\mathrm{Eq}}(\mathcal{C}^e)| \qquad \text{(Meas. 8)}$$

Measure 9: Number of unsolved equality formulas We now introduce the analogue of Component 7 for equality formulas, that is,

$$M_9(\Gamma) = |\{\psi \in \mathsf{F}(\mathcal{C}^e) \mid (\mathcal{P},\mathcal{C},\mathcal{C}^e) \in \Gamma, \psi \text{ unsolved equality formula}\}| \qquad \text{(Meas. 9)}$$

As before Rule (Sat) makes this measure decrease in the case 2 of its application conditions. On the contrary unsolved equality formulas can be generated by two rules: the case distinction rule (Eq) or the vector-simplification rule (Vect-add-Formula).

Measure 10: Remaining most general solutions So far, every time we showed that one of the previous element of the measure (strictly) decrease by application of a case distinction rule, we always focused on the positive branches of case-distinction rules. The negative branches on the contrary only add recipe disequations to the system, which does not increase any the previous components of the measure *but* strictly decreases the number of most general solutions we can compute for the same instance of the rule. For example, if $\Sigma \in mgs(\mathcal{C}^e)$ then $|mgs(\mathcal{C}^e)| > |mgs(\mathcal{C}^e[\mathsf{E}^2 \wedge \neg\Sigma])|$. Hence it sufficies to consider the last component:

$$M_{10}(\Gamma) = \left|\left\{\Sigma \mid \begin{array}{l}\text{there exists a case distinction rule applicable}\\ \text{from } \mathcal{C}^e \text{ with parameter } \Sigma,\ (\mathcal{P},\mathcal{C},\mathcal{C}^e) \in \Gamma\end{array}\right\}\right| \qquad \text{(Meas. 10)}$$

Conclusion This gives the termination of the algorithm for computing $T \in \mathsf{PTree}(P, Q)$. We study more precisely the Components 1 to 9 of the measure in Appendix D and prove that they can all be bound by an exponential in $|P, Q, E|_{\mathsf{dag}}$ (with E the theory). Hence:

> ***Theorem 6.4*** *(termination for partition trees)*
>
> For all P, Q plain processes, Algorithm 1 terminates with arguments P, Q. Moreover each branch of the resulting tree is generated by applying at most an exponential number (in $|P, Q, E|_{\mathsf{dag}}$) of rules, not counting the negative branches of case distinction rules.

2 Bounding the size of most general solutions

Section summary

Using the results of the previous section, we now prove that the partition trees generated by Algorithm 1 have the remarkable property that all nodes have a *most general solution of exponential size*. This will be the core argument to prove equivalences to be decidable in coNEXP.

2.1 Overall approach

Objective We now focus on the theoretical complexity of the decision problems TraceEquiv, Bisimilarity and SessEquiv. Our goal for now is to prove that these three equivalences are decidable in coNEXP and the core argument to achieve this is to prove the theorem:

Theorem 6.5 *(size of most general solutions)*

If T is a partition tree of P, Q (w.r.t. a theory E) generated by Algorithm 1, then for all nodes n of T, $|mgs(n)|_{\mathsf{dag}}$ is exponential in $|P, Q, E|_{\mathsf{dag}}$.

We will detail in Section 3 how to derive a coNEXP decision procedure for equivalence properties by using this result. To bound the size of most general solutions we rely on the results previously established in Section 1.1: in the final partition tree $mgs(n) = \mu^2(\Gamma(n))$ and it therefore sufficies to prove that for all nodes, $\left|\mu^2(\Gamma(n))\right|_{\mathsf{dag}}$ is exponential in $|P, Q, E|_{\mathsf{dag}}$. However we will instead study the easier-to-track bound:

$$|T^2(\Gamma(n))| \geqslant |subterms(img(\mu^2(\Gamma(n))))| = \left|\mu^2(\Gamma(n))\right|_{\mathsf{dag}}$$

Evolution of second-order terms Let us now consider each constraint-solving rule and determine how T^2 evolves along the components along a branch of the partition tree.

1. *Symbolic rules*: only Rules (E-In) and (E-Out) increase the size of T^2 by adding at most two new second order variables.
2. *Simplification, normalisation, vector-simplification rules*: only Rule (Vect-add-Conseq) may increase the size of $T^2(\Gamma)$. Indeed, it transfers a deduction fact from F in K for each extended constraint systems in the current component Γ.
3. *Case distinction rules*: the positive branches of these rules increase the size of T^2 whereas the negative branches leave it unchanged.

It therefore sufficies to prove the following result to obtain Theorem 6.5:

Proposition 6.6 (evolution of second-order terms in partition trees)

If $\mathbb{S} \cup \{\Gamma\} \to \mathbb{S} \cup \mathbb{S}'$ where $\mathbb{S}' = \{\Gamma'\}$ is obtained by Rule (Vect-add-Conseq) or $\Gamma' \in \mathbb{S}'$ is the positive branch of a case distinction rule, then $\left|T^2(\Gamma')\right|_{\mathsf{dag}} - \left|T^2(\Gamma)\right|_{\mathsf{dag}}$ is bounded by a polynomial in $|P, Q, E|_{\mathsf{dag}}$.

Indeed we recall that by Theorem 6.4, we already know that each branch of the partition tree is obtained after applying at most an exponential number of rules (negative branches of case distinction rules excluded). Hence we obtain the expected exponential bound on T^2 when combined with the above proposition. The remaining of Section 2 is dedicated to its proof.

2.2 Bounding the increase of the second-order terms

When applying a mgs We first study the growth of $T^2(\mathcal{C}^e)$ when applying a mgs to $\mathcal{C}^e$, which means proving Theorem 6.5 in the case of Rule (Sat). Similarly to our previous results on most general solutions (Section 1.1), our bounds depend on $\mathit{unused}^1(\mathcal{C}^e)$ the number of first-order terms of $\mathcal{C}^e$ that are not already used in the solution, i.e., in μ^2. We also recall that by Proposition 6.2, this measure is non-increasing when applying any of the mgs simplification and constraint-solving rules, and is even strictly decreasing in the case of Rule (MGS-Res) and (MGS-Cons). Let us now show that its growth is actually inverted compared to T^2, that is, how much T^2 increases can be bounded by how much unused^1 decreases:

Proposition 6.7 (evolution of second-order terms when applying mgs)
For all extended processes $\mathcal{C}^e$ that verify the invariants $\mathsf{Inv}_{wf}(\mathcal{C}^e)$ and $\mathsf{Inv}_{sound}(\mathcal{C}^e)$, we have

$$\forall \Sigma \in \mathit{mgs}(\mathcal{C}^e),\ |T^2(\mathcal{C}^e{:}\Sigma)| \leq |T^2(\mathcal{C}^e)| + |\mathcal{F}| \times (|\mathit{unused}^1(\mathcal{C}^e)| - |\mathit{unused}^1(\mathcal{C}^e{:}\Sigma)|)$$

Proof. We assume $|\mathcal{F}| > 0$ by convention. It sufficies to prove this property when replacing $\mathcal{C}^e{:}\Sigma$ by $\mathcal{C}^{e\prime}$ for $\mathcal{C}^e \to \mathcal{C}^{e\prime} \neq \bot$ obtained by a mgs simplification or constraint-solving rules. We perform a case analysis on the rule in question.

▷ *case 1: simplification rule on formulas*

The simplification rules on formulas only affect first-order terms and second-order disequations and we therefore have $T^2(\mathcal{C}^{e\prime}) = T^2(\mathcal{C}^e)$.

▷ *case 2: mgs simplification rule*

We only need to consider Rule (MGS-Unif). Since it only affects first-order terms, the reasoning is identical to the previous case.

▷ *case 3: mgs constraint-solving rule*

Rules (MGS-Conseq) and (MGS-Res) apply a second-order substitution $\Sigma = \mathit{mgu}(\xi =^? \zeta)$ to $\mathcal{C}^e$ for some $\xi, \zeta \in T^2(\mathcal{C}^e)$. In particular we deduce that $|T^2(\mathcal{C}^{e\prime})| \leqslant |T^2(\mathcal{C}^e)|$ and the conclusion thus follows from the fact that $\mathit{unused}^1(\mathcal{C}^e) - \mathit{unused}^1(\mathcal{C}^{e\prime}) \geqslant 0$ by Proposition 6.2. Finally the only rule that increases $T^2(\mathcal{C}^e)$ is the last one, (MGS-Cons), that generates n fresh second-order variables for some constructor symbol f/n. In particular $|T^2(\mathcal{C}^{e\prime})| \leqslant |T^2(\mathcal{C}^e)| + n$, hence the result since $\mathit{unused}^1(\mathcal{C}^e) - \mathit{unused}^1(\mathcal{C}^{e\prime}) > 0$ by Proposition 6.2. □

In particular we obtain Proposition 6.6 for Rule (Sat) case 1, provided we prove that $\mathit{unused}^1(\mathcal{C}^e)$ is bounded by a polynomial. We explain in Appendix D how to extend the argument to Rules (Eq), (Rew) and (Vect-add-Conseq).

Bound of unused terms To conclude let us therefore establish a polynomial bound on $|\mathit{unused}^1(\mathcal{C}^e)|$. In order to do so we explore the relation between $\mathcal{C}$ and $\mathcal{C}^e$ in $(\mathcal{P}, \mathcal{C}, \mathcal{C}^e) \in \Gamma$. Intuitively $\mathit{unused}^1(\mathcal{C}^e)$ always has less elements then $\mathit{unused}^1(\mathcal{C})$ because

1. the symbolic rules always add the same constraints to $\mathcal{C}$ and $\mathcal{C}^e$, ensuring that $\mathit{unused}^1(\mathcal{C}^e)$ increases at most as much as $\mathit{unused}^1(\mathcal{C})$ by these rules
2. the other rules leave $\mathcal{C}$ untouched and do not make $\mathit{unused}^1(\mathcal{C}^e)$ increase.

Proposition 6.8 (approximation of unused terms)
For all $(\mathcal{P},\mathcal{C},\mathcal{C}^e) \in \Gamma$, $|unused^1(\mathcal{C}^e)| \leqslant \left|\Phi(\mathcal{C})\mu^1(\mathcal{C}), \mu^1(\mathcal{C})\right|_{\mathsf{dag}}$.

Proof. Considering $\mathcal{C}$ instead of $\mathcal{C}^e$, we have the trivial approximation

$$|unused^1(\mathcal{C})| \leqslant |T^1(\mathcal{C})| = subterms(\Phi(\mathcal{C})\mu^1(\mathcal{C}), \mu^1(\mathcal{C})) = \left|\Phi(\mathcal{C})\mu^1(\mathcal{C}), \mu^1(\mathcal{C})\right|_{\mathsf{dag}} .$$

It therefore sufficies to prove that $|unused^1(\mathcal{C}^e)| \leqslant |unused^1(\mathcal{C})|$. For that we show that the inequality $|unused^1(\mathcal{C}^e)| \leqslant |unused^1(\mathcal{C})|$ is preserved when applying any of the constraint-solving rules.

▷ *case 1: symbolic rules*

These rules add the same constraints to $\mathcal{C}^e$ and $\mathcal{C}$ (up to an additional deduction fact added to $\mathsf{F}(\mathcal{C}^e)$ in the case of Rule (E-Out), but this does not affect $unused^1(\mathcal{C}^e)$). In particular since $\mathsf{E}^2(\mathcal{C}) = \mathsf{K}(\mathcal{C}) = \varnothing$, if we consider an instance $(\mathcal{P},\mathcal{C},\mathcal{C}^e) \xrightarrow{\alpha}_{\mathsf{s}} (\mathcal{P}',\mathcal{C}',\mathcal{C}^{e\prime})$ of a symbolic rule we therefore have

$$|unused^1(\mathcal{C}^{e\prime})| - |unused^1(\mathcal{C}^e)| \leqslant |unused^1(\mathcal{C}')| - |unused^1(\mathcal{C})|$$

which gives the expected result.

▷ *case 2: simplification, normalisation, vector-simplification rules*

By definition these rules only affect $\mathcal{C}^e$ and leave $\mathcal{C}$ untouched, hence the conclusion since these rules do not increase $unused^1(\mathcal{C}^e)$.

▷ *case 3: case distinction rules*

Let us consider $\mathsf{CompatSubs}(\mathcal{C}^e)$ the set of substitutions Σ such that the notation $\mathcal{C}^e{:}\Sigma$ is well defined, that is, such that

1. if $dom(\Sigma) \subseteq vars^2(\mathsf{D}(\mathcal{C}^e))$
2. for all $X \in dom(\Sigma)$, there exists t such that $(X\Sigma, t) \in \mathsf{Conseq}(\mathsf{K}(\mathcal{C}^e) \cup \mathsf{D}')$ where $\mathsf{D}' = \{Y \vdash^? u \in \mathsf{D}(\mathcal{C}^e) \mid Y \notin dom(\Sigma)\} \cup D_{fresh}$ with

$$D_{fresh} = \{Y \vdash^? y \mid Y \in vars^2(img(\Sigma_{|vars^2(\mathcal{C}^e)})) \setminus vars^2(\mathcal{C}^e), y \text{ fresh}\}$$

This is intuitively the set of substitutions Σ whose image is constructed from $\mathsf{K}(\mathcal{C}^e)$, up to the new variables of D_{fresh} introduced by Σ. In particular we have for all $\Sigma \in \mathsf{CompatSubs}(\mathcal{C}^e)$, $|unused^1(\mathcal{C}^e{:}\Sigma)| \leqslant |unused^1(\mathcal{C}^e)|$ (which follows in more details from Proposition B.5 in Appendix B), hence the conclusion. □

This relation allows to eventually reduce the problem to give a polynomial bound on $\Phi(\mathcal{C})$ and $\mu^1(\mathcal{C})$ which are only affected by symbolic rules (we recall that the other constraint-solving rules do not modify $\mathcal{C}$). All in all this concludes the proof of the expected polynomial bound:

Corollary 6.9 (polynomial evolution of second-order terms)
For all extended processes $\mathcal{C}^e$ that verify the invariants $\mathsf{Inv}_{wf}(\mathcal{C}^e)$ and $\mathsf{Inv}_{sound}(\mathcal{C}^e)$, we have

$$\forall \Sigma \in mgs(\mathcal{C}^e),\ |T^2(\mathcal{C}^e{:}\Sigma)| \leq |T^2(\mathcal{C}^e)| + 9\,|P,Q,E|^3_{\mathsf{dag}}$$

Proof. We agree on the convention that $|P|_{\mathsf{dag}}$, $|Q|_{\mathsf{dag}}$ and $|E|_{\mathsf{dag}}$ are strictly positive. By Propositions 6.7 and 6.8, it sufficies to prove that for all symbolic traces $P \overset{\mathsf{tr}}{\Rightarrow}_{\mathsf{s}} (\mathcal{P}, \mathcal{C})$, $\left|\Phi(\mathcal{C})\mu^1(\mathcal{C}), \mu^1(\mathcal{C})\right|_{\mathsf{dag}} \leqslant 9\,|P, E|^2_{\mathsf{dag}}$ (which, as we will see, is a very rough approximation). For that a quick induction on the length of tr allows to construct a set of $|\mathsf{tr}|$ variables $Y = \{y_i\}_{i=1}^{|\mathsf{tr}|}$ and finite set of equations S such that

1. $\mu^1(\mathcal{C}) \in mgu_E(S)$
2. for all $(u =^? v) \in S$, u (resp. v) is either a subterm of a term appearing in P or a variable of Y
3. for all terms $u \in img(\Phi(\mathcal{C})\mu^1(\mathcal{C}))$, there exists u_0 subterm of a term appearing in P such that $u_0\mu^1(\mathcal{C}) = u$

The variables of Y model the fresh channel variables introduced when executing (s-In) or (s-Out) transitions, and the set of equations S collects the equality tests performed during the trace and how each variable of P is instantiated by E^1 (including by private communications). Independently from this, by induction on a straightforward algorithm to compute mgu's modulo theory, we have if E is constructor-destructor subterm convergent

$$\begin{aligned} |subterms(\sigma)| &\leqslant |subterms(S)| + |E|_{\mathsf{dag}} \times |\{t \in subterms(S) \mid root(t) \in \mathcal{F}_{\mathsf{d}}\}| \\ &\leqslant 2|subterms(S)| \times |E|_{\mathsf{dag}} \end{aligned} \qquad (\mathcal{E})$$

Altogether we therefore obtain

$$\begin{aligned} \left|\Phi(\mathcal{C})\mu^1(\mathcal{C}), \mu^1(\mathcal{C})\right|_{\mathsf{dag}} &\leqslant |subterms(P)| + 2\left|\mu^1(\mathcal{C})\right|_{\mathsf{dag}} && \text{(by 3)} \\ &\leqslant |P|_{\mathsf{dag}} + 4|subterms(S)| \times |E|_{\mathsf{dag}} && \text{(by 1 and } (\mathcal{E})\text{)} \\ &\leqslant |P|_{\mathsf{dag}} + 4(|P|_{\mathsf{dag}} + |\mathsf{tr}|)\,|E|_{\mathsf{dag}} && \text{(by 2)} \\ &\leqslant 9\,|P, E|^2_{\mathsf{dag}} \end{aligned}$$

□

3 Complexity upper bounds for equivalence properties

Section summary

Building on the results of this chapter we design *theoretical decision procedures* for equivalence properties in *coNEXP*. For that we prove the existence, whenever equivalence does not hold, of a witness of non-equivalence (in the sense of Chapter 3, Section 2) of exponential size. This proof is contructive and relies on the partition tree computed by Algorithm 1.

3.1 Complexity of trace equivalence and equivalence by session

The goal of this section is to prove the following theorem:

Theorem 6.10 *(complexity of trace equivalence and equivalence by session)*

TraceEquiv and SessEquiv are coNEXP for bounded processes and constructor-destructor subterm convergent theories.

We only prove it for trace equivalence; the proof for equivalence by session can be done along the same lines, using the arguments described in Chapter 5, Section 5. The proof for trace equivalence relies on the following arguments that were developed in previous chapters:

1. *characterring trace inclusion with partition trees:* Theorem 3.3 (Chapter 4, Section 2.2)
2. *existence of a mgs of exponential size:* Theorem 6.5
3. *soundness and completeness of the symbolic semantics:* see Chapter 3

Using these ingredients we prove the core property:

Proposition 6.11 (witness of non-trace equivalence of exponential size)

Let P_1, P_2 be two plain processes w.r.t. a constructor-destructor subterm convergent theory E. The following points are equivalent:

1. $P_1 \not\sqsubseteq_t P_2$
2. there exists a trace $t : P_1 \stackrel{\mathsf{tr}}{\Rightarrow} A_1$ such that $|t|_{\mathsf{dag}}$ is exponential in $|P, Q, E|_{\mathsf{dag}}$ and for all $P_2 \stackrel{\mathsf{tr}}{\Rightarrow} A_2$, $A_1 \not\sim A_2$.

Proof. The proof of 2⇒1 is trivial and we therefore focus on 1⇒2. Let us assume that $P_1 \not\sqsubseteq_t P_2$, and let $T \in \mathsf{PTree}(P_1, P_2)$ the partition tree computed by Algorithm 1. By Theorem 3.3 we obtain a partition-tree trace $P_1 \stackrel{\mathsf{tr}}{\Rightarrow}_T (\mathcal{P}, \mathcal{C}), n$ such that there exist no traces of the form $P_2 \stackrel{\mathsf{tr}}{\Rightarrow}_T (\mathcal{P}', \mathcal{C}'), n$. But by Theorem 6.5 we know that the (DAG) size of $mgs(n)$ is of exponential in $|P, Q, E|_{\mathsf{dag}}$, which gives a solution $(\Sigma, \sigma) \in Sol^{\pi(n)}(\mathcal{C})$ of exponential size as well by definition of a mgs.

Let us then consider the trace $t : P_1 \stackrel{\mathsf{tr}\Sigma}{\Longrightarrow} (\mathcal{P}\sigma, \Phi(\mathcal{C})\sigma\downarrow)$ (that exists by soundness of the symbolic semantics) and show that it satisfies the conditions of 2. It is indeed of exponential DAG size. Besides assume by contradiction that there exists a trace $P_2 \stackrel{\mathsf{tr}\Sigma}{\Longrightarrow} (\mathcal{Q}, \Psi)$ such that $\Phi(\mathcal{C})\sigma \sim \Psi$. By using the completeness of the symbolic semantics and the properties of the partition tree (Lemma 3.4 in Chapter 2.2), we would obtain a symbolic process $(\mathcal{P}', \mathcal{C}')$ such that $P_2 \stackrel{\mathsf{tr}}{\Rightarrow}_T (\mathcal{P}', \mathcal{C}'), n$, yielding a contradiction. □

To obtain a decidability result we also use the following result on static equivalence rephrased from [AC06]:

Proposition 6.12 (witness of non-static equivalence of polynomial size)

If two frames Φ and Ψ are not statically equivalent w.r.t. a subterm convergent theory E, there exist two recipes ξ and ζ such that $|\xi, \zeta|_{\mathsf{dag}}$ is polynomial in $|\Phi, \Psi, E|_{\mathsf{dag}}$, $\xi\Phi =_E \zeta\Phi$ and $\xi\Psi \neq_E \zeta\Psi$.

Wrapping everything together we obtain the following NEXP decision procedure for non-trace equivalence:

1. Given two processes P_1, P_2, guess an integer $i \in [\![1, 2]\!]$ and a trace $P_1 \stackrel{\mathsf{tr}}{\Rightarrow} (\mathcal{P}, \Phi)$ of exponential size. In particular, although $|dom(\Phi)| \leqslant |\mathsf{tr}|$, the sizes of the terms in $img(\Phi)$ may be exponential as well.
2. For each of the exponentially-many traces of the form $t : P_2 \stackrel{\mathsf{tr}}{\Rightarrow} (\mathcal{Q}, \Psi)$, guess two recipes ξ_t, ζ_t of exponential size.
3. if for one such trace t we do not have $\xi_t\Phi =_E \zeta_t\Phi \Leftrightarrow \xi_t\Psi =_E \zeta_t\Psi$, conclude that $P_1 \not\approx_t P_2$.

3.2 Complexity of labelled bisimilarity

The goal of this section is to prove the following theorem:

> ***Theorem 6.13*** *(complexity of labelled bisimilarity)*
> Bisimilarity is coNEXP for bounded processes and constructor-destructor subterm convergent theories.

Similarly to trace equivalence we build on the results of the previous chapters, this time using the characterisation of labelled bisimilarity based on symbolic witnesses (Theorem 3.7 in Chapter 3, Section 2.3). Given a partition tree with most general solutions of exponential size, our goal is therefore to derive from it a symbolic witness of non-equivalence and a solution of this witness (Definition 3.14 in Chapter 3, Section 2.3), both of exponential size as well.

Proposition 6.14 (witness of non-labelled bisimilarity of exponential size)
Let P_1, P_2 be two plain processes w.r.t. a constructor-destructor subterm convergent theory E. The following points are equivalent:
1 $P_1 \not\approx_l P_2$
2 there exists a witness w for (P_1, P_2) such that $|\mathsf{w}|_{\mathsf{dag}}$ is exponential in $|P, Q, E|_{\mathsf{dag}}$.

Proof. The proof of 2⇒1 is trivial and we therefore focus on 1⇒2. Let us assume that $P_1 \not\approx_l P_2$, and let $T \in \mathsf{PTree}(P_1, P_2)$ the partition tree computed by Algorithm 1. By Theorem 3.7 we obtain a symbolic witness w_s for $(P_0, P_1, root(T))$ such that $Sol(\mathsf{w}_s) \neq \varnothing$, and it sufficies to prove that there exists a solution of w_s of exponential size (where the size of a solution f_{sol} is $\sum_{N \in dom(f_{\mathsf{sol}})} |f_{\mathsf{sol}}(N)|_{\mathsf{dag}}$). More precisely we construct by induction on w_s a function f mapping the nodes of w_s to second-order substitutions (not necessarily ground) such that:
1 $(f_{mgs} f) \in Sol(\mathsf{w}_s)$, where $f_{mgs}(S, n) = mgs(n)$ and the notation $f = gh$ is defined by $f(N) = g(N)h(N)$ for all nodes N of w_s
2 for all $f_{\mathsf{sol}} \in Sol(\mathsf{w}_s)$, there exists f' such that $f_{\mathsf{sol}} = f_{mgs} f f'$
In particular since f_{mgs} is of exponential size by Theorem 6.5, it sufficies to ensure that f is of exponential size as well.

▷ *case 1:* w_s *is reduced to a leaf* N.
Then it sufficies to choose $f(N) = id$.

▷ *case 2:* w_s *has a root labelled* (S, n) *and children* $N_1, \ldots, N_p$ *labelled* $(S_1, n'), \ldots, (S_p, n')$
Let us write $S = \{A_0, A_1\}$ with, by definition, a symbolic trace $A_0 \xrightarrow{\alpha}_{\mathsf{s}} A'_0$ such that each trace $A_1 \stackrel{\bar{\alpha}}{\Rightarrow}_{\mathsf{s}} A^i_1$ corresponds to a child $S_i = \{A'_0, A^i_1\}$. We apply the induction hypothesis to the children to obtain their respective functions $f_1, \ldots, f_p$. We recall that $Sol(\mathsf{w}_s) \neq \varnothing$ by hypothesis and that all solutions f_{sol} verify $f_{\mathsf{sol}}(N_1) = \cdots = f_{\mathsf{sol}}(N_p)$; thus, since by induction hypothesis all solutions of w_s are instances of $f_{mgs} f_i$, we obtain:

$$mgu(mgs(n')f_1(N_1)\varrho_1 \wedge \ldots \wedge mgs(n')f_p(N_p)\varrho_p) \neq \bot$$

for $\varrho_1, \ldots, \varrho_p$ fresh variables renamings of $img(f_1(N_1)), \ldots, img(f_p(N_p))$, respectively. In

particular, assuming without loss of generality that all the $f_i(N_i)$ have the same domain $(vars^2(n') \setminus dom(mgs(n'))) \cup img(mgs(n'))$, we can write

$$\Sigma = mgu(f_1(N_1)\varrho_1 \wedge \ldots \wedge f_p(N_p)\varrho_p) \neq \bot$$

Note that this mgu is only polynomially bigger than each $f_i(N_i)$. Since $mgs(n')$ is an instance of $mgs(n)$, we also let Σ_0 such that $mgs(n') = mgs(n)\Sigma_0$. We then conclude the proof by defining f as follows:

1. $f(root(\mathsf{w}_s)) = (\Sigma_0\Sigma)_{|vars^2(n)}$
2. for all $i \in [\![1, p]\!]$, for all nodes N in the subtree of w_s rooted in N_i, $f(N) = f_i\Sigma$. □

4 coNEXP hardness: a reduction from succinct satisfiability

> **Section summary**
>
> We show that the coNEXP upper bound proved in this chapter is tight, i.e. deciding the three equivalences are *coNEXP hard in general*. This result holds in various restrictive settings, e.g. without *else branches* or *private channels* or with a *fixed* set of cryptographic primitives.

4.1 Extensions of the calculus

As we explained in Chapter 1, Section 2.3 many extensions can be encodeed in the original calculus for modelling convenience. However, as soon as complexity is involved, one should be careful that using these encodings does not affect the complexity of deciding the related decision problems. In this section we introduce various syntax extensions that will be useful when doing our reduction of coNEXP hardness to process equivalences, and formalise a polynomial-size encoding in the original calculus.

Internal non-deterministic choice We recall the encoding of non-deterministic choice introduced in Chapter 1, Section 2.3. We formally defined it by a process transformation $[\![\cdot]\!]$:

$$[\![P + Q]\!] \stackrel{def}{=} \overline{s}\langle s\rangle \mid s(x).[\![P]\!] \mid s(y).[\![Q]\!] \qquad \text{where } s \in \mathcal{N} \text{ and } x, y \in \mathcal{X}^1 \text{ are fresh} \tag{6.1}$$

and all other cases of the syntax are handled as homomorphic extensions of $[\![\cdot]\!]$. As for the parallel operator we will sometimes use the big operator $\sum$ assuming right-associativity. The correctness of this translation with respect to $\approx_t$ and $\approx_l$ will be stated later on in this section.

We also introduce the $\mathtt{Choose}(x)$ construct which non-deterministically assigns either 0 or 1 to x. $\mathtt{Choose}(x).P$ silently reduces to either $P\{x \mapsto \mathsf{0}\}$ or $P\{x \mapsto \mathsf{1}\}$ and $\mathtt{Choose}(\vec{x}).P$ is defined as $\mathtt{Choose}(x_1).\mathtt{Choose}(x_2)\ldots\mathtt{Choose}(x_n).P$ where $\vec{x} = x_1, \cdots, x_n$. Formally, we extend the operational semantics with the rule

$$(\mathcal{P} \cup \{\!\{\mathtt{Choose}(x).P\}\!\}, \Phi) \xrightarrow{\varepsilon} (\mathcal{P} \cup \{\!\{P\{x \mapsto \mathsf{0}\}\}\!\}, \Phi) \tag{Choose-0}$$

$$(\mathcal{P} \cup \{\!\{\mathtt{Choose}(x).P\}\!\}, \Phi) \xrightarrow{\varepsilon} (\mathcal{P} \cup \{\!\{P\{x \mapsto \mathsf{1}\}\}\!\}, \Phi) \tag{Choose-1}$$

and define

$$[\![\mathtt{Choose}(y).P]\!] \stackrel{def}{=} (\overline{d}\langle 0\rangle + \overline{d}\langle 1\rangle) \mid d(y).[\![P]\!] \quad \text{with } d \in \mathcal{N} \text{ is fresh}$$

Boolean circuits and formulae Complete problems in complexity theory often involve boolean formulae (e.g., SAT or QBF). The ability to evaluate boolean formulae, or boolean circuits in general, within the applied π-calculus is therefore curcial. We can implement such a feature by the means of private channels and internal communication: each edge of a boolean circuit Γ indeed mimics a channel transmitting a boolean over a network (Figure 6.1).

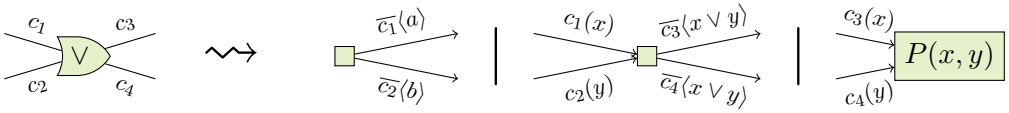


Figure 6.1 Simulation of an OR-gate within the applied π-calculus

Formally, the essence of circuits lies in so-called *logical gates* which are boolean functions with at most two inputs. We assume without loss of generality that the gate has at most two (identical) outputs, to be given as input to other gates. Logical gates usually range over the constants $\mathsf{0}$ and $\mathsf{1}$ and the predicates $\wedge$, $\vee$ and $\neg$ with the usual truth tables but we may use other common operators such as $=$. From that a *boolean circuit* is an acyclic graph of logical gates: each input (resp. output) of a gate is either isolated or connected to a unique output (resp. input) of an other gate, which defines the edges of this graph.

Such a circuit Γ with m isolated inputs and p isolated outputs thus models a boolean function $\Gamma : \mathbb{B}^m \to \mathbb{B}^n$ (where $\mathbb{B} = \{\mathsf{0},\mathsf{1}\}$). We write $(c_1, c_2, g, c_3, c_4) \in \Gamma$ to state that $g : \mathbb{B}^2 \to \mathbb{B}$ is a gate of Γ whose inputs are passed through edges c_1 and c_2 and whose output is sent to edges c_3 and c_4. This notation is naturally lifted to other in-outdegrees.

Embedding into the calculus The syntax of plain processes is now extended with the construction $x_1, \cdots, x_n \leftarrow \Gamma(b_1, \cdots, b_m).P$ where $\Gamma : \mathbb{B}^m \to \mathbb{B}^n$ is a circuit, $x_1, \ldots, x_n$ variables and $b_1, \cdots, b_m$ terms. We fix two distinct terms $\mathsf{0}, \mathsf{1} \in \mathcal{F}_0$ to model $\mathbb{B}$ within the calculus, and the labelled operational semantics is extended with the rule:

$$(\mathcal{P} \cup \{\!\{\vec{x} \leftarrow \Gamma(\vec{b}).P\}\!\}, \Phi) \xrightarrow{\varepsilon} (\mathcal{P} \cup \{\!\{P\{\vec{x} \mapsto \Gamma(\vec{b}\downarrow)\}\}\!\}, \Phi) \quad \text{if } \mathsf{msg}(\vec{b}) \text{ and } \vec{b}\downarrow \subseteq \mathbb{B} \qquad (\textsc{Valuate})$$

Now we have to extend the definition of $[\![\cdot]\!]$ (previous subsection) to handle the new operator. For simplicity we only consider the case where gates have two inputs and two outputs: handling lower arities is straightforward. If $(c_1, c_2, g, c_3, c_4) \in \Gamma$, we first define:

$$[\![c_1, c_2, g, c_3, c_4]\!] \stackrel{def}{=} c_1(x).c_2(y). \prod_{b,b' \in \mathbb{B}} \mathsf{if}\ x = b\ \mathsf{then}\ \mathsf{if}\ y = b'\ \mathsf{then}\ (\overline{c_3}\langle g(b,b')\rangle \mid \overline{c_4}\langle g(b,b')\rangle)$$

where $c_1, c_2, c_3, c_4 \in \mathcal{N}$ (assuming that different circuits in a process do not share edges). To sum it up, we simply see circuit edges as private channels and simulate the logical flow of the gate. It is then easily extended:

$$\left[\!\!\left[\vec{x} \leftarrow \Gamma(\vec{b}).P\right]\!\!\right] \stackrel{def}{=} \left(\prod_{k=1}^{m} \overline{c_{i_k}}\langle b_k\rangle\right) \mid \left(\prod_{(c_1,c_2,g,c_3,c_4)\in\Gamma} [\![c_1, c_2, g, c_3, c_4]\!]\right) \mid c_{o_1}(x_1) \ldots c_{o_n}(x_n).\, [\![P]\!]$$

where $(c_{i_k})_{k=1}^{m}$ (resp. $(c_{o_k})_{k=1}^{n}$) are the isolated input (resp. output) edges of Γ. Note that when b and b' are fixed booleans, $g(b,b')$ denotes the boolean obtained from the truth table of g: we emphasise that g is *not* a function symbol of the signature $\mathcal{F}$.

▶ *Remark: simplifying assumption*
We assume that every input of a circuit goes through at least one gate and has at least one output. This is to avoid irrelevant side cases in proofs. ◀

Correctness of the translation Now we dispose of an extended syntax and semantics as well as a mapping $[\![\cdot]\!]$ removing the new constructors from a process. The correctness of this translation is proven in Appendix D:

Proposition 6.15 (correctness of the encodings)
Let $\approx_t^+$ and $\approx_l^+$ be the notions of trace equivalence and labelled bisimilarity over the extended calculus (the flag $^+$ being omitted outside of this lemma). For all extended processes $A = (\mathcal{P}, \Phi)$, the translation $[\![A]\!] = ([\![\mathcal{P}]\!], \Phi) = (\{\!\{[\![P]\!] \mid P \in \mathcal{P}\}\!\}, \Phi)$ can be computed in polynomial time, $A \approx_t^+ [\![A]\!]$ and $A \approx_l^+ [\![A]\!]$.

▶ *Remark: stability of common fragments*
As the finite and pure fragments of the applied π-calculus are closed under $[\![\cdot]\!]$, sums and circuits can be safely used within any intersection of such fragments. The encoding does not use else branches either. ◀

4.2 Reduction of SuccinctSAT to process equivalence

Consider an instance of SuccinctSAT, Γ, with $m+2$ inputs and $n+1$ outputs and we design $\mathcal{F}$, E subterm destructor and A and B positive processes such that $A \not\approx_t B$ *iff* $A \not\approx_l B$ *iff* $[\![\Gamma]\!]_\varphi$ is satisfiable.

Term algebra Terms are built over the following signature:

$$\begin{array}{lll} \mathcal{F} \stackrel{def}{=} & \mathsf{0},\ \mathsf{1}, & \text{(booleans } \mathbb{B}) \\ & \mathsf{Node}/2,\ \pi/2, & \text{(binary trees)} \\ & \mathsf{h}/2, & \text{(one-way binary hash)} \\ & \mathsf{h_N}/2,\ \mathsf{h_B}/2,\ \mathsf{Test_N}/1,\ \mathsf{Test_B}/1 & \text{(testable binary hashes)} \end{array}$$

We equip this term algebra with the rewriting system E containing the following rules modelling subtree extraction (for binary trees) and argument testing (for hashes):

$$\pi(\mathsf{Node}(x,y),\mathsf{0}) \to x \qquad \pi(\mathsf{Node}(x,y),\mathsf{1}) \to y$$
$$\mathsf{Test_N}(\mathsf{h_N}(\mathsf{Node}(x,y),z)) \to \mathsf{1} \qquad \mathsf{Test_B}(\mathsf{h_B}(\mathsf{0},z)) \to \mathsf{1} \qquad \mathsf{Test_B}(\mathsf{h_B}(\mathsf{1},z)) \to \mathsf{1}$$

In particular E is subterm and destructor, the destructor symbols being π, $\mathsf{Test_N}$ and $\mathsf{Test_B}$. We will also use a shortcut for recursive subtree extraction: if ℓ is a finite sequence of first-order terms, the notation $t_{|\ell}$ is inductively defined by:

$$t_{|\varepsilon} \stackrel{def}{=} t \qquad\qquad t_{|b\cdot\ell} \stackrel{def}{=} \pi(t,b)_{|\ell}$$

Core of the reduction Let us give the intuition behind the construction before diving into the formalism. Recall that we are studying a formula in CNF $[\![\Gamma]\!]_\varphi$ with 2^n variables and 2^m clauses. In particular, given a valuation of its 2^n variables, we can verify in non-deterministic polynomial time in n, m that it falsifies $[\![\Gamma]\!]_\varphi$:

1. guess an integer $i \in [\![0, 2^m - 1]\!]$ as a sequence of m bits;
2. obtain the three literals of the i^{th} clause of $[\![\Gamma]\!]_\varphi$ (requiring three runs of the circuit Γ) and verify that the valuation falsifies the disjunction of the three literals.

This non-deterministic verification is the essence our reduction. In the actual processes:

1. a process $\texttt{CheckTree}(x)$ checks that x is a correct encoding of a valuation, that is, that x is a complete binary tree of height n whose leaves are booleans;
2. a process $\texttt{CheckSat}(x)$ implements the points *1.* and *2.* above.

All of this is then formulated as equivalence properties within A and B (see the intermediary lemmas in the next paragraph for details). Intuitively, we want to express the following statement by equivalence properties: "*for all term x, either x is not an encoding of a valuation or falsifies a clause of $[\![\Gamma]\!]_\varphi$*". A schematised definition is proposed in Figure 6.2.

▶ *Remark: reduction for equivalence by session*
The construction for equivalence by session is more technical due to the requirement that equivalent processes share the same structure. We focus in this section on the cases of trace equivalence and labelled bisimilarity, and refer to Appendix E for equivalence by session. ◀

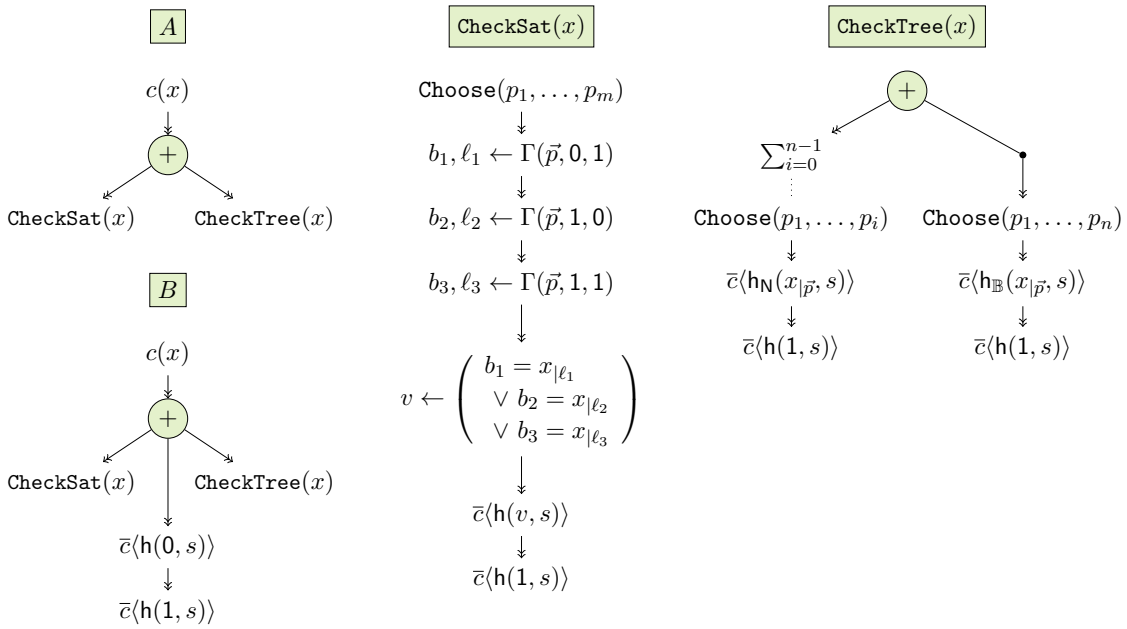


Figure 6.2 Informal definition of A and B

Formal construction Let us now define the processes depicted in Figure 6.2 properly; note that all the proofs about the correctness of this construction are relegated to Appendix D but we still state several intermediary lemmas in order to highlight the proof structure. But

first of all, let us give a name to a frame which is at the core of our reduction:

$$\Phi_0 = \{\mathsf{ax}_1 \mapsto \mathsf{h}(0,s),\ \mathsf{ax}_2 \mapsto \mathsf{h}(1,s)\}$$

Φ_0 is reached after executing the central branch of B and everything is about knowing under which conditions a frame statically equivalent to Φ_0 can be reached in A. Let us define the processes themselves now. We fix $s \in \mathcal{N}$ and define, if x is a protocol term:

$$\begin{aligned}\mathtt{CheckTree}(x) \stackrel{def}{=} \sum_{i=0}^{n-1} \big(\ \mathtt{Choose}(p_1,\dots,p_i).\ \overline{c}\langle \mathsf{h}_{\mathsf{N}}(x_{|p_1\cdots p_i},s)\rangle.\ \overline{c}\langle \mathsf{h}(1,s)\rangle\ \big)\\ + \mathtt{Choose}(p_1,\dots,p_n).\ \overline{c}\langle \mathsf{h}_{\mathbb{B}}(x_{|p_1\cdots p_n},s)\rangle.\ \overline{c}\langle \mathsf{h}(1,s)\rangle\end{aligned}$$

Proposition 6.16 (correctness of the tree checker)

Let x be a message which is not a complete binary tree of height n with boolean leaves. Then there exists a reduction $\mathtt{CheckTree}(x) \stackrel{\varepsilon}{\Rightarrow} (\{\!\{P\}\!\}, \varnothing)$ such that $P \approx_l \overline{c}\langle \mathsf{h}(0,s)\rangle.\overline{c}\langle \mathsf{h}(1,s)\rangle$.

Now let us move on to $\mathtt{CheckSat}(x)$. This process binds a lot of variables:

1. $\vec{p} = p_1,\dots,p_m$ models the non-deterministic choice of a clause number in $[\![0, 2^m - 1]\!]$;
2. b_i, ℓ_i, $i \in [\![1,3]\!]$, where ℓ_i is a sequence of n variables, model the literals of the clause chosen above (b_i is the negation bit and ℓ_i the identifier of the variable);
3. v stores whether the chosen clause is satisfied by the valuation modelled by x.

$$\begin{aligned}\mathtt{CheckSat}(x) \stackrel{def}{=}\ & \mathtt{Choose}(\vec{p}).\\ & b_1,\ell_1 \leftarrow \Gamma(\vec{p},0,1).\\ & b_2,\ell_2 \leftarrow \Gamma(\vec{p},1,0).\\ & b_3,\ell_3 \leftarrow \Gamma(\vec{p},1,1).\\ & v \leftarrow (b_1 = x_{|\ell_1} \vee\ b_2 = x_{|\ell_2} \vee\ b_3 = x_{|\ell_3}).\\ & \overline{c}\langle \mathsf{h}(v,s)\rangle.\overline{c}\langle \mathsf{h}(1,s)\rangle\end{aligned}$$

Proposition 6.17 (correctness of the sat checker)

Let x be a complete binary tree of height n whose leaves are booleans, and val_x be the valuation mapping the variable number i of $[\![\Gamma]\!]_\varphi$ to $x_{|p_1\cdots p_n} \in \mathbb{B}$ where $p_1\cdots p_n$ is the binary representation of i (i.e., $i = \sum_{k=1}^{n} p_k 2^{k-1}$). If val_x does not satisfy $[\![\Gamma]\!]_\varphi$ then there exists $\mathtt{CheckSat}(x) \stackrel{\varepsilon}{\Rightarrow} P$ such that $P \approx_l \overline{c}\langle \mathsf{h}(0,s)\rangle.\overline{c}\langle \mathsf{h}(1,s)\rangle$.

We can finally wrap up everything by defining A and B and stating the last part of the correctness theorem. We recall that all the proofs can be found in Appendix D.

$$\begin{aligned}A &\stackrel{def}{=} c(x).(\mathtt{CheckSat}(x)\ +\ \mathtt{CheckTree}(x))\\ B &\stackrel{def}{=} c(x).(\mathtt{CheckSat}(x)\ +\ \mathtt{CheckTree}(x)\ +\ \overline{c}\langle \mathsf{h}(0,s)\rangle.\overline{c}\langle \mathsf{h}(1,s)\rangle)\end{aligned}$$

Proposition 6.18 (correctness of the reduction)

$[\![\Gamma]\!]_\varphi$ is satisfiable *iff* $A \not\approx_t B$ *iff* $A \not\approx_l B$.

As a conclusion we obtain the coNEXP hardness of the three equivalence properties for constructor-destructor subterm convergent theories. This is stated by the theorem below, which additionally puts an emphasis on the fact that the theory used in our reduction is constant, that is, it does not depend on Γ.

Theorem 6.19 *(hardness of equivalences)*

There exists a fixed constructor-destructor subterm convergent theory E such that the decision problems E–TraceEquiv, E–SessEquiv and E–Bisimilarity are coNEXP hard for bounded positive processes.

As mentioned earlier, the more involved construction for equivalence by session is provided in Appendix E.

Chapter 7:

The big picture: survey and new results

Summary.

In the previous chapter we proved the equivalence problems to be coNEXP complete for a quite large class of processes; however they may naturally be decidable more efficiently in *specific subclasses* of bounded (or even unbounded) processes. In this chapter we survey in more details, and cast in a *common formalism*, the existing complexity results about the decision of equivalence properties in the applied pi-calculus. This effort allowed us to identify subtle differences across the literature on how the problems were stated, sometimes influencing the complexity. We also include a couple of *new results* (see Table 7.1 in Section 5 for a summary).

NB. *We found it interesting to outline some proofs of the surveyed results, even when they are not novel. However due to the number of results of this chapter, for readability reasons most of the proofs of our own results are only sketched in this chapter. Detailed proofs in Appendix E.*

1 Complexity of static equivalence

> ***Section summary***
>
> We first study the complexity of the StatEquiv problem. The main result is that it is P for subterm convergent theories provided they are considered as a *constant of the problem*, and coNP complete without this assumption.

1.1 Subterm convergent theories

We first focus on the class of subterm convergent theories. Historically, the complexity of static equivalence has only been considered for *fixed* theories [AC06, Bau07], that is, the theory was not part of the input of the problem and its size was seen as a constant in the complexity analysis. We recall that this corresponds to the decision problem E–StatEquiv defined in Chapter 1, Section 4.3. This was consistent with most formalisms and verification tools at the time, which would not allow for user-defined theories and only consider a fixed set of cryptographic primitives, such as in the spi-calculus for example [AG99]. For example:

> ***Theorem 7.1*** *(static equivalence for fixed subterm convergent theories [AC06])*
>
> For all subterm convergent theories E, E–StatEquiv is in P.

However a generic P completeness result would not make sense in the context of a fixed theory, since choosing one theory or another may influence the effective complexity of the problem. This is typically illustrated by the following result:

Theorem 7.2 *(static equivalence in the pure fragment)*
In the pure pi-calculus (i.e., with the empty theory) StatEquiv is in LOGSPACE.

Proof. In the pure π-calculus, two frames Φ and Φ' of same domain can only contain constants and names. Hence since $\mathcal{F} = \varnothing$ static equivalence can be characterised as follows:

$$\Phi \sim \Phi' \qquad \textit{iff} \qquad dom(\Phi) = dom(\Phi') \wedge \bigwedge_{\xi,\xi' \in T} \left(\xi\Phi = \xi'\Phi \Leftrightarrow \xi\Phi' = \xi'\Phi'\right)$$

where $T = dom(\Phi) \cup img(\Phi, \Phi')$. This characterisation is easily implementable by a quadratic procedure storing a finite number of counters only, each of which being bounded by the size of T which is linear in the size of the problem and whose binary representation is of logarithmic size. In particular, note that there is no need for T to be computed and stored (since it can be read on-the-fly directly on the input) and analogously, testing an equality $\xi\Phi = \xi'\Phi$ does not require additional storage space. We therefore obtain the expected LOGSPACE bound. □

Despite the potential variations of complexity when considering very simple theories as above, we can show that the P bound is optimal in the following sense in the context of cryptographic protocols:

Theorem 7.3 *(hardness of static equivalence for fixed theories)*
For all fixed theories E containing symmetric encryption, E–StatEquiv is P hard.

Proof sketch. We proceed by reduction from HornSAT. Let X be the set of variables of a Horn formula $\varphi = C_1 \wedge \ldots \wedge C_n$, and k_x be names for all $x \in X \cup \{\bot\}$. Then to each clause $C_i = x_1, \ldots, x_n \Rightarrow x$, $x \in X \cup \{\bot\}$ we associate the term

$$t_{C_i} = \mathsf{senc}(\ldots \mathsf{senc}(\mathsf{senc}(k_x, k_{x_1}), k_{x_2}), \ldots, k_{x_n})\,.$$

Putting k_x under several layers of encryption ensures that k_x is deducible if all the keys $k_{x_1}, \ldots, k_{x_n}$ are deducible as well. In particular $k_\bot$ is deducible from the terms $t_{C_1}, \ldots, t_{C_n}$ *iff* the formula φ is unsatisfiable. Then if $0, 1 \in \mathcal{F}_0$ and $\Phi = \{\mathsf{ax}_1 \mapsto t_{C_1}, \ldots, \mathsf{ax}_n \mapsto t_{C_n}\}$, the following two frames are statically equivalent *iff* φ is satisfiable:

$$\Phi \cup \{\mathsf{ax} \mapsto \mathsf{senc}(0, k_\bot)\} \qquad\qquad \Phi \cup \{\mathsf{ax} \mapsto \mathsf{senc}(1, k_\bot)\}$$ □

However automated tools have improved since then and some provers like Kiss [CDK12], Yapa [BCD13] or Fast [CBC11] are able to handle user-defined theories. It is therefore interesting today to account for the size of the theory in the complexity analysis. When not considering the theory as a constant of the problem the complexity is upgraded as follows:

Theorem 7.4 *(static equivalence for subterm convergent theories)*
StatEquiv is coNP complete for subterm convergent theories.

Proof sketch. The results of [AC06] cited for Theorem 7.1 actually justify that StatEquiv is coNP. More precisely they imply that whenever $\Phi \not\sim \Psi$ for a subterm convergent theory, there exist two recipes ξ, ζ such that $|\xi, \zeta|_{\mathsf{dag}}$ is polynomial in $|\Phi, \Psi, E|_{\mathsf{dag}}$ and $\xi\Phi =_E \zeta\Phi$ and $\xi\Psi \neq_E \zeta\Psi$. In particular a straightforward coNP procedure consists of guessing two recipes ξ and ζ of polynomial size and verifying whether $\xi\Phi =_E \zeta\Phi$ and $\xi\Psi \neq_E \zeta\Psi$ (which can be done in polynomial time in $|\Phi, \Psi, E|_{\mathsf{dag}}$ for subterm convergent theories).

We now prove that non-equivalence is NP hard by reduction from SAT. We let φ a SAT formula and construct a theory E and two frames Φ, Ψ such that $\Phi \sim \Psi$ *iff* φ is satisfiable. For that we consider $\mathsf{0}, \mathsf{1} \in \mathcal{F}_0$, $k \in \mathcal{N}$, $f/2, g/2 \in \mathcal{F}_{\mathsf{c}}$ and the two frames

$$\Phi = \{\mathsf{ax}_0 \mapsto f(\mathsf{0}, k), \mathsf{ax}_1 \mapsto f(\mathsf{1}, k)\} \qquad \Psi = \{\mathsf{ax}_0 \mapsto g(\mathsf{0}, k), \mathsf{ax}_1 \mapsto g(\mathsf{1}, k)\}$$

Interpreting $\mathsf{0}$ and $\mathsf{1}$ as the booleans false and true, Φ and Ψ intuitively point to terms that can be seen as booleans *but* that can only be accessed by reference through the axioms $\mathsf{ax}_0, \mathsf{ax}_1$. For example, since k is a name the only recipes permitting to deduce $f(\mathsf{0}, k)$ will use ax_0 in Φ. Given the variables $x_1, \ldots, x_n$ of φ we then add another symbol $\mathsf{eval}/n \in \mathcal{F}_{\mathsf{d}}$ and rewrite rules so that the following points are equivalent for all valuations $v : \{x_1, \ldots, x_n\} \to \mathbb{B}$ of φ: 1 v falsifies φ, 2 $\mathsf{eval}(g(v(x_1), k), \ldots, g(v(x_n), k)) \to \mathsf{0}$. Such a theory can be defined by the p rewrite rules:

$$\mathsf{eval}(\mathsf{g}(t_1^i, y), \ldots, \mathsf{g}(t_n^i, y)) \to \mathsf{0}$$

where $1 \leqslant i \leqslant p$ and if C_i is the i^{th} clause of φ:

$$t_j^i = \begin{cases} x_j & \text{if } x_j \text{ does not appear in } C_i \\ \mathsf{0} & \text{if } x_j \text{ appears positively in } C_i \\ \mathsf{1} & \text{if } x_j \text{ appears negatively in } C_i \end{cases}$$

Note that this definition assumes that no clause of φ contains both a litteral and its negation. Such clauses are tautological and can be removed by a preprocessing step in LOGSPACE, inducing no loss of generality. Then if we add the rewrite rule

$$\mathsf{eval}(\mathsf{f}(x_1, y), \ldots, \mathsf{f}(x_n, y)) \to \mathsf{0}$$

we obtain that φ is satisfiable *iff* $\Phi \not\sim \Psi$ (proof formalised in Appendix E). □

1.2 Beyond subterm convergence

As rewriting is Turing complete, unsurprisingly static equivalence is undecidable in general for convergent rewriting systems [AC06]. It is also proved in [AC06] that Deducibility reduces to StatEquiv. As a consequence, the results of [ANR07] imply that static equivalence is also undecidable for so-called *optimally-reducing* rewrite systems, a subclass of rewrite systems that have the finite-variant property [CCCK16].

Some complexity results also exist outside of subterm theories. We can for example mention the decidability in polynomial time of *monoidal theories* [CD07], combination of disjoint theories where static equivalence is decidable [CD12], or *group theories* [DKP12]. It is also proved in [CDK12] that static equivalence was in P for theories modelling *blind signatures*, *trapdoor commitment* schemes and *malleable encryption*.

2 Complexity of dynamic equivalences

Section summary

We survey or prove some complexity results for trace equivalence and labelled bisimilarity in addition of the coNEXP completeness result of Chapter 6. We cover the case of the *pure pi-calculus* in the bounded fragment for the sake of completeness, and then study various classes of *unbounded processes*.

2.1 Classical fragments of the calculus

In addition to the assumptions on the theory (e.g., subterm convergence), there are several common restrictions made on the processes to obtain decidability.

Conditionals and patterns We first present some classes of processes defined by restrictions on the conditionals. The first one excludes negative tests [Bau07, CCD13, CKR18a]:

Definition 7.1 (positive fragment)
For succinctness we often write $[u = v]\,P$ instead of $\mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ 0$. We say that a process is *positive* if all of its conditionals have trivial else branches like this one.

Another common retriction consists of only allowing tests that are performed by *patterns* on input variables:

Definition 7.2 (patterned fragment)
We say that a process is patterned if it contains no conditionals but may use the patterned-inputs syntax extension (as defined in Chapter 1, Section 2.3).

We recall that a condition for using a pattern is that it can be simulated by conditionals without else branches, in particular the patterned fragment is a subset of the positive fragment.

Ping pong protocols These protocols [CCD15b, DY81, HS03] consist of an unbounded number of parallel processes receiving one message and sending a reply. Although the precise formalisms may differ from one work to another, the mechanisms at stake are essentially captured by processes $P = !\,P_1 \mid \cdots \mid !\,P_n$ where each P_i can be written under the form

$$P_i = c_i(x).\,[u_1^i = v_1^i] \cdots [u_{n_i}^i = v_{n_i}^i]\mathsf{new}\ k_1 \cdots \mathsf{new}\ k_{r_i}.\,\overline{c_i}\langle w_i \rangle$$

In particular ping-pong protocols are positive.

Simple processes A common middleground in terms of expressivity and decidability is the class of simple processes, for example studied in [CCD13, CCD15a]. Intuitively, they consist of a sequence of parallel processes that operate each on a distinct, public channel—including replicated processes that generate dynamically a fresh channel for each copy. Formally they are of the form

$$P_1 \mid \cdots \mid P_m \mid !^{ch}\, P_{m+1} \mid \cdots \mid !^{ch}\, P_n \qquad \text{with } !^{ch}\, P = \,!\, \mathsf{new}\ c_P.\, \overline{c'_P}\langle c_P\rangle.\, P$$

where each P_i does not contain parallel operators nor replications and uses a unique, distinct communication channel c_{P_i}, and

$$!^{ch}\, P = \,!\, \mathsf{new}\ c_P.\, \overline{c'_P}\langle c_P\rangle.\, P\,.$$

Unlike ping pong protocols, each parallel process may input several messages and output messages that depend on several previous inputs. Determinate process (as defined in Chapter 5, see more discussions on this later in this chapter) are a generalisation of simple processes.

2.2 Complexity results: bounded fragment

The bounded fragment is a common restriction to study decidability, as removing replication bounds the length of traces. However, as the attacker still has an unbounded number of possibilities for generating inputs, the transition system still has infinite branching in general. Besides additional restrictions are necessary on the cryptographic primitives (at least because static equivalence is undecidable in general). Typically we mention once again the main result of Chapter 6:

> ***Theorem 6.1*** *(complexity of equivalences)*
> The decision problems TraceEquiv, Bisimilarity and SessEquiv are coNEXP complete for bounded processes and constructor-destructor subterm convergent theories.

We recall that the decision procedures use a dedicated constraint solving approach to show that, whenever equivalence is violated, there exists a non-equivalence witness of exponential DAG size, thus proving non-equivalence to be decidable in NEXP. As before, we may also study the problem for fixed theories to investigate their influence on the complexity. Typically with the empty theory:

> ***Theorem 7.5*** *(equivalences in the pure pi calculus)*
> In the pure pi-calculus, Bisimilarity (resp. TraceEquiv and SessEquiv) is PSPACE complete (resp. Π_2 complete) for bounded processes, and for bounded positive processes.

Although the PSPACE and Π_2 upper bounds follow from the definitions in a rather straightforward manner (see the definition of the polynomial hierarchy in Chapter 1), the corresponding complexity lower bounds rely on a non-trivial reduction to QBF_2 and QBF that we detail in Appendix E, Section 2. However, unlike static equivalence, fixing the theory does not make it possible to obtain a better bound than the general one for all theories due to the reduction presented in Chapter 6, Section 4:

Theorem 6.19 *(hardness of equivalences)*
There exists a fixed constructor-destructor subterm convergent theory E such that the decision problems E–TraceEquiv, E–SessEquiv and E–Bisimilarity are coNEXP hard for bounded positive processes.

The reduction indeed relied on a fixed theory which, we recall, modelled some forms of binary trees and testable functions. Actually we also prove in Appendix E that, provided we discard the positivity requirement, it is possible to manage the proof with a theory limited to symmetric encryption and pairs. This shows that the problem remains theoretically hard even with a minimal theory. Besides, in the case of trace equivalence and equivalence by session, the reduction can be done without private channels (whereas the reduction of Chapter 6 heavily relies on private communications, which may give the false intuition that they are the cause of the high complexity).

2.3 Complexity results: unbounded fragment

Equivalence is undecidable in general since the calculus is Turing complete even for simple theories. For example, Hüttel [Hüt03] shows that Minsky's two counter machines can be simulated within the spi-calculus (and hence the applied pi-calculus with symmetric encryption only). It is not difficult to adapt the proof to a simulation using only a free symbol, i.e., a function symbol h of positive arity and an empty rewrite system. These two encodings can be performed within the *finite-control fragment*, typically not Turing complete in the pure pi-calculus (i.e., without this free function symbol) [Dam97].

For ping pong protocols While equivalence is undecidable for ping-pong protocols [CCD15b, HS03] some decidability results exist under additional assumptions. For example [HS03] studies a problem that can be described in our model essentially as Bisimilarity for ping-pong protocols with 2 participants or less (namely $n \leqslant 2$ with the notations of the definition). This is proved decidable under some model-specific assumptions that we do not detail here. We also mention a result for patterned ping-pong protocols (cf Section 2.1) without a limit on the number of participants [CCD15b].

Definition 7.3 (deterministic ping-pong protocol)
Given a constructor-destructor theory, a ping-pong protocol P is *deterministic* when each P_i (using the same notations as the definition) can be written under the form

$$P_i = c_i(u_i).\,\mathsf{new}\ k_1 \cdots \mathsf{new}\ k_{r_i}.\overline{c_i}\langle v_i\rangle$$

with c_i a constant and $u_1, \ldots, u_n$ a family of patterns verifying the following properties:
1 *binding uniqueness*: for all i, u_i does not contain two different variables;
2 *pattern determinism*: for all $i \neq j$, if u_i and u_j are unifiable then $c_i \neq c_j$.

In [CCD15b] there is an additional, technical syntactic restriction on the structures of u_i and v_i that is specific to the fixed theory considered in [CCD15b] for randomised symmetric and asymmetric encryption and digital signature. Intuitively the terms u_i and v_i are defined

by grammars essentially imposing that the encryption randomness (resp. keys) they contain are indeed fresh nonces (resp. long-term keys), that is, they are names among $k_1, \ldots, k_{r_i}$ (resp. are of the form k or $\mathsf{pk}(k)$ for some name $k \notin \{k_1, \ldots, k_{r_i}\}$). We refer to [CCD15b] for details about this last assumption.

> ***Theorem 7.6*** *(trace equivalence for deterministic pin-pong protocols [CCD15b])*
> For a theory limited to randomised symmetric and asymmetric encryption as well as digital signature, TraceEquiv is PRIMREC for deterministic ping-pong protocols.

Decidability is obtained by a reduction of the problem to the language equivalence of deterministic pushdown automata, which is decidable in primitive recursive time. A complexity lower bound for this problem is open (beyond the P hardness trivially inherited from static equivalence, recall Theorem 7.3).

For simple processes We now study a decidability result for patterned simple processes [CCD15a]. In this work the theory is limited to symmetric encryption and pairs, and the processes must be *type compliant* and *acyclic*. These notions are rather technical and we only formalise them in Appendix E in the vocabulary of our model, but give an intuition of their meaning here. Type compliance relies on a type system to ensure that, whenever two (subterms of) terms u, v appearing in the process are unifiable then they can only be instantiated by terms of the same structure during a process execution. Then acyclicity is a property of the *dependency graph* of the process. The vertices of this graph are the instructions of the process. There is an edge $a \to a'$ when it may be necessary to execute a' before a to perform some attacker actions.

▶ *Example 7.1*
There are three kind of edges in a dependency graph. *Sequential dependency* is for actions following each other, for example in $\beta.\alpha.P$ there is an edge $\alpha \to \beta$. *Pattern* and *deduction dependencies* are for actions that allow the attacker to produce a term of a given pattern or deduce a subterm of an output message, respectively. For example in $\alpha.P \mid \beta.Q \mid \gamma.R$ with

$$\alpha = \overline{c}\langle \mathsf{senc}(u,k)\rangle \qquad \beta = d(\mathsf{senc}(x,k)) \qquad \gamma = \overline{e}\langle k\rangle$$

there is an edge $\beta \to \alpha$ because the term $\mathsf{senc}(u,k)$ could be used as an input term for the pattern $\mathsf{senc}(x,k)$. Also $\gamma \to \alpha$ because the term k output in γ can be used to deduce u from $\mathsf{senc}(u,k)$ in α. Similarly note that there is a cyclic dependency in

$$!^{ch}\,\beta.\alpha \qquad \text{with} \qquad \alpha = \overline{c}\langle \mathsf{senc}(u,k)\rangle \qquad \beta = c(\mathsf{senc}(x,k))\,.$$

We have $\alpha \to \beta$ by sequential dependency, but also $\beta \to \alpha$ by pattern dependency across the different copies of $\beta.\alpha$. ◀

There is also a restriction to *atomic keys*, i.e. for all encryptions $\mathsf{senc}(u,v)$ appearing in the process, $v \in \mathcal{F}_0 \cup \mathcal{N} \cup \mathcal{X}$. This restriction is also applied to attacker's recipes in the semantics by strenghtening the msg predicate (which therefore also impacts the definition of static equivalence).

Theorem 7.7 *(trace equivalence for type-compliant acyclic processes [CCD15a])*
For a theory limited to pairs and symmetric encryption, TraceEquiv is coNEXP for patterned, simple, type-compliant, acyclic processes with atomic keys.

Proof. Given a trace we consider its so-called *execution graph*: its vertices are the actions of the trace and its edges mirror those of the dependency graph of the process. It is proved in [CCD15a] that when two patterned, simple, type-compliant, acyclic processes P and Q are not trace equivalent, there exists an attack trace, say, in P, whose execution graph D has these properties:

1. D is acyclic and $\mathsf{depth}(D)$ (max length of a path of D) is polynomial in the size of P.
2. $\mathsf{width}(D)$ (maximal number of outgoing edges from a vertex of D) is exponential in the size of P and of the type system.
3. $\mathsf{nbroots}(D)$ (number of vertices of D that have no ingoing edges) is exponential in the size of P and of the type system.

From each root of D, the number of reachable vertices is at most the size of a tree of width $\mathsf{width}(D)$ and of depth $\mathsf{depth}(D)$, i.e. $\mathsf{width}(D)^{\mathsf{depth}(D)+1}-1$. Hence the number of vertices of D is bounded by $\mathsf{nbroots}(D)\cdot\mathsf{width}(D)^{\mathsf{depth}(D)+1}$ which is exponential in the size of P. Since the number of vertices of D is an upper bound on the number of sessions needed to execute the underlying trace, it suffices to prove the equivalence of P and Q for an exponential number number of sessions. This leads to an overall coNEXP procedure since trace equivalence of bounded, positive, simple processes is coNP for subterm theories (see the combination of Theorem 7.17 and Proposition 7.19 later in Section 4). □

Note that complexity was not the focus of [CCD15a] and the authors only claimed a triple exponential complexity for their procedure, hence the contribution of the above complexity analysis. Moreover no lower bounds were investigated but we prove that the problem is actually coNEXP complete.

Theorem 7.8 *(hardness of trace equivalence for type-compliant acyclic processes)*
For the theory of pairs and symmetric encryption, TraceEquiv is coNEXP hard for patterned, simple, type-compliant, acyclic processes with atomic keys.

The reduction shares some similarities with the proof of coNEXP hardness for trace equivalence of bounded processes (see Theorem 6.19), compensating the more deterministic structure of simple processes by the use of replication. We give below an intuition of our construction, detailed in Appendix E.

Proof sketch. We proceed by reduction from SuccinctSAT. Let φ be a formula with 2^m clauses and 2^n variables $x_0,\dots,x_{2^n-1}$ and Γ be a circuit encoding this formula. We construct two simple, type-compliant, acyclic processes that are trace equivalent *iff* φ is unsatisfiable. Using pairs $\langle u,v\rangle$ we encode binary trees: a leaf is a non-pair value and, if u and v encode binary trees, $\langle u,v\rangle$ encodes the tree whose root has u and v as children. Given a term t, we build a process $P(t)$ behaving as follows:

1. $P(t)$ first waits for an input x from the attacker. This term x is expected to be a binary tree of depth n with boolean leaves, modelling a valuation of φ (the i^{th} leaf of x being

the valuation of x_i).

2 The goal is to make $P(t)$ verify that this valuation satisfies φ; if the verification succeeds the process outputs t. Given two constants $\mathsf{0}$ and $\mathsf{1}$, $P(\mathsf{0})$ and $P(\mathsf{1})$ will thus be trace equivalent *iff* φ is unsatisfiable.

3 However it is not possible to inline, within a process of polynomial size, the verification that the valuation encoded by x satisfies the 2^m clauses of φ. Hence we replicate a process that, given x, verifies one clause at a time. Intuitively, the attacker will guide the verification of the 2^m clauses of φ and, when the i^{th} clause is successfully verified, the process reveals the binary representation of i (encrypted using a key unknown to the attacker).

4 In particular, the attacker obtains the encryption of all integers $i \in [\![0, 2^m - 1]\!]$ only after successfully verifying that the initial input x effectively encodes a valuation satisfying all clauses of φ. It then suffices to design a process that outputs t if the attacker is able to provide all such ciphertexts. This can be encoded by a replicated process that, upon receiving the encryption of two integers that differ only by their least significant bit, reveals the encryption of these integers with the least significant bit truncated. The verification ends when revealing the encryption of the empty binary representation. □

3 Complexity of constraint solving

Section summary

The decision of equivalence properties has often been reduced to forms of constraint solving as in Chapter 3. We formalise and study the complexity of such problems which will be at the core of our last complexity results for equivalence properties.

3.1 Constraint solving

Although the models we survey differ in their technical definition, their proof techniques often share a similar core. In the bounded fragment for example it is common to abstract the infinitely-branching transition relation by symbolic constraints as in Chapter 3 or [Bau07, CCD13]. We recall that in our model, a constraint system is a triple $\mathcal{C} = (\Phi, \mathsf{D}, \mathsf{E}^1)$ where Φ is a frame, D is a conjunction of constraints indicating which terms should be deducible or not, and E^1 expresses the relations between the protocol messages.

This approach reduces reachability or equivalence properties to the study of equivalences to various flavours of constraint-solving problems; in our decision procedure for trace equivalence and labelled bisimilarity for example, the role of the constraint solver was to refine a set of constraint systems to group those with statically-equivalent solutions. We did not study the precise complexity of this problem, but we survey in the rest of this section some decidability and complexity results for simpler constraint-solving problems (satisfiability and equivalence).

Decision problem — CSysSAT

Input: a theory E, a constraint system $\mathcal{C}$

Question: Is $Sol(\mathcal{C})$ empty w.r.t. E?

More generally, the Weak Secrecy problem can be decided in non-deterministic polynomial time with oracle to CSysSAT, intuitively as follows:

1 guess non-deterministically one symbolic execution of P (which is of polynomial size) and collect the corresponding constraints into a constraint system $\mathcal{C} = (\Phi, \mathsf{D}, \mathsf{E}^1)$

2 answer yes $Sol(\Phi, \mathsf{D} \wedge X \vdash^? x, \mathsf{E}^1 \wedge x =^? t) \neq \varnothing$, with $X{:}n, x$ fresh and $n = |dom(\Phi)|$.

Regarding equivalence properties, as recalled above, the situation is more complex if we want to handle trace equivalence or labelled bisimilarity of arbitrary processes as in Chapter 4. However a simpler variant has been considered for deciding equivalence properties in more restrictive settings [Bau07, CCD13]:

Decision problem — CSysEquiv

Input: A theory, two constraint systems $(\Phi_i, \mathsf{D}_i, \mathsf{E}_i^1)$, $i \in \{1, 2\}$, with $dom(\Phi_1) = dom(\Phi_2)$

Question: Do the constraint systems $(\Phi_i, \mathsf{D}_i \wedge \varphi, \mathsf{E}_i^1 \wedge x =^? y)$ have the same set of solutions, where

$$\varphi = X \vdash^? x \wedge Y \vdash^? y \qquad \text{with } X{:}n, Y{:}n, x, y \text{ fresh and } n = |dom(\Phi_1)|$$

This problem is called *S-equivalence* in [Bau07]. Note that this can be seen as a generalisation of the StatEquiv problem which can be retrieved with $\mathsf{D}_1 = \mathsf{D}_2 = \mathsf{E}_1^1 = \mathsf{E}_2^1 = \top$.

3.2 Complexity

We now present some decidability and complexity results for CSysSAT and CSysEquiv; they will be at the core of the results presented in the next sections. These two problems have been studied in majority in [Bau07] for the decidability of reachability properties and diff equivalence, in the case of *fixed* subterm theories in the positive fragment.

Proposition 7.9 (constraint solving for fixed subterm convergent theories [Bau07])

For all fixed subterm convergent theories, CSysSAT (resp. CSysEquiv) is NP (resp. coNP for positive constraint systems).

As far as we know the complexity of this problem has only been studied for fixed theories. However the result of [Bau07] above can be adapted to parametric theories; inspecting the proof we observe that 1 in the complexity bounds, the dependencies in the theory are polynomial and 2 the proof uses the fact that static equivalence is P for fixed theories (Theorem 7.1) but the arguments still hold if we only assume static equivalence to be coNP. Since it has also been proved in [Bau07] that CSysSAT was NP hard if the theory includes at least a free binary function symbol, we obtain the more general complexity result:

Theorem 7.10 *(constraint solving for subterm convergent theories)*

CSysSAT (resp. CSysEquiv) is NP complete (resp. coNP complete) for subterm convergent theories and positive constraint systems. Moreover CSysSAT is also NP complete without the positivity assumption.

Regarding the complexity lower bounds for fixed theories, similarly to the problems we surveyed in the previous sections, the complexity may vary from one theory to the other:

Theorem 7.11 *(constraint solving in the pure fragment)*
With the empty theory, CSysSAT and CSysEquiv are LOGSPACE.

Proof. It suffices to prove that CSysEquiv is LOGSPACE. We let two constraint systems $\mathcal{C}_1$ and $\mathcal{C}_2$ such that $dom(\Phi_1) = dom(\Phi_2) = \{\mathsf{ax}_1, \ldots, \mathsf{ax}_p\}$ and $vars^2(\mathcal{C}_1) = vars^2(\mathcal{C}_2) = \{X_1, \ldots, X_n\}$. In the pure fragment there are finitely-many second-order substitutions Σ for $\mathcal{C}_1$ and $\mathcal{C}_2$ up to bijective renaming of fresh constants (which does not affect whether Σ is a solution of $\mathcal{C}_1$ or $\mathcal{C}_2$). Indeed for all $i \in [\![1, n]\!]$, the recipe $X_i\Sigma$ is either

1. a constant appearing either in Φ_1, Φ_2, in an equation of S_1 or S_2 or in some $X_j\Sigma$, $j < i$
2. a fresh constant (i.e., a constant not captured by the previous case)
3. an axiom ax_j such that $j < arity(X_i)$.

Given a second-order substitution Σ, we can verify that it is a solution of $\mathcal{C}_1$ and $\mathcal{C}_2$ in LOGSPACE since the constraint systems only contain equations and disequations between constants, names and variables. The problem can thus be solved in LOGSPACE by bruteforce, using three nested loops:

1. the first two loops are of size in n and p and are used to enumerate all second-order substitutions Σ up to bijective renaming of fresh constants
2. the third loop of size polynomially-bounded by $|\mathcal{C}_1| + |\mathcal{C}_2|$ verifying that Σ is a solution of $\mathcal{C}_1$ *iff* it is a solution of $\mathcal{C}_2$. □

Since CSysEquiv is a generalisation of StatEquiv it can also be interesting to compare their complexity. Regarding fixed theories, StatEquiv is P (Theorem 7.1) and this is optimal in the sense that the problem is P hard for all theories containing symmetric encryption (Theorem 7.3). The coNP bound is also optimal for CSysEquiv in the same sense:

Theorem 7.12 *(hardness of constraint solving for fixed theories)*
For all theories E containing symmetric encryption, CSysSAT (resp. CSysEquiv) is NP hard (resp. coNP hard) for positive constraint systems.

Proof. It suffices to prove that CSysSAT is NP hard. By reduction from SAT we let $\varphi = \bigwedge_{i=1}^{p} C_i$ a SAT formula with variables $x_1, \ldots, x_n$. Given a family of distinct names $k_1, \ldots, k_n$, we first consider the following frame with n free variables

$$\Phi_{val} = \{\mathsf{ax}_1 \mapsto \mathsf{senc}(x_1, k_1), \ldots, \mathsf{ax}_n \mapsto \mathsf{senc}(x_n, k_n)\}.$$

Given a clause C of φ, we let $x_{i_1}, x_{i_2}, x_{i_3}$ its variables, $b_{i_1}, b_{i_2}, b_{i_3}$ its negation bits, and a fresh name k_c. We define a frame Φ_c such that, for all valuations σ of $x_1, \ldots, x_n$, the name k_c is deducible from $\Phi_{val}\sigma \cup \Phi_c$ *iff* σ satisfies C (i.e. *iff* there exists $j \in [\![1, 3]\!]$ such that $x_{i_j}\sigma = b_{i_j}$):

$$\Phi_c = \left\{ \begin{array}{l} \mathsf{ax}_1^c \mapsto \mathsf{senc}(k_c, \mathsf{senc}(b_{i_1}, k_{i_1})) \\ \mathsf{ax}_2^c \mapsto \mathsf{senc}(k_c, \mathsf{senc}(b_{i_2}, k_{i_2})) \\ \mathsf{ax}_3^c \mapsto \mathsf{senc}(k_c, \mathsf{senc}(b_{i_3}, k_{i_3})) \end{array} \right\}$$

All in all the following constraint system $(\Phi, \mathsf{D}, \mathsf{E}^1)$ is satisfiable *iff* φ is satisfiable, with:

$$\Phi = \Phi_{val} \cup \Phi_{c_1} \cup \cdots \cup \Phi_{c_p} \qquad \mathsf{D} = \bigwedge_{i=1}^{n} X_i \vdash^? x_1 \wedge \bigwedge_{j=1}^{p} Y_i \vdash^? y_1 \qquad \mathsf{E}^1 = \bigwedge_{i=1}^{p} y_j =^? k_{c_j}$$

with $X_1{:}0, \ldots, X_1{:}0, Y_1{:}n, \ldots, Y_p{:}n, y_1, \ldots, y_p$ fresh. □

The complexity of the general problem (that is, with disequations) is open. However it is easily seen less general than trace equivalence and thus inherits its complexity upper bounds.

Proposition 7.13 (reduction of constraint solving to trace equivalence)
CSysEquiv is logspace-reducible to TraceEquiv of bounded processes with the same theory.

Proof. Consider a constraint system $\mathcal{C} = (\Phi, \mathsf{D}, \mathsf{E}^1)$ with $\Phi = \{\mathsf{ax}_1 \mapsto t_1, \ldots, \mathsf{ax}_n \mapsto t_n\}$ and

$$\mathsf{D} = \{X_1 \vdash^? x_1, \ldots, X_p \vdash^? x_p\} \qquad \mathsf{E}^1 = \{u_1 \sim_1 v_1, \ldots, u_q \sim_q v_q\}$$

where for all $i \in [\![1, q]\!]$, $\sim_i \in \{=^?, \neq^?_E\}$. Assuming that the second-order variables X_i are sorted by increasing arity, we let $1 = i_0 \leqslant i_1 \leqslant \cdots \leqslant i_n \leqslant i_{n+1} = p + 1$ the sequence of integers such that $arity(X_i) = \ell$ *iff* $i_\ell \leqslant i < i_{\ell+1}$. We then let c be a constant and define the following process given another process R:

$$\begin{aligned} P(\mathcal{C}, R) = {} & c(x_{i_0}).\cdots c(x_{i_1-1}). \\ & \overline{c}\langle t_1\rangle.c(x_{i_1}).\cdots c(x_{i_2-1}). \\ & \quad \vdots \\ & \overline{c}\langle t_n\rangle.c(x_{i_n}).\cdots c(x_{i_{n+1}-1}). \\ & [u_1 \sim_1 v_1] \cdots [u_q \sim_q v_q]\, R \end{aligned}$$

where $[u \sim v]P$ is a shortcut for either "if $u = v$ then P else 0" (when $\sim$ is $=^?$) or "if $u = v$ then 0 else P" (when $\sim$ is $\neq^?_E$). The process $P(\mathcal{C}, R)$ is well-defined (i.e. does not contain variables that are not bound by a prior input) if $\mathcal{C}$ verifies the origination property. In $P(\mathcal{C}, R)$, the subprocess R can be executed *iff* $x_1, \ldots, x_n$ are instanciated by recipes that define a solution of $\mathcal{C}$. In particular given a constant d and two constraint systems $\mathcal{C}_0, \mathcal{C}_1$ verifying the hypotheses of the problem CSysEquiv, $\mathcal{C}_0$ and $\mathcal{C}_1$ are equivalent *iff* for all traces t of $P(\mathcal{C}_i, \overline{d}\langle d\rangle)$ containing an output on d, $i \in \{0, 1\}$, there exists a trace t' of $P(\mathcal{C}_{1-i}, \overline{d}\langle d\rangle)$ such that $t \sim t'$. In particular $\mathcal{C}_0$ and $\mathcal{C}_1$ are equivalent *iff*

$$P(\mathcal{C}_0, \overline{d}\langle d\rangle) \oplus P(\mathcal{C}_1, 0) \qquad \text{and} \qquad P(\mathcal{C}_0, 0) \oplus P(\mathcal{C}_1, \overline{d}\langle d\rangle)$$

are trace equivalent where, for $k, k' \in \mathcal{N}$ and $e \in \mathcal{F}_0$ fresh:

$$A \oplus B \quad = \quad \overline{e}\langle k\rangle \mid \overline{e}\langle k'\rangle \mid e(x).\,(\,[x = k]\,A \mid [x = k']\,B\,)$$ □

Theorem 7.14 *(constraint equivalence for subterm convergent theories)*
CSysEquiv is decidable in coNEXP for subterm convergent constructor-destructor theories.

4 Diff equivalence and determinacy

Section summary

In this section we survey complexity results that can be derived from the results about constraint solving detailed in the previous section. The new results are mostly applicable to *diff equivalence* and equivalence of *determinate processes*.

4.1 Diff equivalence

Although undecidable in general, diff equivalence can be decided in the bounded positive fragment by using a reduction to CSysEquiv [Bau07]:

Proposition 7.15 (reduction of diff equivalence to constraint solving [Bau07])

In the bounded (resp. bounded positive) fragment, given a non-deterministic algorithm A for non-CSysEquiv (resp. for non-CSysEquiv of positive constraint systems), non-DiffEquiv is NP, where a call to A is seen as an elementary instruction.

Proof sketch. The decision procedure of [Bau07] for non equivalence consists of 1 guessing a symbolic trace t, 2 consider the unique (if it exists) candidate equivalent trace t' in the other process, and 3 conclude that the processes are not diff-equivalent if the constraint systems corresponding to t and t' are not equivalent. In the case of the positive fragment, an additional argument is required to prove that it is not necessary to consider symbolic traces that produce disequation constraints. □

In particular when composing this with the different complexity results for CSysEquiv mentioned in Section 3.2:

Theorem 7.16 *(diff equivalence of bounded processes)*

DiffEquiv is 1 coNEXP for bounded processes and constructor-destructor subterm convergent theories, 2 coNP for bounded positive processes and subterm convergent theories.

The problem is also known coNP hard even in the positive fragment for a theory containing only a free binary symbol h [Bau07]. However a simple proof justifies that DiffEquiv is actually coNP hard even for the empty theory and, hence, for any fixed theory:

Theorem 7.17 *(diff equivalence in the pure fragment)*

In the pure pi-calculus, DiffEquiv is coNP complete for positive bounded processes.

Proof. By reduction from SAT let a formula $\varphi = \bigwedge_{i=1}^{m} C_i$ in CNF and $\vec{x} = x_1, \ldots, x_n$ its variables. For each clause C_i, let k_i be a fresh name and define

$$CheckSat_i(\vec{x}) = [x_{i_1} = b_{i_1}]\overline{c}\langle k_i\rangle \mid \cdots \mid [x_{i_p} = b_{i_p}]\overline{c}\langle k_i\rangle$$

where $x_{i_1}, \ldots, x_{i_p}$ are the variables of C_i and $b_{i_1}, \ldots, b_{i_p}$ their negation bits. That is, at least

one output of k_i is reachable in $\mathit{CheckSat}_i(\vec{x})$ if $\vec{x}$ is a valuation of φ that satisfies C_i. If

$$\mathit{CheckSat} = c(x_1).\dots.c(x_n).(\mathit{CheckSat}_1(\vec{x}) \mid \dots \mid \mathit{CheckSat}_m(\vec{x}))$$
$$\mathit{Final}(t) = c(y_1).[y_1 = k_1]\dots c(y_m).[y_m = k_m]\ \overline{c}\langle t\rangle$$

then for $0, 1 \in \mathcal{F}_0$, $\mathit{CheckSat} \mid \mathit{Final}(0) \approx_d \mathit{CheckSat} \mid \mathit{Final}(1)$ *iff* φ is unsatisfiable. □

In particular this gives the exact complexity of DiffEquiv in the bounded positive fragment. As far as we know the question remains open without the positivity assumption.

Corollary 7.18 (diff equivalence of bounded processes, fixed and parametric theories)

DiffEquiv is coNP complete for subterm convergent theories (fixed or not) and bounded positive processes.

4.2 The case of determinacy

Definition(s) We now focus on *determinate* processes, already defined in Chapter 5. This class has been investigated significantly in the literature [BDH15, CCCK16, CCD13] but several variants coexist, as discussed in [BCK20]. For example we recall that the partial-order reductions for trace equivalence [BDH15] mentioned in Chapters 3 and 5 are valid for *action-determinate* processes; such processes shall never reach an intermediary state where two inputs (resp. outputs) on the same communication channel are executable in parallel. On the other hand a more permissive definition is used in [CCD13] (not detailed in this survey). There also exists a notion that is stricter than all of these, referred as *strong determinacy* [BCK20].

Definition 7.4 (strongly determinate process)

A process is *strongly determinate* when it verifies all of the following properties:

1. it does not contain private channels,
2. it is bounded,
3. all its syntactic subprocesses are strongly determinate,
4. in case the process is of the form $P \mid Q$ there exist no channels c such that both P and Q contain an input (resp. output) on c.

For example this process is action-determinate but not strongly-determinate:

$$\mathsf{if}\ a = b\ \mathsf{then}\ c(x)\ \mathsf{else}\ 0 \quad \mid \quad \mathsf{if}\ a = b\ \mathsf{then}\ 0\ \mathsf{else}\ c(x)\,.$$

Effects on equivalences The following proposition summarises the relation between the different notions we presented so far and their implications in terms of decidability.

Proposition 7.19 (main properties of determinate and simple processes [CCD13, BCK20])

1. $\approx_t$, $\approx_s$ and $\approx_l$ coincide for action-determinate processes.
2. DiffEquiv, TraceEquiv, SessEquiv and Bisimilarity are logspace-reducible to each others for simple processes.
3. Strongly-determinate processes are action determinate, simple processes are action determinate, and bounded simple processes are strongly determinate.

In particular Item 2 follows from the fact that, up to a reordering of parallel operators at toplevel, equivalence by session and diff-equivalence are the same decision problem for simple processes. Regarding complexity, it is shown in [CCD13] that, for bounded simple positive processes, the equivalence problem can be reduced to CSysEquiv similarly to Theorem 7.15 for diff-equivalence. Their arguments can be generalised from simple to strongly-determinate processes in a straightforward manner; however it is not clear whether this would also be true for action-determinate processes in general or for processes with else branches. We thus obtain the same complexity as diff-equivalence for this fragment:

Theorem 7.20 *(reduction of equivalences to constraint solving [CCD13])*

TraceEquiv, Bisimilarity and SessEquiv are coNP-complete for subterm convergent theories and positive strongly-determinate processes. The coNP completeness also holds for all fixed subterm convergent theories.

5 Summary of the results

Section summary

We summarise in Table 7.1 all the complexity results that were proved, surveyed or streghtened in this chapter. We highlight in particular some remaining *open problems* and the impact of choosing a *fixed rewrite system* in the complexity analysis.

Figure 7.1 summarises the main results of this chapter and highlights remaining open questions. Cells for which the complexity results are not tight have coloured background. For instance, for subterm-convergent constructor-destructor theories and bounded processes, DiffEquiv is known coNEXP and coNP hard, but the precise complexity remains unknown. Consistently with the results of the paper we also include some complexity results with the theory seen as a constant of the problem (denoted as "fixed" in the *theory* columns). The corresponding cells contain bounds applying to *all* theories of the class; e.g. for Bisimilarity of bounded processes, with fixed subterm-convergent constructor-destructor theories, the problem is decidable in coNEXP and PSPACE hard. Despite the gap between the two bounds, they are optimal since there exist theories for which the problem is PSPACE complete and others for which it is coNEXP complete. Therefore this cell is not highlighted. In our opinion the most interesting open questions are:

1. Can upper bounds on constructor-destructor theories be lifted to more general subterm convergent theories?
2. Without the positivity assumption, can we tighten the complexity for diff equivalence, and strongly determinate processes?

This last question might allow to better understand why strongly determinate processes benefit from optimisations that improve verification performance that much. Finally, as witnessed by the contrast between the high complexity of equivalence by session and its practical efficiency, worst-case complexity may not always be an adequate measure.

Table 7.1 Summary of the results. Coloured cells indicate configurations with open problems. Naturally, in the case of StatEquiv and CSysEquiv, the non-applicable hypotheses on processes (e.g. boundedness) should be ignored when reading the table.
All results of this table for diff-equivalence also apply to trace equivalence / labelled bisimilarity / equivalence by session of strongly-determinate processes.

theory			*process*		StatEquiv	CSysEquiv	DiffEquiv	Bisimilarity	SessEquiv	TraceEquiv
subterm convergent	any	bounded	any		coNP complete	coNP hard		coNEXP hard		
			pos.							
	any (fixed)	bounded	any		P	coNP hard		PSPACE hard	Π_2 hard	
			pos.			coNP complete				
constructor-destructor	any	bounded	any		coNP complete	coNEXP coNP hard		coNEXP complete		
			pos.							
	any (fixed)	bounded	any		P	coNEXP coNP hard		coNEXP PSPACE hard	coNEXP Π_2 hard	
			pos.			coNP				
	sign, rsenc, raenc	unbounded	patterned, ping-pong, deterministic		P complete	coNP complete	P hard		PRIMREC P hard	
	senc, $\langle\,\rangle$	unbounded	patterned, type compliant, acyclic, simple, atomic keys				coNEXP complete			
		bounded	any			coNEXP coNP hard				
			pos.						coNEXP Π_2 hard	
	empty	bounded	any		LOGSPACE		coNP complete	PSPACE complete	Π_2 complete	
			pos.							

General conclusion

We now conclude this thesis by a summary of our approach and results and consider some potential future work.

In this thesis we have contributed to the theory and practice of the automated verification of security protocols in symbolic models, with a focus on privacy-type properties modelled by behavioural equivalences. Our main contributions are:

1. a decision procedure for trace equivalence and labelled bisimilarity of bounded processes for constructor-destructor subterm convergent theories, implemented as the DeepSec prover that compares favorably to the similar state-of-the-art tools in terms of expressivity and performances;
2. a tight complexity analysis of the problem, proving it to be coNEXP complete, and several other complexity results integrated in a survey to give a clearer view of the theoretical limits and open problems;
3. a refinement of trace equivalence (equivalence by session) used as a powerful optimisation technique permitting to make analyses scale that were out of the scope of many state-of-the-art automated provers in terms of expressivity of resources;
4. the application of all of our verification techniques to carefully modelled examples, illustrating the capabilities of DeepSec on real cases.

We now identify some possible future work related to our contributions.

Short term There are some natural extensions of our results to be investigated. First, all aspects of the completeness of equivalence by session, that is, how to handle false attacks when they arise: when an analysis of equivalence by session concludes that two processes are not equivalent, but may be trace equivalent, it would be interesting to build up on this preliminary analysis to prove trace equivalence so that the proof does not have to be done from scratch. One would typically exploit the branches of the partition tree that have already been treated successfully, and only focus on failed branches.

Some other theoretically less demanding future work can be mentioned, such as implementing a native procedure for reachability in DeepSec (which would be a simplification of the current procedure for equivalence, using a lighter version of the partition tree) to verify natively a larger class of properties in the tool. It is proved in [BDH15] that the partial order reductions developed for proving the equivalence of determinate processes can be applied to any process when proving reachability properties, suggesting a rather efficient procedure without the need to rely on techniques analogue to equivalence by session.

We can also investigate implementing a native procedure for verifying trace inclusion: currently when we want to verify $P \sqsubseteq_t Q$ we prove equivalently that $P + Q \approx_t Q$ which however involves bigger processes. A preliminary experimental implementation suggests that the verification time can be cut by half by using a native procedure, but a theoretical and engineering effort would be needed to design a clean procedure (in particular to adapt the partial order reductions integrated to DeepSec for determinate processes to the decision of trace inclusion).

Although this may require more work due to the more complex, rather impractical, theoretical decision procedure we provided in this thesis, it may also be useful to implement a decision procedure for labelled bisimilarity of bounded processes in DeepSec in addition to the current implementation that only supports trace equivalence.

Longer term There are also several tracks to be investigated in the long run. The straightforward ones are the generalisation of the algorithm of DeepSec to larger theories that are not subterm, or remove the constructor-destructor restriction. This would require to rethink the whole data structure behind the algorithm (for example the finite knowledge base used to model the attacker knowledge heavily relies on the assumptions on the theory). This extension comes with its own set of theoretical challenges as far as complexity is concerned, as there are many open issues with its regard (typically the complexity of equivalences when removing the constructor-destructor assumption, or in some classes with else branches). Another central complexity question is whether the worst-case, asymptotic complexity is the right tool in our context: our results show for example that equivalence by session and trace equivalence have the same theoretical complexity in the context of DeepSec's framework despite the significant difference in practice performance. A finer analysis taking more parameters into account—for example relying on the notion of *parametrised complexity*—may be a better fit to explain the improve the understanding of the computational difference between the two equivalences.

Another interesting track would be to see to which extent is DeepSec generalisable to different types of properties. In this thesis we have defined and used the partition tree to decide equivalence properties. We also mentioned earlier in the conclusion that the same approach could be used to decide reachability properties. One may wonder whether the notion of partition tree (or a generalisation) could actually be used at the core of a decision procedure for more general relational *hyperproperties*. Taking inspiration from the decision procedure provided in this thesis, we could investigate whether some fragments of the HyperLTL logic could be decided using a similar approach for example.

Finally, another interesting result, directly tied to our experiments, is the investigation of reduction results for concrete protocols. Since the scope DeepSec is limited to the bounded fragment, its only way to provide complete security guarantees are theoretical results of the form "if $!P \not\approx_t !Q$ then $!^n P \not\approx_t !^n Q$" (where n is explicit). For example we mentioned in Chapter 2 such results for the Helios voting protocol: when ballot privacy is modelled using vote swap, it is known that some parameters of the problem can be bounded without loss of generality (number of voters, number of ballots accepted by the ballot box). Bounding the other parameters (number of ballots emitted by each voter) and investigating whether similar arguments could apply to the BPRIV definition would be interesting tracks to improve the guarantees offered by DeepSec for this protocol.

Bibliography

[ABF17] Martín Abadi, Bruno Blanchet, and Cédric Fournet. The applied pi calculus: Mobile values, new names, and secure communication. *Journal of the ACM (JACM)*, 2017.

[AC06] Martín Abadi and Véronique Cortier. Deciding knowledge in security protocols under equational theories. *Theoretical Computer Science*, 2006.

[ACK16] Myrto Arapinis, Véronique Cortier, and Steve Kremer. When are three voters enough for privacy properties? In *European Symposium on Research in Computer Security (ESORICS)*, 2016.

[ACRR10] Myrto Arapinis, Tom Chothia, Eike Ritter, and Mark Ryan. Analysing unlinkability and anonymity using the applied pi calculus. In *IEEE Computer Security Foundations Symposium (CSF)*, 2010.

[Adi08] Ben Adida. Helios: Web-based open-audit voting. In *USENIX security symposium*, 2008.

[AF04] Martín Abadi and Cédric Fournet. Private authentication. *Theoretcal Computer Science*, 2004.

[AG99] Martın Abadi and Andrew D Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and computation*, 1999.

[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 *Conference on Computer and Communications Security (CCS)*, 2012.

[ANR07] Siva Anantharaman, Paliath Narendran, and Michael Rusinowitch. Intruders with caps. In *International Conference on Rewriting Techniques and Applications*, 2007.

[BAF08] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated verification of selected equivalences for security protocols. *The Journal of Logic and Algebraic Programming*, 2008.

[Bau07] Mathieu Baudet. *Sécurité des protocoles cryptographiques: aspects logiques et calculatoires*. PhD thesis, École normale supérieure de Cachan, 2007.

[BBK17] Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. Verified models and reference implementations for the TLS 1.3 standard candidate. In *IEEE Symposium on Security and Privacy, (S&P)*, 2017.

[BC14] David A. Basin and Cas Cremers. Know your enemy: Compromising adversaries in protocol analysis. *ACM Transactions on Information and System Security (TISSEC)*, 2014.

[BCD13] Mathieu Baudet, Véronique Cortier, and Stéphanie Delaune. YAPA: A generic tool for computing intruder knowledge. *ACM Trans. Comput. Log.*, 2013.

[BCEDH12] Mayla Brusó, Konstantinos Chatzikokolakis, Sandro Etalle, and Jerry Den Hartog. Linking unlinkability. In *International Symposium on Trustworthy Global Computing*, 2012.

[BCG+15] David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. Sok: A comprehensive analysis of game-based ballot privacy definitions. In *IEEE Symposium on Security and Privacy (S&P)*, 2015.

[BCK20] Kushal Babel, Vincent Cheval, and Steve Kremer. On the semantics of communications when verifying equivalence properties. *Journal of Computer Security*, 2020.

[BDH14] David Baelde, Stéphanie Delaune, and Lucca Hirschi. A reduced semantics for deciding trace equivalence using constraint systems. In *International Conference on Principles of Security and Trust (POST)*, 2014.

[BDH15] David Baelde, Stéphanie Delaune, and Lucca Hirschi. Partial order reduction for security protocols. In *International Conference on Concurrency Theory (CONCUR)*, 2015.

[BDH18a] David Baelde, Stéphanie Delaune, and Lucca Hirschi. POR for security protocol equivalences - beyond action determinism. In *European Symposium on Research in Computer Security (ESORICS)*, 2018.

[BDH+18b] David A. Basin, Jannik Dreier, Lucca Hirschi, Sasa Radomirovic, Ralf Sasse, and Vincent Stettler. A formal analysis of 5g authentication. In *ACM Conference on Computer and Communications Security (CCS)*, 2018.

[BDNP01] Michele Boreale, Rocco De Nicola, and Rosario Pugliese. Proof techniques for cryptographic processes. *SIAM Journal on Computing*, 2001.

[Ben87] Josh C Benaloh. *Verifiable secret-ballot elections*. PhD thesis, Yale University, 1987.

[Bla04] Bruno Blanchet. Automatic proof of strong secrecy for security protocols. In *IEEE Symposium on Security and Privacy (S&P)*, 2004.

[Bla16] Bruno Blanchet. Modeling and verifying security protocols with the applied pi calculus and proverif. *Foundations and Trends in Privacy and Security*, 2016.

[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 *IEEE Symposium on Security and Privacy (S&P)*, 2008.

[BS18] Bruno Blanchet and Ben Smyth. Automated reasoning for equivalences in the applied pi calculus with barriers. *Journal of Computer Security*, 2018.

[BSCS20] Bruno Blanchet, Ben Smyth, Vincent Cheval, and Marc Sylvestre. *Automatic Cryptographic Protocol Verifier, User Manual and Tutorial*, 2020. available at https://prosecco.gforge.inria.fr/personal/bblanche/proverif/manual.pdf.

[BY86] Josh C Benaloh and Moti Yung. Distributing the power of a government to enhance the privacy of voters. In *ACM symposium on Principles of distributed computing (PODC)*, 1986.

[CB13] Vincent Cheval and Bruno Blanchet. Proving more observational equivalences with proverif. In *International Conference on Principles of Security and Trust (POST)*, 2013.

[CBC11] Bruno Conchinha, David A Basin, and Carlos Caleiro. Fast: an efficient decision procedure for deduction and static equivalence. In *International Conference on Rewriting Techniques and Applications (RTA)*, 2011.

[CC05] Hubert Comon and Véronique Cortier. Tree automata with one memory set constraints and cryptographic protocols. *Theoretical Computer Science*, 2005.

[CCCK16] Rohit Chadha, Vincent Cheval, Ştefan Ciobâcă, and Steve Kremer. Automated verification of equivalence properties of cryptographic protocols. *ACM Transactions on Computational Logic (TOCL)*, 2016.

[CCD13] Vincent Cheval, Véronique Cortier, and Stéphanie Delaune. Deciding equivalence-based properties using constraint solving. *Theoretical Computer Science*, 2013.

[CCD15a] Rémy Chrétien, Véronique Cortier, and Stéphanie Delaune. Decidability of trace equivalence for protocols with nonces. In *IEEE Computer Security Foundations Symposium (CSF)*, 2015.

[CCD15b] Rémy Chrétien, Véronique Cortier, and Stéphanie Delaune. From security protocols to pushdown automata. *ACM Transactions on Computational Logic (TOCL)*, 2015.

[CCLD11] Vincent Cheval, Hubert Comon-Lundh, and Stéphanie Delaune. Trace equivalence decision: Negative tests and non-determinism. In *ACM conference on Computer and communications security (CCS)*, 2011.

[CD07] Véronique Cortier and Stéphanie Delaune. Deciding knowledge in security protocols for monoidal equational theories. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning*, 2007.

[CD09] Véronique Cortier and Stéphanie Delaune. A method for proving observational equivalence. In *IEEE Computer Security Foundations Symposium (CSF)*, 2009.

[CD12] Véronique Cortier and Stéphanie Delaune. Decidability and combination results for two notions of knowledge in security protocols. *Journal of Automated Reasoning*, 2012.

[CDD17] Véronique Cortier, Antoine Dallon, and Stéphanie Delaune. Sat-equiv: an efficient tool for equivalence properties. In *IEEE Computer Security Foundations Symposium (CSF)*, 2017.

[CDD18] Véronique Cortier, Antoine Dallon, and Stéphanie Delaune. Efficiently deciding equivalence for standard primitives and phases. In *European Symposium on Research in Computer Security (ESORICS)*, 2018.

[CDK12] Ştefan Ciobâcă, Stéphanie Delaune, and Steve Kremer. Computing knowledge in security protocols under convergent equational theories. *Journal of Automated Reasoning*, 2012.

[CDSV04] Ivan Cibrario, Luca Durante, Riccardo Sisto, and Adriano Valenzano. Exploiting symmetries for testing equivalence in the spi calculus. In *International Symposium on Automated Technology for Verification and Analysis (ATVA)*, 2004.

[CGCG+18] Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican, and Kevin Milner. On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. In *ACM Conference on Computer and Communications Security (CCS)*, 2018.

[CGG19] Véronique Cortier, Pierrick Gaudry, and Stéphane Glondu. Belenios: a simple private and verifiable electronic voting system. In *Foundations of Security, Protocols, and Equational Reasoning*, 2019.

[CGLM17] Véronique Cortier, Niklas Grimm, Joseph Lallemand, and Matteo Maffei. A type system for privacy properties. In *ACM Conference on Computer and Communications Security (CCS)*, 2017.

[CGLM18] Véronique Cortier, Niklas Grimm, Joseph Lallemand, and Matteo Maffei. Equivalence properties by typing in cryptographic branching protocols. In *International Conference on Principles of Security and Trust (POST)*, 2018.

[Che14] Vincent Cheval. APTE: an algorithm for proving trace equivalence. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2014.

[CHH+17] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. A comprehensive symbolic analysis of TLS 1.3. In *ACM Conference on Computer and Communications Security (CCS)*, 2017.

[CJM03] Edmund Clarke, Somesh Jha, and Will Marrero. Efficient verification of security protocols using partial-order reductions. *International Journal on Software Tools for Technology Transfer (STTT)*, 2003.

[CKR18a] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. DEEPSEC: Deciding equivalence properties in security protocols theory and practice. In *IEEE Symposium on Security and Privacy (S&P)*, 2018.

[CKR18b] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. The deepsec prover. In *International Conference on Computer Aided Verification (CAV)*, 2018.

[CKR19] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. Exploiting symmetries when proving equivalence properties for security protocols. In *ACM Conference on Computer and Communications Security (CCS)*, 2019.

[CKR20] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. The hitchhiker's guide to decidability and complexity of equivalence properties in security protocols. In *Andre Scedrov's Festschrift (ScedrovFest65)*, 2020.

[CKRY20] Vincent Cheval, Steve Kremer, Itsaka Rakotonirina, and Victor Yon. DeepSec v2.0.0, 2020. website: `https://deepsec-prover.github.io/`.

[CKW11] Véronique Cortier, Steve Kremer, and Bogdan Warinschi. A survey of symbolic methods in computational analysis of cryptographic systems. *Journal of Automated Reasoning*, 2011.

[CLD05] Hubert Comon-Lundh and Stéphanie Delaune. The finite variant property: How to get rid of some algebraic properties. In *International Conference on Rewriting Techniques and Applications*, 2005.

[CLW20] Véronique Cortier, Joseph Lallemand, and Bogdan Warinschi. Fifty shades of ballot privacy: Privacy against a malicious board. In *IEEE Computer Security Foundations Symposium (CSF)*, 2020.

[CR12] Yannick Chevalier and Michaël Rusinowitch. Decidability of equivalence of symbolic derivations. *Journal of Automated Reasoning*, 2012.

[CS10] Tom Chothia and Vitaliy Smirnov. A traceability attack against e-passports. In *International Conference on Financial Cryptography and Data Security (Financial CRYPTO)*, 2010.

[CS13] Véronique Cortier and Ben Smyth. Attacking and fixing helios: An analysis of ballot secrecy. *Journal of Computer Security*, 2013.

[Dam97] Mads Dam. On the decidability of process equivalences for the π-calculus. *Theoretical Computer Science*, 1997.

[DEK82] Danny Dolev, Shimon Even, and Richard M. Karp. On the security of ping-pong protocols. *Information and Control*, 1982.

[DH17] 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*, 2017.

[DKL+98] Jean-Francois Dhem, Francois Koeune, Philippe-Alexandre Leroux, Patrick Mestré, Jean-Jacques Quisquater, and Jean-Louis Willems. A practical implementation of the timing attack. In *International Conference on Smart Card Research and Advanced Applications*, 1998.

[DKP12] Stéphanie Delaune, Steve Kremer, and Daniel Pasaila. Security protocols, constraint systems, and group theories. In *International Joint Conference on Automated Reasoning*, 2012.

[DKR07] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Symbolic bisimulation for the applied pi calculus. In *International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2007.

[DKR09] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 2009.

[DLFP+21] Antoine Delignat-Lavaud, Cédric Fournet, Bryan Parno, Jonathan Protzenko, Tahina Ramananandro, Jay Bosamiya, Joseph Lallemand, Itsaka Rakotonirina, and Yi Zhou. A security model and fully verified implementation for the ietf quic record layer. In *IEEE Symposium on Security and Privacy (S&P)*, 2021.

[DLM04] Nancy A. Durgin, Patrick Lincoln, and John C. Mitchell. Multiset rewriting and the complexity of bounded security protocols. *Journal of Computer Security*, 2004.

[DLMS99] Nancy A. Durgin, Patrick Lincoln, John C. Mitchell, and Andre Scedrov. Undecidability of bounded security protocols. In *Proc. Workshop on formal methods in security protocols*, 1999.

[DNH84] Rocco De Nicola and Matthew CB Hennessy. Testing equivalences for processes. *Theoretical computer science*, 1984.

[DSV03] Luca Durante, Riccardo Sisto, and Adriano Valenzano. Automatic testing equivalence verification of spi calculus specifications. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2003.

[DY81] D. Dolev and A.C. Yao. On the security of public key protocols. In *Symposium on Foundations of Computer Science (FOCS)*, 1981.

[ES96] E Allen Emerson and A Prasad Sistla. Symmetry and model checking. *Formal methods in system design*, 1996.

[FHMS19] Ihor Filimonov, Ross Horne, Sjouke Mauw, and Zach Smith. Breaking unlinkability of the ICAO 9303 standard for e-passports using bisimilarity. In *European Symposium on Research in Computer Security (ESORICS)*, 2019.

[For04] PKI Task Force. PKI for machine readable travel documents offering ICC read-only access. Technical report, International Civil Aviation Organization, 2004.

[GHS+20] Guillaume Girol, Lucca Hirschi, Ralf Sasse, Dennis Jackson, Cas Cremers, and David Basin. A spectral analysis of noise: A comprehensive, automated, formal analysis of diffie-hellman protocols. In *USENIX Security Symposium*, 2020.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of computer and system sciences*, 1984.

[HBD16] Lucca Hirschi, David Baelde, and Stéphanie Delaune. A method for verifying privacy-type properties: the unbounded case. In *IEEE Symposium on Security and Privacy, (S&P)*, 2016.

[HS03] Hans Hüttel and Jirí Srba. Recursive ping-pong protocols. *BRICS Report Series*, 2003.

[Hüt03] Hans Hüttel. Deciding framed bisimilarity. *Electronic Notes in Theoretical Computer Science*, 2003.

[JK18] Charlie Jacomme and Steve Kremer. An extensive formal analysis of multi-factor authentication protocols. In *IEEE Computer Security Foundations Symposium (CSF)*, 2018.

[KBB17] Nadim Kobeissi, Karthikeyan Bhargavan, and Bruno Blanchet. Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2017.

[KKNS14] Max I. Kanovich, Tajana Ban Kirigin, Vivek Nigam, and Andre Scedrov. Bounded memory protocols. *Computer Languages, Systems & Structures*, 2014.

[Koc96] Paul C Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Annual International Cryptology Conference (CRYPTO)*, 1996.

[LL10] Jia Liu and Huimin Lin. A complete symbolic bisimulation for full applied pi calculus. In *International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, 2010.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I. *Inf. Comput.*, 1992.

[MVB10] Sebastian Mödersheim, Luca Vigano, and David Basin. Constraint differentiation: Search-space reduction for the constraint-based analysis of security protocols. *Journal of Computer Security*, 2010.

[Pap03] C.H. Papadimitriou. *Computational complexity.* John Wiley and Sons Ltd., 2003.

[Rak20] Itsaka Rakotonirina. DeepSec files used throughout this thesis, 2020. available at https://github.com/irakoton/phd-files/releases/tag/final.

[RK19] Itsaka Rakotonirina and Boris Köpf. On aggregation of information in timing attacks. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2019.

[RS06] Peter Y. A. Ryan and Steve A. Schneider. Prêt-à-voter with re-encryption mixes. In *European Symposium On Research In Computer Security (ESORICS)*, 2006.

[RT03] Michaël Rusinowitch and Mathieu Turuani. Protocol insecurity with a finite number of sessions, composed keys is NP-complete. *Theoretical Computer Science*, 2003.

[SEMM14] Sonia Santiago, Santiago Escobar, Catherine Meadows, and José Meseguer. A formal definition of protocol indistinguishability and its verification using maude-npa. In *International Workshop on Security and Trust Management*, 2014.

[SMCB13] Benedikt Schmidt, Simon Meier, Cas Cremers, and David Basin. The TAMARIN prover for the symbolic analysis of security protocols. In *International Conference on Computer Aided Verification (CAV)*, 2013.

[TD10] Alwen Tiu and Jeremy Dawson. Automating open bisimulation checking for the spi calculus. In *IEEE Computer Security Foundations Symposium (CSF)*, 2010.

[TNH16] Alwen Tiu, Nam Nguyen, and Ross Horne. SPEC: an equivalence checker for security protocols. In *Asian Symposium on Programming Languages and Systems (APLAS'16)*, 2016.

Appendix

Appendix A:

Proofs of Chapter 3

Summary.

In this appendix we prove the technical results used in Chapter 3 to decide trace equivalence and labelled bisimilarity of bounded processes using partition trees.

1 For trace equivalence

We first prove the two technical properties used in the criterion for trace equivalence. They essentially generalise the properties of the partition tree, stated for edges in Definition 3.10, to arbitrary branches. We first generalise the fact that the nodes of the tree are labelled by maximal configurations, i.e., Definition 3.10, Item 4:

Lemma 3.4

Assume that $(\mathcal{P}_1, \mathcal{C}_1), n \overset{\mathsf{tr}}{\Rightarrow}_T (\mathcal{P}'_1, \mathcal{C}'_1), n'$ and $(\mathcal{P}_2, \mathcal{C}_2) \overset{\mathsf{tr}}{\Rightarrow}_{\mathsf{s}} (\mathcal{P}'_2, \mathcal{C}'_2)$ with $(\mathcal{P}_2, \mathcal{C}_2) \in \Gamma(n)$. We also consider, for all $i \in \{1, 2\}$, a solution $(\Sigma', \sigma'_i) \in Sol^{\pi(n')}(\mathcal{C}'_i)$ such that $\Phi(\mathcal{C}'_1)\sigma'_1 \sim \Phi(\mathcal{C}'_2)\sigma'_2$. Then we have $(\mathcal{P}_2, \mathcal{C}_2), n \overset{\mathsf{tr}}{\Rightarrow}_T (\mathcal{P}'_2, \mathcal{C}'_2), n'$.

Proof. We proceed by induction on the length tr. The case $\mathsf{tr} = \varepsilon$ follows from the saturation of nodes under τ-transition (Definition 3.10, Item 1). Otherwise we let, with $\mathsf{tr} = \alpha \cdot \tilde{\mathsf{tr}}$,

$$(\mathcal{P}_1, \mathcal{C}_1), n \overset{\alpha}{\Rightarrow}_T (\tilde{\mathcal{P}}_1, \tilde{\mathcal{C}}_1), \tilde{n} \overset{\tilde{\mathsf{tr}}}{\Rightarrow}_T (\mathcal{P}'_1, \mathcal{C}'_1), n' \qquad (\mathcal{P}_2, \mathcal{C}_2) \overset{\alpha}{\Rightarrow}_{\mathsf{s}} (\tilde{\mathcal{P}}_2, \tilde{\mathcal{C}}_2) \overset{\tilde{\mathsf{tr}}}{\Rightarrow}_{\mathsf{s}} (\mathcal{P}'_2, \mathcal{C}'_2)$$

We also consider the restrictions $\Sigma = \Sigma'_{|vars^2(n)}$ and $\tilde{\Sigma} = \Sigma'_{|vars^2(\tilde{n})}$. In particular $\Sigma \subseteq \tilde{\Sigma}$ and there exist $\sigma_2, \tilde{\sigma}_1, \tilde{\sigma}_2$ such that

$$(\Sigma, \sigma_2) \in Sol(\mathcal{C}_2) \qquad (\tilde{\Sigma}, \tilde{\sigma}_1) \in Sol(\tilde{\mathcal{C}}_1) \qquad (\tilde{\Sigma}, \tilde{\sigma}_2) \in Sol(\tilde{\mathcal{C}}_2)$$

The hypothesis that $\Phi(\mathcal{C}'_1)\sigma'_1 \sim \Phi(\mathcal{C}'_2)\sigma'_2$ also implies that $\Phi(\tilde{\mathcal{C}}_1)\tilde{\sigma}_1 \sim \Phi(\tilde{\mathcal{C}}_2)\tilde{\sigma}_2$. Besides since predicates are refined along branches (Definition 3.10, Item 3) and are defined on the variables of their configurations (Definition 3.9, Item 1), we know that Σ and $\tilde{\Sigma}$ verify $\pi(n)$ and $\pi(\tilde{n})$, respectively.

All in all we can use the maximality of the node $\tilde{n}$ (Definition 3.10, Item 4 applied to the edge $n \xrightarrow{\alpha} \tilde{n}$), which gives that $(\tilde{\mathcal{P}}_2, \tilde{\mathcal{C}}_2) \in \Gamma(\tilde{n})$. Hence $(\mathcal{P}_2, \mathcal{C}_2), n \overset{\alpha}{\Rightarrow}_T (\tilde{\mathcal{P}}_2, \tilde{\mathcal{C}}_2), \tilde{n}$ by definition

and the conclusion then follows from the induction hypothesis applied to the remaining of the traces. □

We now generalise the fact symbolic transitions are reflected in the tree, in the sense of Definition 3.10, Item 2:

Lemma 3.5

Let n be a node of a partition tree T and $(\mathcal{P},\mathcal{C}) \in \Gamma(n)$. If $(\mathcal{P},\mathcal{C}) \overset{\mathsf{tr}}{\Rightarrow}_{\mathsf{s}} (\mathcal{P}',\mathcal{C}')$ and $(\Sigma,\sigma) \in Sol^{\pi(n)}(\mathcal{C}')$ then there exist a node n' and a substitution Σ' such that $(\mathcal{P},\mathcal{C}),n \overset{\mathsf{tr}}{\Rightarrow}_T (\mathcal{P}',\mathcal{C}'),n'$ and $(\Sigma',\sigma) \in Sol^{\pi(n')}(\mathcal{C}')$.

Proof. We proceed by induction on the length of tr. If $\mathsf{tr} = \varepsilon$ it sufficies to choose $n = n'$ and the conclusion immediately follows. Otherwise let us decompose the symbolic trace into

$$(\mathcal{P},\mathcal{C}) \overset{\tilde{\mathsf{tr}}}{\Rightarrow}_{\mathsf{s}} (\tilde{\mathcal{P}},\tilde{\mathcal{C}}) \overset{\alpha}{\Rightarrow}_{\mathsf{s}} (\mathcal{P}',\mathcal{C}') \qquad\qquad \mathsf{tr} = \tilde{\mathsf{tr}} \cdot \alpha$$

Note that $(\Sigma_{|vars^2(\tilde{n})}, \sigma_{|vars^1(\tilde{n})}) \in Sol(\tilde{\mathcal{C}})$ and that $\Sigma_{|vars^2(\tilde{n})}$ verifies $\pi(n)$ by definition of a configuration (since Σ verifies it and has the same restriction to $vars^2(\Gamma(n))$ as $\Sigma_{|vars^2(\tilde{n})}$). By induction hypothesis we therefore obtain $\tilde{n},\tilde{\Sigma}$ such that $(\mathcal{P},\mathcal{C}),n \overset{\tilde{\mathsf{tr}}}{\Rightarrow}_T (\tilde{\mathcal{P}},\tilde{\mathcal{C}}),\tilde{n}$ and $(\tilde{\Sigma},\sigma_{|vars^1(\tilde{n})}) \in Sol^{\pi(\tilde{n})}(\tilde{\mathcal{C}})$. Let us then consider the extension

$$\tilde{\Sigma}^e = \tilde{\Sigma} \cup \Sigma_{|vars^2(n') \setminus vars^2(\tilde{n})}$$

To conclude the proof it sufficies to apply the Item 2 of Definition 3.10 to the symbolic transition $(\tilde{\mathcal{P}},\tilde{\mathcal{C}}) \overset{\alpha}{\Rightarrow}_{\mathsf{s}} (\mathcal{P}',\mathcal{C}')$ and the solution $(\tilde{\Sigma}^e,\sigma)$; what remains to prove is therefore that we effectively have $(\tilde{\Sigma}^e,\sigma) \in Sol^{\pi(\tilde{n})}(\mathcal{C}')$. First of all we indeed have by construction $dom(\tilde{\Sigma}^e) = vars^2(\mathcal{C}')$ and $dom(\sigma) = vars^1(\mathcal{C}')$. We also know that $\tilde{\Sigma}^e$ satisfies the predicate $\pi(\tilde{n})$ because $\tilde{\Sigma} = \tilde{\Sigma}^e_{|vars^2(\tilde{n})}$ satisfies it. The first-order solution σ satisfies the constraints of $\mathsf{E}^1(\mathcal{C}')$ since $(\Sigma,\sigma) \in Sol(\mathcal{C}')$ by hypothesis. Finally we let $\varphi \in \mathsf{D}(\mathcal{C}')$ and prove that $(\Phi(\mathcal{C}'),\tilde{\Sigma}^e,\sigma) \models \varphi$:

- ▷ *case 1:* $\varphi = (X \vdash^? x) \in \mathsf{D}(\tilde{\mathcal{C}})$

  The conclusion follows from the fact that $(\tilde{\Sigma},\sigma_{|vars^1(\tilde{n})}) \in Sol(\tilde{\mathcal{C}})$.

- ▷ *case 2:* $\varphi = (X \vdash^? x) \in \mathsf{D}(\mathcal{C}') \setminus \mathsf{D}(\tilde{\mathcal{C}})$

  The conclusion follows from the fact that $(\Sigma,\sigma) \in Sol(\mathcal{C}')$.

- ▷ *case 3:* $\varphi = \forall X.\, X \not\vdash^? x$

  We have to prove that $x\sigma$ is not deducible from the frame $\Phi(\mathcal{C}')\sigma$, which is a consequence from the fact that $(\Sigma,\sigma) \in Sol(\mathcal{C}')$. □

2 For labelled bisimilarity

We now prove a technical lemma that proves the correctness of the criterion for deciding labelled bisimilarity (Theorem 3.7). We proceed by induction on the structure of the (symbolic) witness and have to strengthen the statement of the theorem for this purpose.

Lemma A.1

Let n be a node of a partition tree T and $A_0, A_1 \in \Gamma(n)$. We let $A_i = (\mathcal{P}_i, \mathcal{C}_i)$ and $\Sigma, \sigma_0, \sigma_1$ such that $(\Sigma, \sigma_i) \in Sol^{\pi(n)}(\mathcal{C}_i)$. If $A_i^c = (\mathcal{P}_i\sigma_i, \Phi(\mathcal{C}_i)\sigma_i\downarrow)$, the following points are equivalent:

1. $A_0^c \not\approx_l A_1^c$
2. there exist a symbolic witness w_s for A_0, A_1, n and a solution $f_{\mathsf{sol}} \in Sol(\mathsf{w}_s)$ such that $f_{\mathsf{sol}}(root(\mathsf{w}_s)) = \Sigma$

To prove this lemma we first observe that, by definition of a configuration (Definition 3.9), $A_0^c \sim A_1^c$ because these two processes are obtained by instanciating two symbolic processes from a same node n with a common solution Σ. We then prove the two directions of the theorem separately.

Proof of Lemma A.1, 1⇒2.

We prove the result by induction on $|\mathcal{P}_0, \mathcal{P}_1|$. The conclusion is immediate if $|\mathcal{P}_0, \mathcal{P}_1| = 0$ as it yields a contradiction: the multisets $\mathcal{P}_0$ and $\mathcal{P}_1$ can only contain null processes and the fact that $A_0^c \sim A_1^c$ justifies that $A_0^c \approx_l A_1^c$. Otherwise we let by Proposition 3.6 a witness w of (A_0^c, A_1^c). Thus, by definition, there exist $b \in \{0,1\}$ and a transition $A_b^c \xrightarrow{\alpha} A_b'^c = (\mathcal{Q}, \Phi)$ such that for all traces $A_{1-b}^c \stackrel{\bar{\alpha}}{\Rightarrow} A_{1-b}'^c$ such that $A_0'^c \sim A_0'^c$, we have $(A_0'^c, A_1'^c) \in \mathsf{w}$ (and therefore $A_0'^c \not\approx_l A_1'^c$ by Proposition 3.6). Let us now construct a symbolic witness w_s of A_0, A_1, n and a suitable solution f_{sol}.

▷ *case 1:* $\alpha \neq \tau$

By completeness of the symbolic semantics (Proposition 3.2) applied to the transition $A_b^c \xrightarrow{\alpha} A_b'^c$, we let a symbolic transition $A_b \xrightarrow{\alpha_s}_{\mathsf{s}} A_b' = (\mathcal{Q}_s, \mathcal{C})$ and a solution $(\Sigma', \sigma') \in Sol(\mathcal{C})$ such that $\Sigma \subseteq \Sigma'$, $\alpha = \alpha_s\Sigma'$, $\mathcal{Q} = \mathcal{Q}_s\sigma'$ and $\Phi = \Phi(\mathcal{C})\sigma'\downarrow$. Note that by hypothesis Σ verifies $\pi(n)$ and, therefore, so does its extension Σ' (since by definition predicates are stable by domain extension, recall Definition 3.9). Then since the symbolic transition $A_b \xrightarrow{\alpha_s}_{\mathsf{s}} A_b'$ is reflected in T (in the sense of Definition 3.10, Item 2), we obtain a transition $A_b, n \xrightarrow{\alpha_s}_T A_b', n'$ and Σ'' such that $(\Sigma'', \sigma') \in Sol^{\pi(n')}(\mathcal{C})$ and $\Sigma''_{|vars^2(n)} = \Sigma'_{|vars^2(n)} \quad (= \Sigma)$.

▷ *case 1a*: there exist no A'_{1-b} such that $A_{1-b}, n \stackrel{\alpha_s}{\Longrightarrow}_T A'_{1-b}, n'$

Then we define w_s to be the tree whose root is labelled $(\{A_0, A_1\}, n)$ and that has a unique child labelled $(\{A_b'\}, n')$. We then consider f_{sol} mapping the child to Σ'' and the root to $\Sigma''_{|vars^2(n)} = \Sigma$, which is a solution of w_s.

▷ *case 1b*: otherwise

In this case we define w_s as follows. Its root is labelled $(\{A_0, A_1\}, n)$ and its children are all the nodes labelled $(\{A_0', A_1'\}, n')$, with $A_{1-b}, n \stackrel{\alpha_s}{\Longrightarrow}_T A'_{1-b}, n'$. For each such node, as explained in the beginning of the proof we have $A_0'^c \not\approx_l A_1'^c$ which permits to apply the induction hypothesis with the solution Σ''. This gives a symbolic witness rooted in this node and f_{sol} a solution mapping this node to Σ''. Let us write more explicitly these witnesses $\mathsf{w}_s^1, \ldots, \mathsf{w}_s^p$ and $f_{\mathsf{sol}}^1, \ldots, f_{\mathsf{sol}}^p$ the corresponding solutions. To conclude it then sufficies to choose $\mathsf{w}_s^1, \ldots, \mathsf{w}_s^p$ as the children of the root of w_s, and f_{sol} maps the root of w_s to $\Sigma''_{|vars^2(n)} = \Sigma$ and each node n of w_s^i to $f_{\mathsf{sol}}^i(n)$.

▷ *case 2:* $\alpha = \tau$

Analogue to case 1 in the simpler case where $n = n'$ and $\Sigma = \Sigma' = \Sigma''$. Note also that the analogue of case 1a cannot arise. □

Proof of Lemma A.1, 2⇒1.
We construct a concrete witness w of (A_0^c, A_1^c) as follows:

$$\mathsf{w} = \left\{ ((\mathcal{P}_0\sigma_0, \Phi(\mathcal{C}_0)\sigma_0\downarrow), (\mathcal{P}_1\sigma_1, \Phi(\mathcal{C}_1)\sigma_1\downarrow)) \;\middle|\; \begin{array}{l} N \text{ node of } \mathsf{w}_s \text{ labelled } ((\mathcal{P}_0, \mathcal{C}_0), (\mathcal{P}_1, \mathcal{C}_1), n), \\ \forall i \in \{0,1\}, (f_{\mathsf{sol}}(N), \sigma_i) \in Sol(\mathcal{C}_i) \end{array} \right\}$$

The fact that all $(B_0, B_1) \in \mathsf{w}$ verify $B_0 \sim B_1$ follows from Definition 3.9. Then let us consider $((\mathcal{P}_0\sigma_0, \Phi(\mathcal{C}_0)\sigma_0\downarrow), (\mathcal{P}_1\sigma_1, \Phi(\mathcal{C}_1)\sigma_1\downarrow)) \in \mathsf{w}$ using the notations of the construction of w above. By definition of a symbolic witness there exists $b \in \{0,1\}$ and a transition $(\mathcal{P}_b, \mathcal{C}_b), n \xrightarrow{\alpha}_T (\mathcal{P}'_b, \mathcal{C}'_b), n'$ such that:

▷ *case 1:* $N = root(\mathsf{w}_s)$ has a unique child N' labelled $\{(\mathcal{P}'_b, \mathcal{C}'_b)\}, n'$

Then consider the concrete transition $(\mathcal{P}_b\sigma_b, \Phi(\mathcal{C}_b)\sigma_b\downarrow) \xrightarrow{\alpha f_{\mathsf{sol}}(N')} (\mathcal{P}'_b\sigma'_b, \Phi(\mathcal{C}'_b)\sigma'_b\downarrow)$ obtained by soundness of the symbolic semantics (Proposition 3.2) where $(f_{\mathsf{sol}}(N'), \sigma'_b) \in Sol^{\pi(n')}(\mathcal{C}'_b)$. By completeness of the symbolic semantics (which is possible to apply since $f_{\mathsf{sol}}(N) \subseteq f_{\mathsf{sol}}(N')$ by definition of a solution of a symbolic witness) and maximality of the node n' (Definition 3.10, Item 4), there cannot exist any concrete trace of the form

$$(\mathcal{P}_{1-b}\sigma_{1-b}, \Phi(\mathcal{C}_{1-b})\sigma_{1-b}\downarrow) \xrightarrow{\alpha f_{\mathsf{sol}}(N')} (\mathcal{P}, \Phi)$$

such that $\Phi \sim \Phi(\mathcal{C}'_b)\sigma'_b$, hence the conclusion.

▷ *case 2:* the children of $N = root(\mathsf{w}_s)$ are all the nodes N' labelled $(\{(\mathcal{P}'_0, \mathcal{C}'_0), (\mathcal{P}'_1, \mathcal{C}'_1)\}, n')$, where $(\mathcal{P}_{1-b}, \mathcal{C}_{1-b}), n \overset{\alpha}{\Rightarrow}_T (\mathcal{P}'_{1-b}, \mathcal{C}'_{1-b}), n'$ (and there is at least one such child)

Let N' be an arbitrary child of N, labelled $(\{(\mathcal{P}'_0, \mathcal{C}'_0), (\mathcal{P}'_1, \mathcal{C}'_1)\}, n')$ with the above notations. As in the previous case we consider the concrete transition obtained by soundness of the symbolic semantics, $(\mathcal{P}_b\sigma_b, \Phi(\mathcal{C}_b)\sigma_b\downarrow) \xrightarrow{\alpha f_{\mathsf{sol}}(N')} (\mathcal{P}'_b\sigma'_b, \Phi(\mathcal{C}'_b)\sigma'_b\downarrow)$. Then let us consider a trace of the form

$$(\mathcal{P}_{1-b}\sigma_{1-b}, \Phi(\mathcal{C}_{1-b})\sigma_{1-b}\downarrow) \xRightarrow{\alpha f_{\mathsf{sol}}(N')} (\mathcal{P}, \Phi) = A \qquad\qquad \Phi \sim \Phi(\mathcal{C}'_b)\sigma'_b$$

Our goal is to prove that $(A, (\mathcal{P}_b\sigma_b, \Phi(\mathcal{C}_b)\sigma_b\downarrow)) \in \mathsf{w}$. Using the completeness of the symbolic semantics and the maximality of n' as in the previous case, we obtain a partition-tree trace $(\mathcal{P}_{1-b}, \mathcal{C}_{1-b}), n \overset{\alpha}{\Rightarrow}_T (\mathcal{P}''_{1-b}, \mathcal{C}''_{1-b}), n'$ and $(f_{\mathsf{sol}}(N'), \sigma''_{1-b}) \in Sol^{\pi(n')}(\mathcal{C}''_{1-b})$ with $\mathcal{P} = \mathcal{P}''_{1-b}\sigma''_{1-b}$ and $\Phi = \Phi(\mathcal{C}''_{1-b})\sigma''_{1-b}\downarrow$. By hypothesis there therefore exists a node N'' labelled $(\{(\mathcal{P}'_b, \mathcal{C}'_b), (\mathcal{P}''_{1-b}, \mathcal{C}''_{1-b})\}, n')$ a child of N. The conclusion then follows from the fact that $f_{\mathsf{sol}}(N') = f_{\mathsf{sol}}(N'')$ by definition of a solution of a symbolic witness. □

Appendix B:

Proofs of Chapter 4

Summary.

In this appendix we prove the technical results used in Chapter 4 to construct the partition tree of two bounded processes using constraint solving.

1 Invariants of the procedure

In this section we present some additional properties that are verified all along the procedure by the nodes of the partition tree under construction. Such nodes are sets of extended symbolic processes (i.e. tuples $(\mathcal{P}, \mathcal{C}, \mathcal{C}^e)$ with $\mathcal{P}$ a process, $\mathcal{C}$ a constraint system and $\mathcal{C}^e$ an extended constraint system). Understanding the technical details of these invariants is not necessary to understand the algorithm itself, however most of our subprocedure (e.g. the generation of most general solutions) are only correct in their context.

Invariant 1: Well-formedness The first invariant is about the shape of the extended constraint systems. Two important properties are that all equations of E^1 and E^2 are trivially satisfiable (they are essentially of the form $x =^? u$ where x appears nowhere else in the constraint system) and that those of E^2 only use terms that can be constructed from the knowledge base (i.e. they are consequences of $\mathsf{K} \cup \mathsf{D}$).

Definition B.1

We define the predicate Inv_{wf} on extended constraint systems as follows; we have that $\mathsf{Inv}_{wf}((\Phi, \mathsf{D}, \mathsf{E}^1, \mathsf{E}^2, \mathsf{K}, \mathsf{F}))$ holds when

- Variables in K and F: $vars^2(\mathsf{K}, \mathsf{F}) \subseteq vars^2(\mathsf{D})$
- Equation: $mgu(\mathsf{E}^i) \neq \bot$, $dom(mgu(\mathsf{E}^i)) \cap vars^i(\mathsf{D}) = \varnothing$, $vars^i(img(mgu(\mathsf{E}^i))) \subseteq vars^i(\mathsf{D})$.
- Solution is consequence: $img(mgu(\mathsf{E}^2)) \subseteq \mathsf{Conseq}(\mathsf{K} \cup \mathsf{D})$.
- Shape of K: For all $\psi = (\xi \vdash^? u) \in \mathsf{K}$, $\psi \in \mathsf{F}$, $u \notin \mathcal{X}^1$, $u \in subterms(\Phi)$ and $subterms(\xi) \subseteq \mathsf{Conseq}(\mathsf{K} \cup \mathsf{D})$.
- Shape of F: For all $\forall S.\, \varphi \Rightarrow H \in \mathsf{F}(\mathcal{C})$, S is empty and φ only contains syntactic equations as hypothesis, i.e. no deduction facts. Moreover $subterms^2_{\neq}(H) \subseteq \mathsf{Conseq}(\mathsf{K} \cup \mathsf{D})$, and if $H = \xi \vdash^? u$ then either $u \in subterms(\Phi)$ or there exists a recipe ζ such that $(\zeta, u) \in \mathsf{Conseq}(\mathsf{K} \cup \mathsf{D})$.

This invariant is lifted to sets of extended constraint systems in the natural way, i.e. $\mathsf{Inv}_{wf}(S)$ holds *iff* for all $\mathcal{C} \in S$, $\mathsf{Inv}_{wf}(\mathcal{C})$ holds.

Invariant 2: Formula soundness The second invariant states that any substitution that satisfies the deduction facts of D and the equalities of E^1 also satisfies all formulas of $\mathsf{K} \cup \mathsf{F}$. This means that the procedure only adds correct formulas to the constraint system, sometimes under some hypothese for the formulas of F.

Definition B.2

We define the predicate Inv_{sound} on extended constraint systems as follows; we have that $\mathsf{Inv}_{sound}((\Phi, \mathsf{D}, \mathsf{E}^1, \mathsf{E}^2, \mathsf{K}, \mathsf{F}))$ holds when for all $\psi \in \mathsf{K} \cup \mathsf{F}$, for all substitutions Σ, σ, if

$$(\Phi, \Sigma, \sigma) \models \mathsf{D}' \wedge \mathsf{E}^1{}_{=} \qquad \text{with } \mathsf{D}' = \{\psi' \in \mathsf{D} \mid vars^2(\psi') \subseteq vars^2(\psi)\}$$

where $\mathsf{E}^1{}_{=}$ if the set of equations of E^1, then $(\Phi, \Sigma, \sigma) \models \psi$.

Invariant 3: Formula completeness Given a formula $\psi = \varphi \Rightarrow H$ in F, the soundness invariant above states that when some substitutions satisfy φ then they also satisfy the head H. The next invariant can be seen as a kind of converse statement: when some substitutions satisfy the head H, there exists a formula $\psi' \in \mathsf{F}$ (not necessarily the same) that has the same head and whose hypotheses are satisfied. This means that the procedure always covers all cases when generating the potential hypotheses of a given head H.

Definition B.3

In this definition, we say that (Φ, Σ, σ) *weakly satisfies* a head H when:

- if $H = \xi \vdash^? u$ then $\mathsf{msg}(\xi\Sigma\Phi\sigma)$, i.e. the recipe ξ leads to a valid message
- if $H = \xi =^?_f \zeta$ then (Φ, Σ, σ) satisfies H in the usual sense, i.e. the recipes ξ, ζ lead to the same valid message.

Given a set of extended processes Γ, the predicate $\mathsf{Inv}_{comp}(\Gamma)$ holds when for all $\mathcal{C}^e_1, \mathcal{C}^e_2 \in \Gamma$, for all $(\varphi \Rightarrow H) \in \mathsf{F}(\mathcal{C}^e_1)$, for all $(\Sigma, \sigma) \in Sol(\mathcal{C}^e_2)$, if $(\Phi(\mathcal{C}^e_2), \Sigma, \sigma)$ weakly satisfies H then there exists $(\varphi' \Rightarrow H') \in \mathsf{F}(\mathcal{C}^e_2)$ such that $H' \simeq_{\mathsf{h}} H$ and $(\Phi(\mathcal{C}^e_2), \Sigma, \sigma) \models \varphi'$.

Invariant 4: Knowledge-base saturation The next invariant states that the knowledge base K is saturated, i.e. that we do not miss solutions by imposing that they are constructed from K (see Definition 4.4).

Definition B.4

We define the predicate Inv_{satur} on extended constraint systems as follows. We have that $\mathsf{Inv}_{satur}(\mathcal{C})$ holds when, considering the minimal k such that $vars^2(\mathsf{D}(\mathcal{C})) \subseteq \mathcal{X}^2_{\leqslant k}$, for all $\mathsf{f}/n \in \mathcal{F}_{\mathsf{d}}$, for all $(\xi_1, u_1), \ldots, (\xi_n, u_n) \in \mathsf{Conseq}(\mathsf{K}(\mathcal{C}))$ such that $\xi_1, \ldots, \xi_n \in \mathcal{T}^2_k$, if $\mathsf{f}(u_1, \ldots, u_n)\!\downarrow$ is a constructor term then there is $\xi \in \mathcal{T}^2_k$ such that $(\xi, \mathsf{f}(u_1, \ldots, u_n)\!\downarrow) \in \mathsf{Conseq}(\mathsf{K}(\mathcal{C}) \cup \mathsf{D}(\mathcal{C}))$.

Invariant 5: Preservation of solutions Finally the last invariant states that all the constraint solving performed on the additional data of extended constraint systems (E^2, K, F) preserves the solutions. That is, in an extended symbolic process $(\mathcal{P}, \mathcal{C}, \mathcal{C}^e)$, where $\mathcal{C}$ is the constraint system obtained by only collecting constraints during the execution of the process $\mathcal{P}$ (i.e. without additional constraint solving), all solutions of $\mathcal{C}^e$ are solutions of $\mathcal{C}$.

Definition B.5

We define the predicate Inv_{sol} defined on extended symbolic processes where $\mathsf{Inv}_{sol}((\mathcal{P}, \mathcal{C}, \mathcal{C}^e))$ holds when

- for all $i \in \mathbb{N}$, $vars^2(\mathcal{C}) \subseteq \mathcal{X}^2_{\leqslant i}$ iff $vars^2(\mathcal{C}^e) \subseteq \mathcal{X}^2_{\leqslant i}$.
- for all $(\Sigma, \sigma) \in Sol(\mathcal{C}^e)$, $(\Sigma_{|vars^2(\mathcal{C})}, \sigma_{|vars^1(\mathcal{C})}) \in Sol(\mathcal{C})$.

We restrict the substitutions to the variables of $\mathcal{C}$ since our extended constraint system may introduce new variables (e.g. by applying most general solutions) but all these variables are uniquely defined by the instantiation of the variables of $\mathcal{C}$.

Invariant 6: Component structure Finally we state an invariant on components stating that all of their constraint systems have the same second-order structure. This invariant is preserved during the procedure by the fact that the constraint-solving rules that modify K or E^2 are always applied to the entire components.

Definition B.6

We define the predicate Inv_{str} defined on sets of extended symbolic processes where $\mathsf{Inv}_{sol}(\Gamma)$ holds when for all $(\mathcal{P}_1, \mathcal{C}_1, \mathcal{C}^e_1), (\mathcal{P}_2, \mathcal{C}_2, \mathcal{C}^e_2) \in \Gamma$,

- $dom(\Phi(\mathcal{C}^e_1)) = dom(\Phi(\mathcal{C}^e_2))$
- $vars^2(\mathcal{C}^e_1) = vars^2(\mathcal{C}^e_2)$
- $\{\xi \mid (\xi \vdash^? u) \in \mathsf{K}(\mathcal{C}^e_1)\} = \{\xi \mid (\xi \vdash^? u) \in \mathsf{K}(\mathcal{C}^e_2)\}$

Overall invariant As we mentioned, all invariants are lifted to sets of extended symbolic processes in the natural way if needed. We refer as

$$\mathsf{Inv}_{all}(\Gamma) = \mathsf{Inv}_{wf}(\Gamma) \wedge \mathsf{Inv}_{sound}(\Gamma) \wedge \mathsf{Inv}_{comp}(\Gamma) \wedge \mathsf{Inv}_{satur}(\Gamma) \wedge \mathsf{Inv}_{sol}(\Gamma) \wedge \mathsf{Inv}_{str}(\Gamma)$$

the invariant of the whole procedure on the nodes of the partition tree (i.e. sets of extended symbolic processes).

2 Preservation of the invariants

In this section we prove that the invariants of the procedure stated in Section 1 are preserved all along the procedure. First of all we prove the case of the case distinction rules:

Lemma B.1

Let $\mathbb{S}$ be a set of set of extended symbolic processes such that $\mathsf{Inv}_{all}(\mathbb{S})$. Let $\mathbb{S} \to \mathbb{S}'$ by

applying one case distinction rule and then normalising the result with the simplification rules. We have that $\mathsf{Inv}_{all}(\mathbb{S}')$.

For the proof we let $\Gamma' \in \mathbb{S}'$ and write $\Gamma \in \mathbb{S}$ the set from which Γ' is originated, i.e. Γ' is one of the sets obtained after applying one case distinction rule to Γ (either the positive or the negative branch) and then normalising with the simplification rules.

Proof of preservation of Inv_{wf}. Let $\mathcal{C}^{e\prime} = (\Phi, \mathsf{D}, \mathsf{E}^1, \mathsf{E}^2, \mathsf{K}, \mathsf{F})$ for some $(\mathcal{P}', \mathcal{C}', \mathcal{C}^{e\prime}) \in \Gamma'$. We consider each item of the definition of the predicate (Definition B.1) and show that $\mathcal{C}^{e\prime}$ verifies them.

▷ *property 1 (Variables in* K *and* F*):* $vars^2(\mathsf{K}, \mathsf{F}) \subseteq vars^2(\mathsf{D})$.

We first observe that this property is preserved by all simplification rules: therefore it suffices to prove that it is preserved by application of case distinction rules. For that we also observe that if $\mathcal{C}^e$ is an extended constraint system verifying this property then for all second-oder substitutions Σ, $\mathcal{C}^e{:}\Sigma$ verifies it as well (which is precisely why we consider this notation rather than a raw application $\mathcal{C}^e\Sigma$). This is sufficient for getting the expected result for Rule (Sat). The case of the negative branches of Rules (Rew) and (Eq) are trivial. Regarding their positive branches, we only treat the case of Rule (Eq) since the treatment of (Rew) can be obtained by using a similar reasoning on each formulas added by the rule (the sets F_0 in the notations of Rule (Rew)). The rule (Eq) does not introduce elements in K and the only second-oder variables introduced in F that are not trivially added to D are from recipes ξ_1, ξ_2 such that $\xi_1 \vdash^? u_1, \xi_2 \vdash^? u_2 \in \mathsf{K}(\mathcal{C}^e)$ for some $(\mathcal{P}, \mathcal{C}, \mathcal{C}^e) \in \Gamma$. In particular by hypothesis $vars^2(\xi_1, \xi_2) \subseteq vars^2(\mathsf{D}(\mathcal{C}^e)) \subseteq vars^2(\mathsf{D}(\mathcal{C}^{e\prime}))$, hence the conclusion.

▷ *property 2 (first and second-order equations):* we have $mgu(\mathsf{E}^i) \neq \bot$, $dom(mgu(\mathsf{E}^i)) \cap vars^i(\mathsf{D}) = \varnothing$, and $vars^i(img(mgu(\mathsf{E}^i))) \subseteq vars^i(\mathsf{D})$.

The fact that $mgu(\mathsf{E}^i) \neq \bot$ is a direct consequence of the facts that $\mathcal{C}^{e\prime} \neq \bot$ and that $\mathcal{C}^{e\prime}$ is already normalised by the simplification rules of Figure 4.2, in particular Rules Norm-Unif and the rules inherited from Figure 4.1. The remaining properties are simple invariants of the mgu's that are straightforward to verify.

▷ *property 3 (Solution is consequence):* $img(mgu(\mathsf{E}^2)) \subseteq \mathsf{Conseq}(\mathsf{K} \cup \mathsf{D})$.

This property comes from the fact that $mgu(\mathsf{E}^2)$ is only modified in the positive branches of the case distinction rules by applying a mgs Σ to the extended constraint systems $\mathcal{C}^e \in \Gamma$. A quick induction on the constraint solving relation for computing mgs $\rightsquigarrow$ show that $img(\Sigma) \subseteq \mathsf{Conseq}(\mathsf{K}(\mathcal{C}^e) \cup \mathsf{D}(\mathcal{C}^e))$, hence the conclusion.

▷ *property 4 (Shape of* K*):* for all $\psi = (\xi \vdash^? u) \in \mathsf{K}$, $\psi \in \mathsf{F}$, $u \notin \mathcal{X}^1$, $u \in subterms(\Phi)$ and $subterms(\xi) \subseteq \mathsf{Conseq}(\mathsf{K} \cup \mathsf{D})$.

The only rule adding a deduction fact to K is the rule (Vect-add-Conseq); in particular this gives $\psi \in \mathsf{F}$. The added deduction facts are of the form $\xi \vdash^? u$ where, for some $(\mathcal{P}, \mathcal{C}, \mathcal{C}^e) \in \Gamma$, $\xi \vdash^? u \in \mathsf{F}(\mathcal{C}^e)$ and $(\zeta, u) \notin \mathsf{Conseq}(\mathsf{K}(\mathcal{C}^e) \cup \mathsf{D}(\mathcal{C}^e))$ for any recipe ζ. The fact that $\xi \vdash^? u \in \mathsf{F}(\mathcal{C}^e)$ and $\mathsf{Inv}_{wf}(\Gamma)$ (property 5) justify that $subterms(\xi) \subseteq \mathsf{Conseq}(\mathsf{K} \cup \mathsf{D})$; this justifies that $u \in subterms(\Phi)$ as well when taking into account that u is not a consequence of $\mathsf{D}(\mathcal{C}^e)$ (in particular it cannot be a ground contructor term without names otherwise it would even be consequence of the empty set). Finally the property $u \notin \mathcal{X}^1$ also follows from

(ζ, u) not being a consequence of $\mathsf{D}(\mathcal{C}^e)$.

▷ *property 5 (Shape of* F*):* For all $\varphi \Rightarrow H \in \mathsf{F}(\mathcal{C}^{e\prime})$, φ only contains syntactic equations as hypothesis, i.e. no deduction facts. Moreover $\mathit{subterms}^2_{\neq}(H) \subseteq \mathsf{Conseq}(\mathsf{K} \cup \mathsf{D})$, and if $H = \xi \vdash^? u$ then either $u \in \mathit{subterms}(\Phi)$ or there exists a recipe ζ such that $(\zeta, u) \in \mathsf{Conseq}(\mathsf{K} \cup \mathsf{D})$.

The property that the hypotheses of the formula only contain syntactic equations can be obtained by a straightforward inspection of each case-distinction and simplification rules. Note in particular that whenever a formula ψ with deduction-fact hypotheses is considered (in Rules (Rew) and (Eq)), they are applied to a substitution Σ so that $\psi{:}(\Sigma, \mathcal{C}^e)$ only has equations as hypotheses. As for the second part of property 5 we write 1 the property $\mathit{subterms}^2_{\neq}(H) \subseteq \mathsf{Conseq}(\mathsf{K} \cup \mathsf{D})$ and 2 the rest (the property about the head terms of deduction formulas). We perform a case analysis on the rule that added the formula to Γ'.

- rule (Vect-add-Formula): the proof of 2 directly follows from $\mathsf{Inv}_{wf}(\Gamma)$ since the rule does not add a deduction formula. Regarding 1, using the notations of the rule, the head of the formula is $\xi =^?_f \zeta$ where $\xi \vdash^? u \in \mathsf{F}$ for some u, and $\zeta \in \mathsf{Conseq}(\mathsf{K} \cup \mathsf{D})$. In particular the conclusion follows from $\mathsf{Inv}_{wf}(\Gamma)$ (property 4 for the subterms of ξ and property 5 for the subterms of ζ)
- rule (Rew): we first prove 1. Using the notations of the rule, all non-root positions of the head of a formula of F_0 are either variables X such that D contains a deduction fact $X \vdash^? x$ (generated by the skeleton), a position of ξ_0, or a public function symbol (since the rewriting system is constructor, the left-hand sides ℓ of the rewrite rules only contain a destructor at their roots). In particular all strict subterms of H are consequences of $\mathsf{K} \cup \mathsf{D}$. Now let us prove 2. By definition of the rule, u is a constructor term in normal form obtained after applying one rewrite rule $\ell \to r$ at the root of $C[u_0]$ for some context C (not containing names but possibly containing variables x such that $X \vdash^? x \in \mathsf{D}$ for some X). We recall that the rewriting system is constructor-destructor and subterm convergent, which leaves two cases. The first is that r is a ground constructor term, and then $u = r$ is trivially a consequence of $\mathsf{K} \cup \mathsf{D}$ for the recipe $\zeta = r$. Otherwise r is a subterm of ℓ, meaning that u is a subterm of $C[u_0]$. This implies that u is either a subterm of u_0, a subterm of the context C, or a term of the form $C'[u_0]$ for some subcontext C' of C. In all cases, since $u_0 \in \mathit{subterms}(\Phi)$ by $\mathsf{Inv}_{wf}(\Gamma)$ (property 4), this gives the expected conclusion.
- rule (Eq): the proof of 2 follows from $\mathsf{Inv}_{wf}(\Gamma)$ since the rule does not add a deduction formula. Regarding 1, this follows from $\mathsf{Inv}_{wf}(\Gamma)$ (property 4). □

Proof of preservation of Inv_{sound}. Let $\mathcal{C}' = (\Phi, \mathsf{D}, \mathsf{E}^1, \mathsf{E}^2, \mathsf{K}, \mathsf{F})$ for some $(\mathcal{P}', \mathcal{C}', \mathcal{C}^{e\prime}) \in \Gamma'$, $\psi \in \mathsf{K} \cup \mathsf{F}$, and (Σ, σ) such that

$$(\Phi, \Sigma, \sigma) \models \bigwedge_{\substack{\psi' \in \mathsf{D} \\ vars^2(\psi') \subseteq vars^2(\psi)}} \psi' \quad \wedge \mathsf{E}^1{}_{=}$$

We have to prove that $(\Phi, \Sigma, \sigma) \models \psi$. Here we prove more precisely that the conclusion holds when applying one time any of the case-distinction or simplification rules. First of all we

observe that this property is preserved by the application of a mgs to an extended constraint system. In particular this makes the conclusion immediate for all rules except the following:

- Rule (Rew) (positive branch): using the notations of the rule and recalling $\mathsf{Inv}_{sound}(\Gamma)$, the conclusion follows from the fact that if $\xi_0 \Sigma \Phi \sigma\downarrow = u_0\downarrow$, $\ell \to r \in \mathcal{R}$ and $C[u] = \ell\sigma'$ for some substitution σ', then $C[\xi_0]\Sigma\Phi\sigma\downarrow = r\sigma'\downarrow$.
- Rule (Eq) (positive branch): using the notations of the rule and recalling $\mathsf{Inv}_{sound}(\Gamma)$, if ψ is the formula added to F by this rule, we have $(\Phi, \Sigma, \sigma) \models \psi$ iff for some recipes ξ_1, ξ_2, if $\xi_1\Sigma\Phi\sigma\downarrow = z$ and $\xi_2\Sigma\Phi\sigma\downarrow = z$ then $\xi_1\Sigma\Phi\sigma\downarrow = \xi_2\Sigma\Phi\sigma\downarrow$. This naturally holds.
- Rule (Vect-add-Conseq): since the element added to K is originated from F, the conclusion follows from $\mathsf{Inv}_{sound}(\Gamma)$.
- Rule (Vect-add-Formula): the reasoning is identical to the one for (Eq). □

Proof of preservation of Inv_{comp}. Let $\mathcal{C}_1^{e\prime}, \mathcal{C}_2^{e\prime}$ for some $(\mathcal{P}_1', \mathcal{C}_1', \mathcal{C}_1^{e\prime}), (\mathcal{P}_2', \mathcal{C}_2', \mathcal{C}_2^{e\prime}) \in \Gamma'$, $\psi = \varphi \Rightarrow H \in \mathsf{F}(\mathcal{C}_1^{e\prime})$ and $(\Sigma, \sigma) \in Sol(\mathcal{C}_2^{e\prime})$. We assume that $(\Phi(\mathcal{C}_2^{e\prime}), \Sigma, \sigma)$ weakly satisfies H. We want to prove that there exists $\psi' = (\varphi' \Rightarrow H') \in \mathsf{F}(\mathcal{C}_2^{e\prime})$ such that $H' \simeq_{\mathsf{h}} H$ and $(\Phi(\mathcal{C}_2^{e\prime}), \Sigma, \sigma) \models \varphi'$. We prove the strenghtened property stating that the invariant is preserved after the application of each case-distinction and simplification rules.

- Rules (Norm-Unif), (Norm-no-MGS), (Norm-Diseq): the conclusion directly follows from $\mathsf{Inv}_{comp}(\Gamma)$.
- Rule (Norm-Formula): by $\mathsf{Inv}_{comp}(\Gamma)$ we let $\psi_0' = (\varphi_0' \Rightarrow H_0') \in \mathsf{F}(\mathcal{C}_2^e)$ such that $H_0' \simeq_{\mathsf{h}} H$ and $(\Phi(\mathcal{C}_2^e), \Sigma, \sigma) \models \varphi_0'$. The conclusion directly follows from this, except if ψ_0' is the formula removed by the rule, i.e. if $mgs(\mathcal{C}_2^e[\mathsf{E}^1 \mapsto \mathsf{E}^1 \wedge \varphi_0']) = \varnothing$. However this would yield a contradiction with $(\Phi(\mathcal{C}_2^e), \Sigma, \sigma_2) \models \varphi_0'$, hence the conclusion.
- Rule (Norm-Dupl): using the same notations as in the previous case, we let ψ_0' by $\mathsf{Inv}_{comp}(\Gamma)$ and the only non-trivial case is again the one where ψ_0' is removed from $\mathcal{C}_2^e$ by the rule. This means that there exists $\psi' \in \mathsf{F}(\mathcal{C}_2^{e\prime})$ solved such that $\psi' \simeq_{\mathsf{h}} H_0'$, hence the conclusion since $\mathsf{hyp}(\psi') = \top$ and $H_0' \simeq_{\mathsf{h}} H$.
- Rules (Vect-rm-Unsat) and (Vect-Split): the conclusion follows from the $\mathsf{Inv}_{comp}(\Gamma)$ since $\Gamma' \subseteq \Gamma$.
- Rule (Vect-add-Conseq): directly follows from $\mathsf{Inv}_{comp}(\Gamma)$.
- Rule (Vect-add-Formula): Let us write

$$\Gamma' = \{(\mathcal{P}, \mathcal{C}, \mathcal{C}^e[\mathsf{F} \mapsto \mathsf{F} \wedge \psi{:}(\Sigma, \mathcal{C}^e)]) \mid (\mathcal{P}, \mathcal{C}, \mathcal{C}^e) \in \Gamma\}$$

  with the notations and assumptions of the rule. If $i \in \{1, 2\}$ we write $S_i = (\mathcal{P}, \mathcal{C}, \mathcal{C}_i^e)$. The only case that does not directly follows from $\mathsf{Inv}_{comp}(\Gamma)$ is the case where $\psi = \psi{:}(\Sigma, \mathcal{C}_1^e)$. But since there exists a formula $\xi \vdash^? u_{S_2} \in \mathsf{F}(\mathcal{C}_2^e)$ by hypothesis, we know by $\mathsf{Inv}_{sound}(\Gamma)$ that $\xi\Phi(\mathcal{C}_2^e)\Sigma\sigma_2\downarrow = u_{S_2}$. In particular since $(\Phi(\mathcal{C}_2^e), \Sigma, \sigma_2)$ weakly satisfies $\xi =_f^? \zeta$, we can write $u'_{S_2} = \zeta\Sigma\Phi(\mathcal{C}_2^e)\sigma_i\downarrow$ and have $u_{S_2} = u_{S_2'}$. Since we also have by definition (after formula normalisation, see Figure 4.1)

$$\psi{:}(\Sigma, \mathcal{C}_2^e) = (u_{S_2} =^? u_{S_2'} \Rightarrow \xi =_f^? \zeta)$$

  we obtain the expected conclusion.
- Any case-distinction rule (negative branch) or Rule (Sat): follows from $\mathsf{Inv}_{comp}(\Gamma)$.

- Rule (Rew) (positive branch): using the notations of the rule, the only case that does not directly follow from the assumption $\mathsf{Inv}_{comp}(\Gamma)$ is the case where $\psi \in \mathsf{F}_0$. The hypothesis $(\Phi(\mathcal{C}_2^e), \Sigma, \sigma)$ weakly models the head of ψ can then be rephrased as $\mathsf{msg}(\xi'\Sigma\Phi(\mathcal{C}_2^e)\sigma)$ for some recipe $\xi' = C[\xi_0]$ such that $root(C) \in \mathcal{F}_\mathsf{d}$, by definition of F_0. Let us write $u = \xi_0\Sigma\Phi(\mathcal{C}_2^e)\sigma\!\downarrow$. Since the rewriting system is constructor-destructor and $root(C[u]) \in \mathcal{F}_\mathsf{d}$, there exists at least one rewrite rule $\ell' \to r' \in \mathcal{R}$ such that $C[u]$ and ℓ' are unifiable. In particular it sufficies to choose ψ' the formula of F_0 corresponding to picking this rule in the definition of $\mathsf{RewF}(\xi, \ell \to r, p)$.
- Rule (Eq) (positive branch): easily follows from $\mathsf{Inv}_{comp}(\Gamma)$ since the rule only adds to each $(\mathcal{P}, \mathcal{C}, \mathcal{C}^e) \in \Gamma$ the same solved formula. □

Proof of preservation of Inv_{satur}. Let $\mathcal{C}' = (\Phi, \mathsf{D}, \mathsf{E}^1, \mathsf{E}^2, \mathsf{K}, \mathsf{F})$ for some $(\mathcal{P}', \mathcal{C}', \mathcal{C}^{e\prime}) \in \Gamma'$ and k such that $vars^2(\mathsf{D}(\mathcal{C})) \subseteq \mathcal{X}^2_{\leqslant k}$. We also let $\mathsf{f}/n \in \mathcal{F}_\mathsf{d}$ and $(\xi_1, u_1), \ldots, (\xi_n, u_n) \in \mathsf{Conseq}(\mathsf{K}(\mathcal{C}))$ such that $\xi_1, \ldots, \xi_n \in \mathcal{T}^2_k$, and $\mathsf{f}(u_1, \ldots, u_n)\!\downarrow$ is a constructor term. We have to prove that there is $\xi \in \mathcal{T}^2_k$ such that $(\xi, \mathsf{f}(u_1, \ldots, u_n)\!\downarrow) \in \mathsf{Conseq}(\mathsf{K}(\mathcal{C}) \cup \mathsf{D}(\mathcal{C}))$. This can be justified by $\mathsf{Inv}_{wf}(\Gamma)$ (in particular the item "shape of K"). Indeed, by definition the term $\mathsf{f}(u_1, \ldots, u_n)$ contains a single destructor, which is the symbol f at its root. In particular if $\mathsf{f}(u_1, \ldots, u_n)\!\downarrow$ is indeed a constructor protocol term u, this has been obtained after applying a single rewriting rule at the root of $\mathsf{f}(u_1, \ldots, u_n)$ since the rewriting system is constructor-destructor. Therefore, by subterm convergence, u is either a ground protocol term (without names, by definition of a rewrite rule) or a subterm of one of the u_is. In the former case the conclusion is immediate, in the latter it follows from the invariant Inv_{wf}. □

Proof of preservation of Inv_{sol}. The preservation of this invariant is straightforward: the case distinction and simplification rules only modify the extended constraint systems by 1 applying a mgs, or 2 adding formulas to E^1, E^2, K,..., or 3 removing formulas from F, or 4 removing trivially-false disequations from E^1. In particular such operations restrict the set of solutions. □

We can then extend this property to the whole procedure by proving the preservation by symbolic rules.

Lemma B.2

Let Γ be a set of extended symbolic processes such that $\mathsf{Inv}_{all}(\{\Gamma\})$. We assume that no case-distinction or simplification rules can be applied to Γ. We then let

$$\Gamma_{\mathsf{in}} = \{(\mathcal{P}', \mathcal{C}', \mathcal{C}^{e\prime}) \mid (\mathcal{P}, \mathcal{C}, \mathcal{C}^e) \in \Gamma, (\mathcal{P}, \mathcal{C}, \mathcal{C}^e) \xrightarrow{Y(X)}_{\mathsf{s}} (\mathcal{P}', \mathcal{C}', \mathcal{C}^{e\prime})\}$$

$$\Gamma_{\mathsf{out}} = \{(\mathcal{P}', \mathcal{C}', \mathcal{C}^{e\prime}) \mid (\mathcal{P}, \mathcal{C}, \mathcal{C}^e) \in \Gamma, (\mathcal{P}, \mathcal{C}, \mathcal{C}^e) \xrightarrow{\overline{Z}\langle \mathsf{ax}_{|\Phi(\mathcal{C}^e)|+1}\rangle}_{\mathsf{s}} (\mathcal{P}', \mathcal{C}', \mathcal{C}^{e\prime})\}$$

and the set of set of symbolic processes $\mathbb{S}$ obtained by normalising $\{\Gamma_{\mathsf{in}}, \Gamma_{\mathsf{out}}\}$ with the simplification rules. Then $\mathsf{Inv}_{all}(\mathbb{S})$.

Proof. Most invariants are either easily seen to be preserved by application of any symbolic or simplification rules, or are a straightforward consequence of the fact that no simplification rules can be applied to $\mathbb{S}$. The only substantial case is the Invariant 4 stating that the

knowledge base is saturated. Let us then consider $(\mathcal{P}, \mathcal{C}, \mathcal{C}^e) \in \Gamma'$ for some $\Gamma' \in \mathbb{S}$ and we let k the minimal index such that $vars^2(\mathsf{D}(\mathcal{C})) \subseteq \mathcal{X}^2_{\leqslant k}$. We then let $\mathsf{f}/n \in \mathcal{F}_\mathsf{d}$ and $(\xi_1, u_1), \ldots, (\xi_n, u_n) \in \mathsf{Conseq}(\mathsf{K}(\mathcal{C}^e))$ such that $u{\downarrow}$, with $u = \mathsf{f}(u_1, \ldots, u_n)$ is a constructor term. We recall that no case-distinction rules can be applied to $\{\Gamma\}$, in particular (Rew). But since the rewriting system is constructor-destructor, $\mathsf{f} \in \mathcal{F}_\mathsf{d}$, and $u{\downarrow}$ is a constructor term, this means that a rewrite rule is applicable at the root of u. We distinguish two cases.

▷ *case 1:* there exists a rule $\ell \to r$ applicable at the root of u, $i \in [\![1, n]\!]$ and p a position such that $\ell_{|i\cdot p} \notin \mathcal{X}^1$ and $(\xi_{i|p} \vdash^? v) \in \mathsf{K}(\mathcal{C}^e)$ for some v.

We consider the instance of Rule (Rew) with the following parameters: the position $i \cdot p$, the rule $\ell \to r$, a recipe $\xi \in \mathcal{T}(\mathcal{F}, \mathcal{X}\colon |\Phi(\mathcal{C}^e)|)$, the formula ψ_0 obtained by considering the rule $\ell \to r$ in $\mathsf{RewF}(\xi, \ell \to r, i \cdot p)$, the deduction fact $\xi_{i|p} \vdash^? v$, $\Sigma_0 = \{\xi_{|i\cdot p} \mapsto \xi_{i|p}\}$, and a mgs Σ such that the head of $\psi_0{:}(\Sigma_0\Sigma, \mathcal{C}^e{:}\Sigma)$ is of the form $\zeta \vdash^? u{\downarrow}$ for some recipe ζ. Since this instance of the rule is not applicable by hypothesis, we deduce that there already exists a solved formula $\psi \in \mathsf{F}(\mathcal{C}^e)$ of the form $\zeta' \vdash^? u{\downarrow}$. Since no simplification rules are applicable neither, in particular (Vect-add-Conseq), we deduce that there exists a recipe ξ' such that $(\xi', u{\downarrow}) \in \mathsf{Conseq}(\mathsf{K}(\mathcal{C}^e) \cup \mathsf{D}(\mathcal{C}^e))$.

▷ *case 2:* otherwise

We let $\ell \to r$ an arbitrary rewrite rule applicable at the root of u. By hypothesis for all positions $i \cdot p$ of ℓ that are not variables we know that $\xi_{i|p}$ is not the recipe of a deduction fact from $\mathsf{K}(\mathcal{C}^e)$; since $\xi_i \in \mathsf{Conseq}(\mathsf{K}(\mathcal{C}^e))$, $root(\xi_{i|p})$ is therefore a constructor symbol. We rule out the immediate case where r is a ground term and only consider the one where r is a strict subterm of ℓ. Then, since the rewriting system is constructor-destructor, we can fix a ground constructor context C such that $r = C[x_p, \ldots, x_q]$ for $x_p, \ldots, x_p$ the variables numbered p to q of ℓ (w.r.t. the lexicographic ordering on the positions of the term ℓ). In particular the recipe $(C[\xi_p, \ldots, \xi_q], u{\downarrow}) \in \mathsf{Conseq}(\mathsf{K}(\mathcal{C}^e))$. □

3 Preliminary technical results

In this section we prove some low level technical results that will be useful during the incoming proofs.

Preservation by application of substitutions In this section, we show that applying substitution preserves in some cases the different notions we use in our algorithms, namely first-order and second-order equations, deduction and equality facts; and uniformity.

Lemma B.3

Let ψ be either a deduction fact, or an equality fact or a first-order equation or a second-order equation. For all ground frame Φ, for all substitutions $\Sigma, \Sigma', \sigma, \sigma'$ if $dom(\Sigma) \cap dom(\Sigma') = \varnothing$ and $dom(\sigma) \cap dom(\sigma') = \varnothing$ then $(\Phi, \Sigma\Sigma', \sigma\sigma') \models \psi$ is equivalent to $(\Phi, \Sigma\Sigma', \sigma\sigma') \models \psi\Sigma\sigma$ and is equivalent to $(\Phi, \Sigma', \sigma') \models \psi\Sigma\sigma$.

Proof. The proof of this lemma is done by case analysis on ψ.

Case $u =^? v$*:* Consider $(\Phi, \Sigma\Sigma', \sigma\sigma') \models u =^? v$. This is equivalent to $u\sigma\sigma' = v\sigma\sigma'$. Since

$vars(\Sigma) \cap \mathcal{X}^1 = \varnothing$, we deduce that $(\Phi, \Sigma\Sigma', \sigma\sigma') \models u =^? v$ is equivalent to $u\Sigma\sigma\sigma' = v\Sigma\sigma\sigma'$. This is also equivalent to $u\Sigma\sigma\sigma\sigma' = v\Sigma\sigma\sigma\sigma'$ and so $(\Phi, \Sigma\Sigma', \sigma\sigma') \models u\Sigma\sigma =^? v\Sigma\sigma$. Note that $u\Sigma\sigma\sigma' = v\Sigma\sigma\sigma'$ is also equivalent to $(\Phi, \Sigma', \sigma') \models u\Sigma\sigma =^? v\Sigma\sigma$.

Case $\xi =^? \xi'$*:* Similar to the previous case.

Case $\xi \vdash^? u$*:* Consider $(\Phi, \Sigma\Sigma', \sigma\sigma') \models \xi \vdash^? u$. It is equivalent to $\xi\Sigma\Sigma'\Phi\downarrow = u\sigma\sigma'$ and $\mathsf{msg}(\xi\Sigma\Sigma'\Phi)$. But $(\xi \vdash^? u)\Sigma\sigma = \xi\Sigma \vdash^? u\sigma$. Moreover, by definition of substitution (in particular the acyclic property), we deduce that $\xi\Sigma\Sigma'\Phi = \xi\Sigma\Sigma\Sigma'\Phi$ and $u\sigma\sigma' = u\sigma\sigma\sigma'$. Hence, $\mathsf{msg}(\xi\Sigma\Sigma'\Phi)$ is equivalent to $\mathsf{msg}(\xi\Sigma\Sigma\Sigma'\Phi)$; and $\xi\Sigma\Sigma'\Phi\downarrow = u\sigma\sigma'$ is equivalent to $\xi\Sigma\Sigma\Sigma'\Phi\downarrow = u\sigma\sigma\sigma'$. Hence $(\Phi, \Sigma\Sigma', \sigma\sigma') \models \xi \vdash^? u$ is equivalent to $(\Phi, \Sigma\Sigma', \sigma\sigma') \models \xi \vdash^? u\Sigma\sigma$. Note that $\mathsf{msg}(\xi\Sigma\Sigma'\Phi)$ and $\xi\Sigma\Sigma'\Phi\downarrow = u\sigma\sigma'$ are also equivalent to $(\Phi, \Sigma', \sigma') \models \xi\Sigma \vdash^? u\sigma$.

Case $\xi =^?_f \xi'$*:* Similar to previous case. □

Properties on consequence of set of deduction facts This section contains some results about the consequence relation when modifying a set of deduction facts. The first lemma is about the application of substitutions.

Lemma B.4

Let S be a set of solved deduction facts. For all substitutions σ of protocol terms, for all $(\xi, t) \in \mathsf{Conseq}(S)$, $(\xi, t\sigma) \in \mathsf{Conseq}(S\sigma)$.

Proof. We know that $(\xi, t) \in \mathsf{Conseq}(S)$ implies $\xi = C[\xi_1, \ldots, \xi_n]$ and $t = C[t_1, \ldots, t_n]$ for some public context C and for all i, $(\xi_i \vdash^? t_i \in S$. Hence $(\xi_i \vdash^? t_i\sigma \in S\sigma$ which allows us to conclude. □

Proposition B.5 (transitivity of consequences)

Let S, S' be two sets of solved deduction facts. Let $\varphi = \{X_i \vdash^? u_i\}_{i=1}^n$ such that all X_i are pairwise distinct, let $(\xi, t) \in \mathsf{Conseq}(S \cup \varphi)$ and Σ, σ be two substitutions. If for all $i \in [\![1, n]\!]$, $(X_i\Sigma, u_i\sigma) \in \mathsf{Conseq}(S\Sigma\sigma \cup S')$ then $(\xi\Sigma, t\sigma) \in \mathsf{Conseq}(S\Sigma\sigma \cup S')$.

Proof. We prove this result by induction on $|\xi|$. The base case ($|\xi| = 0$) is trivial as there are no terms of size 0 and we hence focus on the inductive step. We perform a case analysis on the hypothesis $(\xi, t) \in \mathsf{Conseq}(S \cup \varphi)$.

▷ *case 1:* $\xi = t \in \mathcal{F}_0$

We directly have by definition that $(\xi\Sigma, t\sigma) \in \mathsf{Conseq}(S\Sigma\sigma \cup S')$.

▷ *case 2: there are* $\xi_1, t_1, \ldots, t_m, \xi_m$ *and* $\mathsf{f} \in \mathcal{F}_\mathsf{c}$ *such that* $\xi = \mathsf{f}(\xi_1, \ldots, \xi_m)$, $t = \mathsf{f}(t_1, \ldots, t_m)$ *and for all* $i \in [\![1, m]\!]$, $(\xi_i, t_i) \in \mathsf{Conseq}(S \cup \varphi)$

By induction hypothesis we know that for all $j \in [\![1, m]\!]$, $(\xi_j\Sigma, t_j\sigma) \in \mathsf{Conseq}(S\Sigma\sigma \cup S')$. Writing $\xi\Sigma = \mathsf{f}(\xi_1\Sigma, \ldots, \xi_m\Sigma)$ and $t\sigma = \mathsf{f}(t_1\sigma, \ldots, t_m\sigma)$, we conclude that $(\xi\Sigma, t\sigma) \in \mathsf{Conseq}(S\Sigma\sigma \cup S')$.

▷ *case 3:* $\xi \vdash^? t \in S \cup \varphi$

If $(\xi \vdash^? t) \in S$ then $(\xi\Sigma \vdash^? t\sigma) \in S\Sigma\sigma$ and the result directly holds. Otherwise $\xi \vdash^? t \in \varphi$

and hence by hypothesis $(\xi\Sigma, t\sigma) \in \mathsf{Conseq}(S\Sigma\sigma \cup S')$. □

The previous lemma showed that a consequence (ξ, t) is preserved when applying some substitution Σ, σ under the right conditions. However, it is quite strong since we ensure that $\xi\Sigma$ is consequence with $t\sigma$. In some cases, we cannot guarantee that $\xi\Sigma$ is consequence with $t\sigma$ but with some other first-order term. This is the purpose of the next lemma.

Lemma B.6

Let S, S' be two sets of solved deduction facts. Let $\varphi = \{X_i \vdash^? u_i\}_{i=1}^n$ such that all X_i are pairwise distinct. For all Σ, for all $\xi \in \mathsf{Conseq}(S \cup \varphi)$, if for all $i \in \{1, \ldots, n\}$, $X_i\Sigma \in \mathsf{Conseq}(S\Sigma \cup S')$ then $\xi\Sigma \in \mathsf{Conseq}(S\Sigma \cup S')$.

Proof. We prove this result by induction on $|\xi|$. The base case ($|\xi| = 0$) being trivial as there is no term of size 0, we focus on the inductive step.

Since ξ is consequence of $S \cup \varphi$, we know by definition that there exists t such that one of the following conditions hold:

1. $\xi = t \in \mathcal{F}_0$
2. there exists $\xi_1, t_1, \ldots, t_m, \xi_m$ and $\mathsf{f} \in \mathcal{F}_\mathsf{c}$ such that $\xi = \mathsf{f}(\xi_1, \ldots, \xi_m)$, $t = \mathsf{f}(t_1, \ldots, t_m)$ and for all $i \in \{1, \ldots, m\}$, (ξ_i, t_i) is consequence of $S \cup \varphi$.
3. there exists t such that $\xi \vdash^? t \in S \cup \varphi$.

In case 1, we directly have by definition that $(\xi\Sigma, t)$ is a consequence of $S\Sigma \cup S'$. In case 2, by our inductive hypothesis on $\xi_1, \ldots, \xi_m$, we have that for all $j \in \{1, \ldots, m\}$, $\xi_j\Sigma$ is a consequence of $S\Sigma \cup S'$ hence there exists $t'_1, \ldots, t'_m$ such that for all $j \in \{1, \ldots, m\}$, $(\xi_j\Sigma, t'_j)$ is a consequence of $S\Sigma \cup S'$. With $\xi\Sigma = \mathsf{f}(\xi_1\Sigma, \ldots, \xi_m\Sigma)$ and $t' = \mathsf{f}(t'_1, \ldots, t'_m)$, we conclude that $(\xi\Sigma, t')$ is consequence of $S\Sigma \cup S'$. In case 3, if $\xi \vdash^? t \in S$ then $\xi\Sigma \vdash^? t \in S\Sigma$ and so the result directly holds. Else $\xi \vdash^? t \in \varphi$ and so by hypothesis $\xi\Sigma \in \mathsf{Conseq}(S\Sigma \cup S')$. □

In the next lemma, we show that when a recipe is consequence of the sets of solved deduction formulas $\mathsf{K}\Sigma\sigma$ where (Σ, σ) is a solution of the constraint system, then all subterms of that recipe are also consequence of $\mathsf{K}\Sigma\sigma$. This property is in fact guaranted by the fact that K contains itself recipes consequence of itself. This is an important property that allows us to generate solutions that satisfy the uniformity property.

Lemma B.7

Let $\mathcal{C} = (\Phi, \mathsf{D}, \mathsf{E}^1, \mathsf{E}^2, \mathsf{K}, \mathsf{F})$ be an extended constraint system such that $\mathsf{Inv}_{wf}(\mathcal{C})$. For all $\xi \in \mathsf{Conseq}(\mathsf{K} \cup \mathsf{D})$, $\mathit{subterms}(\xi) \subseteq \mathsf{Conseq}(\mathsf{K} \cup \mathsf{D})$.

Proof. Since $\xi \in \mathsf{Conseq}(\mathsf{K} \cup \mathsf{D})$, we know that $\xi = C[\xi_1, \ldots, \xi_n]$ where C is a public context and $\xi_1, \ldots, \xi_n$ are recipes of deduction facts from K or D. Hence since $\xi' \in \mathit{subterms}(\xi)$, we have that the position p of ξ' in ξ is either a position of C thus $\xi' \in\in \mathsf{Conseq}(\mathsf{K} \cup \mathsf{D})$ from the definition of consequence; or is a position of one of the ξ_i and thus we conclude by the predicate $\mathsf{Inv}_{wf}(\mathcal{C})$. □

Lemma B.8

Let $\mathcal{C} = (\Phi, \mathsf{D}, \mathsf{E}^1, \mathsf{E}^2, \mathsf{K}, \mathsf{F})$ be an extended constraint system such that $\mathsf{Inv}_{wf}(\mathcal{C})$. For all $(\Sigma, \sigma) \in Sol(\mathcal{C})$, for all $\xi \in \mathsf{Conseq}(\mathsf{K}\Sigma\sigma)$, $subterms(\xi) \subseteq \mathsf{Conseq}(\mathsf{K}\Sigma\sigma)$.

Lemma B.9

Let S be a set of ground deduction facts. Let Φ be a ground frame. Assume that for all $\psi \in S$, $\Phi \models \psi$. For all $(\xi, t) \in \mathsf{Conseq}(S)$, $\Phi \models \xi \vdash^? t$.

Proof. We prove this result by induction on $|\xi|$. The base case being trivial, we focus on the inductive step. Since (ξ, t) is consequence of S then one of the following properties holds:

1. $\xi = t \in \mathcal{F}_0$
2. there exist $\xi_1, t_1, \ldots, \xi_n, t_n$ and $\mathsf{f} \in \mathcal{F}_\mathsf{c}$ such that $\xi = \mathsf{f}(\xi_1, \ldots, \xi_n)$, $t = \mathsf{f}(t_1, \ldots, t_n)$ and for all $i \in \{1, \ldots, n\}$, (ξ_i, t_i) is consequence of S
3. $\xi \vdash^? t \in S$.

In Case 1, the result trivially holds. In case two, a simple induction on $(\xi_1, t_1), \ldots, (\xi_n, t_n)$ allows us to conclude. In case 3, we know by hypothesis that $\Phi \models \xi \vdash^? t$ hence the result holds □

4 Correctness of most general solutions

In this section we prove the correctness of the constraint solving procedure for computing mgs' in Section 2. We show that given an extended constraint system $\mathcal{C}$, the Rules (MGS-Conseq), (MGS-Res), (MGS-Cons), (MGS-Unsat), and (MGS-Unif) allow to compute the most general solutions of $\mathcal{C}$.

Lemma B.10

Let $\mathcal{C}$ an extended constraint system such that $\mathsf{Inv}_{wf}(\mathcal{C})$ and $\mathsf{Inv}_{sound}(\mathcal{C})$. If any rule is applicable on $\mathcal{C}$ then for all $(\Sigma, \sigma) \in Sol(\mathcal{C})$, there exists $\mathcal{C}'$, Σ' such that $\mathcal{C} \Rightarrow\!\!\!\!\downarrow \mathcal{C}'$ and $(\Sigma\Sigma', \sigma) \in Sol(\mathcal{C}')$.

Proof. First, assume that there exist $\xi, \zeta \in R(\mathcal{C})^2$ and u such that $\xi \neq \zeta$ and $(\xi, u), (\zeta, u) \in \mathsf{Conseq}(\mathsf{K} \cup \mathsf{D})$. By $\mathsf{Inv}_{sound}(\mathcal{C})$ and Lemmas B.9 and Proposition B.5, we deduce that $\Phi\sigma \models \xi\Sigma \vdash^? u\sigma \wedge \zeta\Sigma \vdash^? u\sigma$. As such we have $\xi\Sigma{\downarrow} = u\sigma = \zeta\Sigma{\downarrow}$. However by definition of a solution it implies that $\xi\Sigma = \zeta\Sigma$. Thus there exists $\Sigma' = mgu(\xi =^? \zeta)$ such that $\Sigma' \neq \varnothing$ and $\Sigma' \neq \bot$.

In such a case, let us show that $(\Sigma, \sigma) \in Sol(\mathcal{C}')$ with $\mathcal{C} \to \mathcal{C}'$ by Rule (MGS-Conseq). We already know that $\Sigma \models \mathsf{E}^2(\mathcal{C})$ and since $\xi\Sigma = \zeta\Sigma$ with $\Sigma' = mgu(\xi =^? \zeta)$, we directly have that $\Sigma \models \mathsf{E}^2(\mathcal{C})\Sigma' \wedge \Sigma'$. Moreover, $\mathsf{K}(\mathcal{C})\Sigma = \mathsf{K}(\mathcal{C})\Sigma'\Sigma$. Hence, the two bullets of the definition of solutions is trivially satisfied by that fact that $(\Sigma, \sigma) \in Sol(\mathcal{C})$. Therefore, we conclude that $(\Sigma, \sigma) \in Sol(\mathcal{C}')$.

Let us now consider the case where our assumption do no hold. Thus since we assume that at least one rule is applicable on $\mathcal{C}$, there exists $X \vdash^? u \in \mathsf{D}(\mathcal{C})$ where $u \notin \mathcal{X}^1$. Let us do a case analysis on $X\Sigma$ since $X\Sigma \in \mathsf{Conseq}(\mathsf{K}\Sigma\sigma)$ by definition of a solution.

- either $X\Sigma \in \mathcal{F}_0$: in such a case, we have $\mathcal{C} \to \mathcal{C}'$ by Rule (MGS-Conseq) and we can prove similarly as in the previous case that $(\Sigma, \sigma) \in Sol(\mathcal{C}')$;

- or $X\Sigma = \mathsf{f}(\xi_1, \ldots, \xi_n)$ where $\xi_i \in \mathsf{Conseq}(\mathsf{K}\Sigma\sigma)$ for all i: note that we know that $X\Sigma\Phi\sigma{\downarrow}= u\sigma$. Hence $u = \mathsf{f}(u_1, \ldots, u_n)$ for some $u_1, \ldots, u_n$. We deduce that for all i, $\Phi\sigma \models \xi_i \vdash^? u_i\sigma$. Thus, by considering $\Sigma' = \{X_i \mapsto \xi_i\}_{i=1}^n$, we can conclude that $\mathcal{C} \to \mathcal{C}'$ by Rule (MGS-Cons) and $(\Sigma\Sigma', \sigma) \in Sol(\mathcal{C}')$;
- or $X\Sigma \vdash^? u\sigma \in \mathsf{K}\Sigma\sigma$ (since once again $X\Sigma\Phi\sigma{\downarrow}= u\sigma$): thus there exists $\xi \vdash^? v \in \mathsf{K}$ such that $\xi\Sigma = X\Sigma$ and $u\sigma = v\sigma$. Hence $mgu\xi X$ exists and $\sigma \models u =^? v$. Thereofore, we can conclude that $\mathcal{C} \to \mathcal{C}'$ by Rule (MGS-Res) and $(\Sigma, \sigma) \in Sol(\mathcal{C}')$. □

Lemma B.11

Let $\mathcal{C} \neq \bot$ an extended constraint system such that $\mathcal{C}\wr = \mathcal{C}$, $\mathsf{Inv}_{wf}(\mathcal{C})$ and $\mathsf{Inv}_{sound}(\mathcal{C})$. If $\mathcal{C} \not\to\wr$ and $\mathcal{C}$ is a solved extended constraint system then $mgs(\mathcal{C}) = \{mgu(\mathsf{E}^2)\}$.

Proof. We know that for all $(\Sigma, \sigma) \in Sol(\mathcal{C})$, $\Sigma \models \mathsf{E}^2(\mathcal{C})$ thus we directly obtain the existence of Σ' such that $\Sigma = mgu(\mathsf{E}^2)\Sigma'$. Consider now the second bullet point of the definition of most general solutions. We know that $\mathcal{C}$ is solved. Hence consider a fresh bijective renaming Σ_1 from $vars^2(\Sigma_0) \cup vars^2(\mathcal{C}) \setminus dom(\Sigma_0)$ to $\mathcal{F}_0$. Let us define $\sigma_1 = \{x \mapsto X\Sigma_1 \mid X \vdash^? x\mathsf{D}(\mathcal{C})\}$. Thanks to $\mathsf{Inv}_{wf}(\mathcal{C})$, $\mathsf{Inv}_{sound}(\mathcal{C})$ and Lemma B.8, B.6, and B.9 that $(\Phi mgu(\mathsf{E}^1(\mathcal{C}))\sigma_1, mgu(\mathsf{E}^2)\Sigma_1, mgu(\mathsf{E}^1(\mathcal{C}))\sigma_1) \models \mathsf{D} \wedge \mathsf{E}^1 \wedge \mathsf{E}^2$. Moreover, by Lemma B.8, we know that the first bullet of the definition of solution is satisfied. Finally, the second bullet is satisfied otherwise Rule (MGS-Unsat) would be applicable which contradict $\mathcal{C}\wr = \mathcal{C}$. Therefore, $(mgu(\mathsf{E}^2)\Sigma_1, mgu(\mathsf{E}^1(\mathcal{C}))\sigma_1) \in Sol(\mathcal{C})$. We conclude that $mgs(\mathcal{C}) = \{mgu(\mathsf{E}^2)\}$. □

Lemma B.12

Let $\mathcal{C}$ an extended constraint system such that $\mathcal{C}\wr = \mathcal{C}$, $\mathsf{Inv}_{wf}(\mathcal{C})$ and $\mathsf{Inv}_{sound}(\mathcal{C})$. If $\mathcal{C} \not\to\wr$ and $\mathcal{C}$ is not solved then $Sol(\mathcal{C}) = \varnothing$.

Proof. Since $\mathcal{C}$ is not solved, we have two possibilities: Either (a) all deduction facts in D are have variables as right hand term but not pairwise distinct. But in such a case Rule (MGS-Conseq) would be applicable which contradicts $\mathcal{C} \not\to\wr$; or (b) there exists $(X \vdash^? u) \in \mathsf{D}(\mathcal{C})$ such that $u \notin \mathcal{X}^1$. Since Rule (MGS-Conseq) is not applicable, we deduce that $u \notin \mathcal{F}_0$ and for all $\xi, \zeta \in used^2(\mathcal{C}) \setminus \{X\}$, $(\xi, u) \notin \mathsf{Conseq}(\mathsf{K} \cup \mathsf{D})$. But rule Rule (MGS-Cons) is not applicable therefore, we deduce that $u \in \mathcal{N}$.

Assume now that $Sol(\mathcal{C}) \neq \varnothing$ and so $(\Sigma, \sigma) \in Sol(\mathcal{C})$. Thus $X\Sigma\Phi{\downarrow}= u$. By definition of a solution, we know that $(X\Sigma, u) \in \mathsf{Conseq}(\mathsf{K}(\mathcal{C})\Sigma\sigma)$. Since $u \in \mathcal{N}$ it implies that there exists $(\xi \vdash^? v) \in \mathsf{K}(\mathcal{C})$ such that $X\Sigma = \xi\Sigma$ and $u = v\sigma$. Note that by $\mathsf{Inv}_{wf}(\mathcal{C})$, we also have that $v \notin \mathcal{X}^1$ and so $u = v$. In such a case, we obtain a contradiction with the fact the Rule (MGS-Res) is not applicable. □

5 Correctness of the partition tree

In this section we prove the correctness of the procedure generating the partition tree, using the invariants proved in Appendix 1. Let us first start by noticing that the case distinction rules and simplification rules preserves the first order solutions of the extended constraint systems. This property is stated in the following lemma.

Lemma B.13

Let $\mathbb{S}$ be a set of set of extended symbolic processes such that $\mathsf{Inv}_{all}(\mathbb{S})$. Let $\mathbb{S} \to \mathbb{S}'$ by applying only case distinction or simplifications rules (i.e. no symbolic transitions). Then:

- *Soundness:*
  for all $S \in \mathbb{S}$, for all $(\mathcal{P}, \mathcal{C}, \mathcal{C}^e) \in S$, for all $(\Sigma, \sigma) \in Sol(\mathcal{C}^e)$, there exist $S' \in \mathbb{S}'$, $(\mathcal{P}, \mathcal{C}, \mathcal{C}^{e\prime}) \in S'$ and $(\Sigma', \sigma') \in Sol(\mathcal{C}^{e\prime})$ such that $\sigma_{|vars^1(\mathcal{C})} = \sigma'_{|vars^1(\mathcal{C})}$
- *Completeness:*
  for all $S' \in \mathbb{S}$, for all $(\mathcal{P}, \mathcal{C}, \mathcal{C}^{e\prime}) \in S'$, for all $(\Sigma', \sigma') \in Sol(\mathcal{C}^{e\prime})$, there exist $S \in \mathbb{S}$, $(\mathcal{P}, \mathcal{C}, \mathcal{C}^e) \in S$ and $(\Sigma, \sigma) \in Sol(\mathcal{C}^e)$ such that $\Sigma_{|vars^2(\mathcal{C})} = \Sigma'_{|vars^2(\mathcal{C})}$ and $\sigma_{|vars^1(\mathcal{C})} = \sigma'_{|vars^1(\mathcal{C})}$

Proof. We do a case analysis on the rule applied.

▷ *case 1:* Normalisation rules (simplification rules of Figure 4.2)

First, let us notice the result directly hold for Rules Norm-Unif, Norm-Formula, Norm-Dupl. Indeed, Rule Norm-Diseq does not modify constraints on recipe and preserves the constraints on protocol terms. Moreover, Rule Norm-Formula,Norm-Dupl affect F which do not impact the solutions of the extended constraint system. For Rule Norm-no-MGS, since $mgs(\mathcal{C}^e) = \varnothing$, we have by definition of most general unifiers that $Sol(\mathcal{C}^e) = \varnothing$ (otherwise the first bullet of the definition is contradicted). Hence the result holds since $Sol(\bot) = \varnothing$. Similarly, Rule Norm-Formula checks whether the disequations $\forall \tilde{x}.\phi$ is trivially true meaning that the rule preserves the solutions.

▷ *case 2:* Simplification rules on partitions of extended symbolic processes (Figure 4.3)

The rule Vect-rm-Unsat only removes an extended symbolic process with an extended constraint systems having no solution hence the result holds. Rule Vect-Split splits a set of $\mathbb{S}$ into two sets thus preserving the extended symbolic processes, and Rule Vect-add-Formula only adds element in F which do not impact the solutions of a constraint system. Therefore, for all these rules, the result hold. For Rule Vect-add-Conseq however, the result is not direct since the rule adds an element in the set K which has an impact on the solutions of a constraint system. However, we know from the application condition of the rule that the head protocol terms of the deduction facts added in K_i are not consequence of $\mathsf{K}_i \cup \mathsf{D}_i$. But we also know that $\mathsf{Inv}_{satur}(\mathbb{S})$ and $\mathsf{Inv}_{wf}(\mathbb{S})$ hold hence it implies that the recipe ξ (see Figure 4.3) contains $\mathsf{ax}_{|\Phi_i|}$ and $vars^2(\mathsf{D}_i) \cap \mathcal{X}^2_{\leqslant|\Phi_i|} = \varnothing$. Hence, ξ cannot appear in the second order solutions $\mathcal{C}^e_i$ which allows us to conclude that the solutions are preserved.

▷ *case 3:* Case distinction rules

The case of case distinction rules is straightforward. Indeed, by definition all rules (Sat), (Eq) and (Rew) always refine a set of extended symbolic process Γ into Γ_1, Γ_2 where Γ_1 is obtained by applying a substitution Σ to Γ, and Γ_2 by adding the constraint $\neg\Sigma$ to Γ. In particular this refinement preserves the solutions as expected. □

Now we can show that the static equivalence is preserved by application of the case distinction and simplification rules.

Lemma B.14

Let $\mathbb{S}$ be a set of set of extended symbolic processes such that $\mathsf{Inv}_{all}(\mathbb{S})$. Let $\mathbb{S} \to \mathbb{S}'$ by applying only case distinction or simplifications rules (i.e. no symbolic transitions), $\Gamma \in \mathbb{S}$, $(\mathcal{P}_1, \mathcal{C}_1, \mathcal{C}_1^e), (\mathcal{P}_2, \mathcal{C}_2, \mathcal{C}_2^e) \in \Gamma$, $(\Sigma, \sigma_1) \in Sol(\mathcal{C}_1^e)$ and $(\Sigma, \sigma_2) \in Sol(\mathcal{C}_2^e)$ such that $\Phi(\mathcal{C}_1)\sigma_1 \sim \Phi(\mathcal{C}_2)\sigma_2$.

Proof. Once again, let us consider the potential rule applied.

▷ *case 1:* Simplification rules

The only non trivial case is Rule Vect-Split (the other ones do not refine the partition and the conclusion is therefore immediate). However by Lemma applied to $\mathbb{S}$ we know that if a deduction fact occurs in constraint systems $\mathcal{C}_1^e$ but no recipe equivalent formula can be found in the constraint system $\mathcal{C}_2^e$, then no solution of $\mathcal{C}_2^e$ can satisfy the head of the formula. Besides by $\mathsf{Inv}_{sound}(\mathbb{S})$ we also know that all solutions of $\mathcal{C}_1^e$ satisfy this deduction fact. Then since $(\Sigma, \sigma_1) \in Sol(\mathcal{C}_1^e)$, $(\Sigma, \sigma_2) \in Sol(\mathcal{C}_2^e)$ and $\Phi(\mathcal{C}_1)\sigma_1 \sim \Phi(\mathcal{C}_2)\sigma_2$, we obtain a contradiction. Therefore, $\mathcal{C}_1^e$ and $\mathcal{C}_2^e$ are necessarily in the same set of $\mathbb{S}'$.

▷ *case 2:* Case distinction rules

Note that for case distinction rules, the proof is simple since each rule create a partition of the second-order solutions with respect to some mgs Σ_0. Thus, assume w.l.o.g. that $(\Sigma', \sigma_1') \in Sol(\mathcal{C}_1^{e\prime})$.

▷ *case 2a:* negative branch of the rule

First consider that S' corresponds to branch in which we applied $\neg\Sigma_0$. In such a case, since we already know that Σ' satisfies $\neg\Sigma_0$ and no other constraint is added, we directly obtain from $(\Sigma, \sigma_2) \in Sol(\mathcal{C}_2^e)$ that $(\Sigma', \sigma_2') \in Sol(\mathcal{C}_2^{e\prime})$ (in this case, we even have $\sigma_2 = \sigma_2'$).

▷ *case 2b:* positive branch of the rule

Now consider that S' corresponds to the branch in which we applied Σ_0. In such a case, the application of Σ_0 on $\mathcal{C}_2^e$ regroups all the solution of $\mathcal{C}_2^e$ that satisfies Σ_0. Since we know that $(\Sigma, \sigma_2) \in Sol(\mathcal{C}_2^e)$ and ${\Sigma'}_{|vars^2(\mathcal{C}_1)} = \Sigma_{|vars^2(\mathcal{C}_1)}$ which implies ${\Sigma'}_{|vars^2(\mathcal{C}_2)} = \Sigma_{|vars^2(\mathcal{C}_2)}$, the result holds. □

The previous two lemmas allow us to obtain the soundness and completeness properties of the partition tree. Note that the monoticity of the second-order predicate is also proved by B.13 (Completeness part) since a solution of a child constraint system is also a solution of parent one. We now need to prove that all nodes of the partition tree are valid configurations. For that we prove properties on extended constraint systems such that no more case distinction rules are applicable.

Lemma B.15

Let $\mathbb{S}$ be a set of sets of extended symbolic processes such that $\mathsf{Inv}_{all}(\mathbb{S})$ and no instance of the rule (Sat) or normalisation rules (i.e. the simplification rules of Figure 4.2) are applicable. For all $S \in \mathbb{S}$, for all $(\mathcal{P}, \mathcal{C}, \mathcal{C}^e) \in S$, writing $\mathcal{C}^e = (\Phi, \mathsf{D}, \mathsf{E}^1, \mathsf{E}^2, \mathsf{K}, \mathsf{F})$ we have that

1. $\mathcal{C}^e$ is solved
2. all formulas $\psi \in \mathsf{F}$ are solved
3. E^1 does not contain disequations

Proof. First of all the non-applicability of Rule (SAT) case 1 gives that either $\mathcal{C}^e$ is solved or $mgs(\mathcal{C}^e) = \varnothing$; due to normalisation rules not being applicable we deduce that $mgs(\mathcal{C}^e) \neq \varnothing$ meaning that $\mathcal{C}^e$ is solved. Sinmilarly by the non applicability of case 2 we know that for all $\psi \in \mathsf{F}$, either ψ is solved or $mgs(\mathcal{C}^e[\mathsf{E}^1 \mapsto \mathsf{E}^1 \wedge \mathsf{hyp}(\psi)]) = \varnothing$. But since the normalisation rules are also not applicable, we know that $mgs(\mathcal{C}^e[\mathsf{E}^1 \mapsto \mathsf{E}^1 \wedge \mathsf{hyp}(\psi)]) \neq \varnothing$: therefore ψ is solved. Finally the non applicability of case 3 and of the normalisation rules also gives us that E^1 is only composed of syntactic equations. □

Lemma B.16

Let $\mathbb{S}$ be a set of sets of extended symbolic processes such that $\mathsf{Inv}_{all}(\mathbb{S})$ and no instance of the rule (SAT) or normalisation rules are applicable. For all $S \in \mathbb{S}$, for all $(\mathcal{P}, \mathcal{C}, \mathcal{C}^e) \in S$, $|mgs(\mathcal{C}^e)| = 1$.

Proof. Let us denote $\mathcal{C}^e = (\Phi, \mathsf{D}, \mathsf{E}^1, \mathsf{E}^2, \mathsf{K}, \mathsf{F})$. By Lemma B.15 we know that $\mathcal{C}^e$ is solved, that all formulas $\psi \in \mathsf{F}$ are solved, and that E^1 only contain equations.

▷ *Step 1:* Construction of (Σ, σ) such that $(\Phi, \Sigma, \sigma) \models \mathsf{D} \wedge \mathsf{E}^1 \wedge \mathsf{E}^2$

Since $\mathcal{C}^e$ is solved, we deduce that all deduction facts in $\mathsf{D} = \{X_i \vdash^? x_i\}_{i=1}^n$ for some n and pairwise distinct x_is and X_is. Consider now the substitutions $\Sigma_0 = \{X_i \to n_i\}_{i=1}^n$ and $\sigma_0 = \{x_i \to n_i\}_{i=1}^n$ where the n_is are pairwise distincts public names, i.e. $n_i \in \mathcal{F}_0$. Since no more normalisation rules are applicable, we know that the disequations in E^2 not trivially unsatisfiable. Therefore by replacing the free variables of the disequations by names allow us to obtain that Σ_0 the disequations of E^2. By considering $\Sigma = mgu(\mathsf{E}^2)\Sigma'$, we obtain that $\Sigma \models \mathsf{E}^2$. Moreover we proved that E^1 does not contain any disequations, we directly obtain that $mgu(\mathsf{E}^1)\sigma_0 \models \mathsf{E}^1$. Therefore, by defining $\sigma = mgu(\mathsf{E}^1)\sigma_0$, we obtain that $(\Phi, \Sigma, \sigma) \models \mathsf{D} \wedge \mathsf{E}^1 \wedge \mathsf{E}^2$.

▷ *Step 2:* Proof that (Σ, σ) is a solution

To prove that (Σ, σ) is an actual solution of $\mathcal{C}^e$ it remains to prove that it verifies the additional required two conditions: K-basis and uniformity. Let us first prove the K-basis, i.e. that for all $\xi \in subterms(img(\Sigma)) \cup subterms^2_{\neq}(\mathsf{K}\Sigma)$, $\mathsf{msg}(\xi\Phi\sigma)$ and $(\xi, \xi\Phi\sigma) \in \mathsf{Conseq}(\mathsf{K}\Sigma\sigma)$. The case $\xi \in subterms^2_{\neq}(\mathsf{K}\Sigma)$ directly follows from $\mathsf{Inv}_{wf}(\mathcal{C}^e)$. Let us therefore consider the case $\xi \in subterms(img(\Sigma))$. Since $\mathsf{Inv}_{wf}(\mathcal{C}^e)$ holds we have that $img(mgu(\mathsf{E}^2)) \subseteq \mathsf{Conseq}(\mathsf{K} \cup \mathsf{D})$; for the same reason we have that for all $\zeta \vdash^? u \in \mathsf{K}$, $subterms(\zeta) \subseteq \mathsf{Conseq}(\mathsf{K} \cup \mathsf{D})$. Therefore by applying Lemma B.6, and by a quick induction on the size of the recipe in $img(\Sigma)$, we obtain that $\xi \in \mathsf{Conseq}(\mathsf{K}\Sigma)$. Finally, by definition of consequence and since $\mathsf{Inv}_{sound}(\mathcal{C}^e)$ holds, we have $\mathsf{msg}(\xi\Phi\sigma)$ hence the K-basis.

Let us now prove uniformity. We know that $\mathcal{C}^e$ is solved which therefore means that for all recipes $\xi, \zeta \in subterms_{\mathsf{c}}(img(mgu(\mathsf{E}^2)), \mathsf{K} \cup \mathsf{D})^2 \cup (\mathcal{F}_0 \times vars^2(\mathsf{D}))$, $(\xi, u), (\zeta, u) \in \mathsf{Conseq}(\mathsf{K} \cup \mathsf{D})$ implies $\xi = \zeta$. Since $\Sigma = mgu(\mathsf{E}^2)\Sigma_0$ we directly obtain that for all $\xi, \zeta \in subterms_{\mathsf{c}}(\Sigma, \mathsf{K}\Sigma)$, $(\xi, u), (\zeta, u) \in \mathsf{Conseq}(\mathsf{K}\Sigma)$ implies $\xi = \zeta$, which is exactly the uniformity.

▷ *Step 3:* Unicity of the solution

This step is rather straightforward: considering that $\Sigma = mgu(\mathsf{E}^2)\Sigma_0$ and any other solutions $(\Sigma', \sigma') \in Sol(\mathcal{C}^e)$ satisfy $\Sigma' \models \mathsf{E}^2$, we deduce that $mgs(\mathcal{C}^e) = \{mgu(\mathsf{E}^2)\}$ and so $|mgs(\mathcal{C}^e)| = 1$. □

Let us now show that all extended constraint systems in the set have the same solutions and that they are statically equivalent.

Lemma B.17

Let $\mathbb{S}$ be a set of set of extended symbolic processes such that $\mathsf{Inv}_{all}(\mathbb{S})$ and no instances of the rules (Sat), (Eq) or (Rew) or simplification rules are applicable. For all $S \in \mathbb{S}$, for all $(\mathcal{P}_1, \mathcal{C}_1, \mathcal{C}_1^e), (\mathcal{P}_2, \mathcal{C}_2, \mathcal{C}_2^e) \in S$, if $(\Sigma, \sigma_1) \in Sol(\mathcal{C}_1^e)$ then $(\Sigma, \sigma_2) \in Sol(\mathcal{C}_2^e)$ and $\Phi(\mathcal{C}_1^e)\sigma_1 \sim \Phi(\mathcal{C}_2^e)\sigma_2$.

Proof. Since (Sat) and normalisation rules are not applicable, we know by Lemma B.15 that all extended constraint systems $\mathcal{C}^e \in S$ have a particular form, that is 1 all deduction facts in $\mathsf{D}(\mathcal{C}^e)$ have pairwise distinct variables as right hand side; and 2 $\mathsf{E}^1(\mathcal{C}^e)$ only contain syntactic equations. Moreover, we know that all extended constraint systems have the same structure. Therefore, if $(\Sigma, \sigma_1) \in Sol(\mathcal{C}_1^e)$, we deduce that $\Sigma \models \mathsf{E}^2(\mathcal{C}_1^e)$ and for all $\xi \in subterms(img(\Sigma))$, $\xi \in \mathsf{Conseq}(\mathsf{K}(\mathcal{C}_1^e)\Sigma)$, meaning that $\Sigma \models \mathsf{E}^2(\mathcal{C}_2^e)$ and $\xi \in \mathsf{Conseq}(\mathsf{K}(\mathcal{C}_2^e)\Sigma)$. Since the first order solutions are always completely defined by the second-order substitutions, we can build σ_2' such that for all $X \vdash^? x \in \mathsf{D}(\mathcal{C}_2^e)$, $X\Sigma(\Phi(\mathcal{C}_2^e)\sigma_2')\downarrow = x\sigma_2'$. Moreover, since $\mathsf{Inv}_{sound}(\mathcal{C}_1^e)$ and $\mathsf{Inv}_{sound}(\mathcal{C}_2^e)$ both hold and since for all $\xi \in subterms(img(\Sigma))$, $\xi \in \mathsf{Conseq}(\mathsf{K}(\mathcal{C}_2^e)\Sigma)$, we deduce that for all $\xi \in subterms(img(\Sigma))$, $\mathsf{msg}(\xi\Phi(\mathcal{C}_2^e)\sigma_2')$. Note that we also need to satisfy the syntactic equations in E^1. However thanks to $\mathsf{Inv}_{wf}(\mathcal{C}_2^e)$ holding, we know that $dom(mgu(\mathsf{E}^1(\mathcal{C}_2^e)) \cap vars^1(\mathsf{D}(\mathcal{C}_2^e)) = \varnothing$. Thus, we can build $\sigma_2 = mgu(\mathsf{E}^1(\mathcal{C}_2^e))\sigma_2'$ and obtain that $(\Sigma, \sigma_2) \models \mathsf{D}(\mathcal{C}_2^e) \wedge \mathsf{E}^1(\mathcal{C}_2^e) \wedge \mathsf{E}^2(\mathcal{C}_2^e)$. Note that by origination property of an extended constraint system, we have $\Phi(\mathcal{C}_2^e)\sigma_2' = \Phi(\mathcal{C}_2^e)\sigma_2$. Therefore, since we already prove that for all $\xi \in subterms(img(\Sigma))$, $\mathsf{msg}(\xi\Phi(\mathcal{C}_2^e)\sigma_2')$ and $\xi \in \mathsf{Conseq}(\mathsf{K}(\mathcal{C}_2^e)\Sigma)$, it only remains to prove the second bullet point of Definition the definition of solutions extended constraint system to obtain that $(\Sigma, \sigma_2) \in Sol(\mathcal{C}_2^e)$.

To prove this it sufficies to prove that $\Phi(\mathcal{C}_1^e)\sigma_1 \sim \Phi(\mathcal{C}_2^e)\sigma_2$: the conclusion will then follow since $(\Sigma, \sigma_1) \in Sol(\mathcal{C}_1^e)$. Therefore we let recipes ξ, ξ' and show that:

(i) $\mathsf{msg}(\xi\Phi(\mathcal{C}_1^e)\sigma_1)$ iff $\mathsf{msg}(\xi\Phi(\mathcal{C}_2^e)\sigma_2)$

(ii) if $\mathsf{msg}(\xi\Phi(\mathcal{C}_1^e)\sigma_1)$, $\xi'\Phi(\mathcal{C}_1^e)\sigma_1)$ then $\xi\Phi(\mathcal{C}_1^e)\sigma_1\downarrow = \xi'\Phi(\mathcal{C}_1^e)\sigma_1\downarrow$ iff $\xi\Phi(\mathcal{C}_2^e)\sigma_2\downarrow = \xi'\Phi(\mathcal{C}_2^e)\sigma_2\downarrow$.

We prove this by lexicographic induction on $(N(\xi, \xi'), \max(|\xi|, |\xi'|)$ where $N(\xi\,\xi')$ is the number of subterms $\zeta \in subterms(\xi, \xi')$ such that $\zeta \notin \mathsf{Conseq}(\mathsf{K}(\mathcal{C}_1^e)\Sigma)$ (recall that since $\mathcal{C}_1^e$ and $\mathcal{C}_2^e$ have the same structure, we have $\zeta \in \mathsf{Conseq}(\mathsf{K}(\mathcal{C}_1^e)\Sigma)$ iff $\zeta \in \mathsf{Conseq}(\mathsf{K}(\mathcal{C}_2^e)\Sigma)$).

▷ *case 1:* $N(\xi, \xi') = 0$ and $\max(|\xi|, |\xi'|) = 0$

Impossible since there exist no terms of size 0.

▷ *case 2:* $N(\xi, \xi') > 0$

▷ *subgoal 2a*: Proof of (i)

Assume $\mathsf{msg}(\xi\Phi(\mathcal{C}_1^e)\sigma_1)$. Let us also assume by contradiction that $\neg\mathsf{msg}(\xi\Phi(\mathcal{C}_2^e)\sigma_2)$. Since we know that $N(\xi, \xi') > 0$, there exists $\zeta \in subterms(\xi, \xi')$ such that $\zeta \notin \mathsf{Conseq}(\mathsf{K}(\mathcal{C}_1^e)$. Without loss of generality we can consider that $\zeta \in subterms(\xi)$ (otherwise we can apply our inductive hypothesis on ξ twice since $N(\xi, \xi')$ would be equal to 0 and so we would obtain a contradiction). Moreover, let us consider ζ such that $|\zeta|$ is minimal. Therefore, by definition of consequence, we deduce that $\zeta = \mathsf{g}(\zeta_1, \ldots, \zeta_n)$ with $\mathsf{g} \in \mathcal{F}_\mathsf{d}$ and for all $i \in \{1, \ldots, n\}$,

$\zeta_i \in \mathsf{Conseq}(\mathsf{K}(\mathcal{C}_1^e))$. Since $\mathsf{msg}(\xi\Phi(\mathcal{C}_1^e)\sigma_1)$ we also deduce that $\mathsf{g}(\zeta_1,\ldots,\zeta_n)\Phi(\mathcal{C}_1^e)\sigma_1{\downarrow}$ is a protocol term. Therefore, there exist a rewrite rule $\mathsf{g}(\ell_1,\ldots,\ell_n) \to r$ and a substitution γ such that $\ell_i\gamma = \zeta_i\Phi(\mathcal{C}_1^e)\sigma_1{\downarrow}$ for all $i = 1\ldots n$.

Recall that the rule (Rew) is not applicable on $\mathcal{C}_1^e$ and $\mathcal{C}_2^e$. Therefore we can show that provided $\neg\mathsf{msg}(\xi\Phi(\mathcal{C}_2^e)\sigma_2)$ and $\mathsf{g}(\zeta_1,\ldots,\zeta_n)\Phi(\mathcal{C}_1^e)\sigma_1{\downarrow}$ is a protocol term then we necessarily have that there exists $\zeta'_1,\ldots,\zeta'_n$ and u such that $\mathsf{g}(\zeta'_1,\ldots,\zeta'_n) \vdash^? u_1 \in \mathsf{F}(\mathcal{C}_1^e)$ and $\zeta'_i\Sigma\Phi(\mathcal{C}_1^e)\sigma_1{\downarrow} = \zeta_i\Phi(\mathcal{C}_1^e)\sigma_1{\downarrow}$. Moreover, since the normalisation rules are also not applicable (in particular Rule Vect-Split), we deduce that there exists u_2 such that $\mathsf{g}(\zeta'_1,\ldots,\zeta'_n) \vdash^? u_2 \in \mathsf{F}(\mathcal{C}_2^e)$. By $\mathsf{Inv}_{wf}(\mathcal{C}_1^e)$, we know that for all $i \in \{1,\ldots,n\}$, $\zeta'_i \in \mathsf{Conseq}(\mathsf{K}(\mathcal{C}_1^e)\cup\mathsf{D}(\mathcal{C}_1^e))$ and so $\zeta'_i\Sigma \in \mathsf{Conseq}(\mathsf{K}(\mathcal{C}_1^e)\Sigma)$. Moreover, by hypothesis on ζ_i, we know that $\zeta_i \in \mathsf{Conseq}(\mathsf{K}(\mathcal{C}_1^e)\Sigma)$. Thus, by applying our inductive hypothesis, we obtain that $\zeta_i\Phi(\mathcal{C}_2^e)\sigma_2{\downarrow} = \zeta'_i\Sigma\Phi(\mathcal{C}_2^e)\sigma_2{\downarrow}$. Moreover, by $\mathsf{Inv}_{sound}(\mathcal{C}_2^e)$, we know that $\mathsf{g}(\zeta'_1,\ldots,\zeta'_n)\Sigma\Phi(\mathcal{C}_2^e)\sigma_2{\downarrow} = u_2\sigma_2$ which is a protocol term. We conclude that $\mathsf{g}(\zeta_1,\ldots,\zeta_n)\Sigma\Phi(\mathcal{C}_2^e)\sigma_2{\downarrow}$ is a protocol term and thus $\mathsf{msg}(\xi\Phi(\mathcal{C}_2^e)\sigma_2)$ gives us a contradiction.

▷ *subgoal 2b*: Proof of (ii)

Assume now that $\xi\Phi(\mathcal{C}_1^e)\sigma_1{\downarrow} = \xi'\Phi(\mathcal{C}_1^e)\sigma_1{\downarrow}$, $\mathsf{msg}(\xi\Phi(\mathcal{C}_1^e)\sigma_1)$ and $\mathsf{msg}(\xi'\Phi(\mathcal{C}_1^e)\sigma_1)$. Let us once again take the smallest $\zeta \in \mathit{subterms}(\xi,\xi')$ such that $\zeta \notin \mathsf{Conseq}(\mathsf{K}(\mathcal{C}_1^e))$. We already proved above that there exist u_1, u_2, g, $\zeta'_1,\ldots,\zeta'_n,\zeta_1,\ldots,\zeta_n$ such that:

- $\zeta = \mathsf{g}(\zeta_1,\ldots,\zeta_n)$
- $\mathsf{g}(\zeta'_1,\ldots,\zeta'_n) \vdash^? u_1 \in \mathsf{F}(\mathcal{C}_1^e)$
- $\mathsf{g}(\zeta'_1,\ldots,\zeta'_n) \vdash^? u_2 \in \mathsf{F}(\mathcal{C}_2^e)$
- for all $i \in \{1,\ldots,n\}$, $\zeta_i\Phi(\mathcal{C}_2^e)\sigma_2{\downarrow} = \zeta'_i\Sigma\Phi(\mathcal{C}_2^e)\sigma_2{\downarrow}$ and $\zeta_i\Phi(\mathcal{C}_1^e)\sigma_1{\downarrow} = \zeta'_i\Sigma\Phi(\mathcal{C}_1^e)\sigma_1{\downarrow}$.

By $\mathsf{Inv}_{satur}(\mathcal{C}_1^e)$, we know that there exists β such that $(\beta,u_1) \in \mathsf{Conseq}(\mathsf{K}(\mathcal{C}_1^e)\cup\mathsf{D}(\mathcal{C}_1^e))$. However the normalisation Rule Vect-add-Formula is not applicable on the set of extended symbolic processes. Thus, we deduce that there exists β' such that $(\beta',u_1) \in \mathsf{Conseq}(\mathsf{K}(\mathcal{C}_1^e)\cup\mathsf{D}(\mathcal{C}_1^e))$ and $\mathsf{g}(\zeta'_1,\ldots,\zeta'_n) =_f^? \beta' \in \mathsf{F}(\mathcal{C}_1^e)$. Once again due to the normalisation Rule Vect-Split, we obtain that $\mathsf{g}(\zeta'_1,\ldots,\zeta'_n) =_f^? \beta' \in \mathsf{F}(\mathcal{C}_2^e)$. But $\mathsf{Inv}_{sound}(\mathcal{C}_2^e)$ and $\mathsf{Inv}_{sound}(\mathcal{C}_1^e)$ hold meaning that $(\Phi(\mathcal{C}_2^e)\sigma_2,\Sigma,\sigma_2) \models \mathsf{g}(\zeta'_1,\ldots,\zeta'_n) =_f^? \beta'$ and $(\Phi(\mathcal{C}_1^e)\sigma_1,\Sigma,\sigma_1) \models \mathsf{g}(\zeta'_1,\ldots,\zeta'_n) =_f^? \beta'$.

Note that if p is the position of ζ in ξ then we have $N(\xi[\beta'\Sigma]_p,\xi') < N(\xi,\xi')$. Thus by applying our inductive hypothesis, we obtain that $(\Phi(\mathcal{C}_2^e)\sigma_2,\Sigma,\sigma_2) \models \xi[\beta'\Sigma]_p =_f^? \xi'$. Since $(\Phi(\mathcal{C}_2^e)\sigma_2,\Sigma,\sigma_2) \models \mathsf{g}(\zeta'_1,\ldots,\zeta'_n) =_f^? \beta'$ and $(\Phi(\mathcal{C}_2^e)\sigma_2,\Sigma,\sigma_2) \models \mathsf{g}(\zeta'_1,\ldots,\zeta'_n) =_f^? \mathsf{g}(\zeta_1,\ldots,\zeta_n)$, we conclude that $(\Phi(\mathcal{C}_2^e)\sigma_2,\Sigma,\sigma_2) \models \xi =_f^? \xi'$.

▷ *case 3:* $N(\xi,\xi') = 0$ and $\max(|\xi|,\xi'|) > 0$

In such a case, we know that $\xi,\xi' \in \mathsf{Conseq}(\mathsf{K}(\mathcal{C}_1^e)\Sigma)$ and $\xi,\xi' \in \mathsf{Conseq}(\mathsf{K}(\mathcal{C}_2^e)\Sigma)$. By definition of consequence and by $\mathsf{Inv}_{sound}(\mathcal{C}_1^e)$ and $\mathsf{Inv}_{sound}(\mathcal{C}_2^e)$, we directly obtain that $\mathsf{msg}(\xi\Phi(\mathcal{C}_1^e)\sigma_1)$ and $\mathsf{msg}(\xi\Phi(\mathcal{C}_2^e)\sigma_2)$ (same thing for ξ'). Now assume that $(\Phi(\mathcal{C}_1^e)\sigma_1,\Sigma,\sigma_1) \models \xi =_f^? \xi'$. Since both ξ,ξ' are consequences of $\mathsf{K}(\mathcal{C}_1^e)\Sigma$, we deduce that:

- either $\xi = \mathsf{f}(\xi_1,\ldots,\xi_n)$ and $\xi' = \mathsf{f}(\xi'_1,\ldots,\xi'_n)$ with $\mathsf{f} \in \mathcal{F}_\mathsf{c}$ and $(\Phi(\mathcal{C}_1^e)\sigma_1,\Sigma,\sigma_1) \models \xi_i =_f^? \xi'_i$ for all i. Therefore, we can apply our inductive hypothesis on the (ξ_i,ξ'_i)s to conclude.
- or $\xi\Sigma,\xi'\Sigma \in \mathsf{K}(\mathcal{C}_1^e)\Sigma$: Since we know that the rule (Eq) is not applicable, it implies that $\xi =_f^? \xi' \in \mathsf{F}(\mathcal{C}_1^e)$ and so $\xi =_f^? \xi' \in \mathsf{F}(\mathcal{C}_2^e)$ thanks to the normalisation Rule Vect-Split.

Since $\mathsf{Inv}_{sound}(\mathcal{C}_2^e)$ holds, we can conclude that $(\Phi(\mathcal{C}_2^e)\sigma_2, \Sigma, \sigma_2) \models \xi =_f^? \xi'$.

- or $\xi\Sigma \in \mathsf{K}(\mathcal{C}_1^e)\Sigma$ and $\xi' = \mathsf{f}(\xi'_1, \ldots, \xi'_n)$ with $\mathsf{f} \in \mathcal{F}_\mathsf{c}$; Once again since the rule (Eq) is not applicable, we deduce that there exists $\zeta'_1, \ldots, \zeta'_n$ such that $\xi =_f^? \mathsf{f}(\zeta'_1, \ldots, \zeta'_n) \in \mathsf{F}(\mathcal{C}_1^e)$. Note from $\mathsf{Inv}_{sound}(\mathcal{C}_1^e)$ that in such a case, $(\Phi(\mathcal{C}_1^e)\sigma_1, \Sigma, \sigma_1) \models \xi =_f^? \mathsf{f}(\zeta'_1, \ldots, \zeta'_n)$ meaning that $(\Phi(\mathcal{C}_1^e)\sigma_1, \Sigma, \sigma_1) \models \xi'_i =_f^? \zeta'_i$ for all $i \in \{1, \ldots, n\}$. Since $|\xi'_i\Phi(\mathcal{C}_1^e)\sigma_1\downarrow| < |\xi\Phi(\mathcal{C}_1^e)\sigma_1\downarrow|$, we can apply our inductive hypothesis on all (ξ'_i, ζ'_i) meaning that $(\Phi(\mathcal{C}_2^e)\sigma_2, \Sigma, \sigma_2) \models \mathsf{f}(\zeta'_1, \ldots, \zeta'_n) =_f^? \xi'$. However, by the rule Vect-Split not being applicable, $\xi =_f^? \mathsf{f}(\zeta_1, \ldots, \zeta'_n) \in \mathsf{F}(\mathcal{C}_1^e)$ implies $\xi =_f^? \mathsf{f}(\zeta'_1, \ldots, \zeta'_n) \in \mathsf{F}(\mathcal{C}_2^e)$ and so by $\mathsf{Inv}_{sound}(\mathcal{C}_2^e)$, we obtain that $(\Phi(\mathcal{C}_2^e)\sigma_2, \Sigma, \sigma_2) \models \xi =_f^? \mathsf{f}(\zeta'_1, \ldots, \zeta'_n)$ which allows us to conclude that $(\Phi(\mathcal{C}_2^e)\sigma_2, \Sigma, \sigma_2) \models \xi =_f^? \xi'$. □

Appendix C:

Proofs of Chapter 5

Summary.
In this appendix we prove the correctness of the partial order reductions and reductions by symmetry introduced in Chapter 5 to optimise the decision of equivalence by session. For simplicity of the notations, all along this appendix, the conditions on skeleton equivalence are written as skel equalities.

1 Explicit session matchings

In this section we present an alternative characterisation of equivalence by session. The process matchings modelled by twin processes are here represented by an explicit permutation with properties mirroring the structure of twin processes. This formalisation makes closer link with the definition of trace equivalence and is more convenient convenient for some proofs (see e.g. Appendices 2 and 4). Besides, note that this is the characterisation we use in the implementation, fitting better to the existing procedure of the DeepSec prover for trace equivalence.

Session matchings We first characterise the condition under which, given two traces t, t', there exists t^2 such that $\mathsf{fst}(t^2) = t$ and $\mathsf{snd}(t^2) = t'$. For that we rely on the notion of *labels* introduced in Section 3.1 to make reference to subprocess positions. In the rest of the paragraph, we refer to two labelled extended processes A_0, B_0 such that $\mathsf{skel}(A_0) = \mathsf{skel}(B_0)$, and as usual we assume that neither of them contains two labels such that one is the prefix of the other. We also let $t \in \mathbb{T}(P), t' \in \mathbb{T}(Q)$

$$t : A_0 \xrightarrow{[a_1]^{\ell_1}} \cdots \xrightarrow{[a_n]^{\ell_n}} A_n \qquad\qquad t' : B_0 \xrightarrow{[b_1]^{\ell'_1}} \cdots \xrightarrow{[b_n]^{\ell'_n}} B_n$$

We write L and L' the sets of labels appearing in t and t'.

Definition C.1

A *session matching* for t and t' is a bijection $\pi : L \to L'$ verifying the following properties

1. for all $i \in [\![0, n]\!]$, π defines a bijection from the labels of A_i to the labels of B_i. Besides if A_i and B_i contain processes $[\,P\,]^{\ell}$ and $[\,Q\,]^{\pi(\ell)}$ respectively then $\mathsf{skel}(P) = \mathsf{skel}(Q)$
2. for all $i \in [\![1, n]\!]$, $\pi(\ell_i) = \ell'_i$

3 $\forall \ell \cdot p \in dom(\pi), \exists q, \pi(\ell \cdot p) = \pi(\ell) \cdot q$

Proposition C.1

The following two points are equivalent:

1 $actions(t) = actions(t')$ (labels removed) and there exists a session matching for t and t'.

2 $\exists t^2,\ \mathsf{fst}(t^2) = t$ and $\mathsf{snd}(t^2) = t'$.

Besides in the i^{th} extended twin process of t^2, the process labelled ℓ in A_i is paired with the process labelled $\pi(\ell)$ in B_i.

Proof of 1 ⇒ 2. The trace t^2 can be easily constructed by induction on the length of t:

- Item 1 of Definition C.1 ensure that the twin processes in t^2 are composed of pairs of processes with the same skeleton,
- Item 2 ensures that pairs of transitions of P and Q can be mapped into transitions of twin-processes, and
- The permutations that are required by applications of the rule (Match) can be inferred from Item 3. Indeed, consider two instances of the rule (Par) in t and t':

$$(\{\!\{[\,P_1 \mid \ldots \mid P_n\,]^{\ell}\}\!\} \cup \mathcal{P}, \Phi) \xrightarrow{\tau} (\{\!\{[\,P_i\,]^{\ell \cdot i}\}\!\}_{i=1}^{n} \cup \mathcal{P}, \Phi)$$
$$(\{\!\{[\,Q_1 \mid \ldots \mid Q_n\,]^{\ell'}\}\!\} \cup \mathcal{Q}, \Psi) \xrightarrow{\tau} (\{\!\{[\,Q_i\,]^{\ell' \cdot i}\}\!\}_{i=1}^{n} \cup \mathcal{Q}, \Psi)$$

Given π a session matching for t and t', we consider the permutation of $[\![1, n]\!]$ mapping $i \in [\![1, n]\!]$ to the (unique) j such that $\pi(\ell \cdot p) = \ell' \cdot j$. This permutation can be used to construct the instance of rule (Match) corresponding to these two (Par) transitions. □

Proof of 2 ⇒ 1. Let t^2 be a trace given by Item 2. We lift the labellings of $t = \mathsf{fst}(t^2)$ and $t' = \mathsf{snd}(t^2)$ to the twin processes appearing in t^2. Then as explicited in the theorem statement it sufficies to consider π the mapping from L to L' such that $\pi(\ell) = \ell'$ for all twin process $([\,P\,]^{\ell}, [\,Q\,]^{\ell'})$ appearing in t^2. A quick induction on the length of t^2 shows that π is well defined and is a session matching for t and t'. We recall in particular the invariant that each A_i or B_i only contains labels that are incomparable w.r.t. the prefix ordering. □

Important properties As a direct corollary, we give an alternative characterisation of equivalence by session.

Proposition C.2

Let P, Q be plain processes in $\rightsquigarrow$-normal form such that $\mathsf{skel}(P) = \mathsf{skel}(Q)$. The following points are equivalent:

1 $P \sqsubseteq_s Q$

2 for all $t \in \mathbb{T}(P)$, there exist $t' \in \mathbb{T}(Q)$ and a session matching for t and t' such that $t \sim t'$ (*note:* in "$t \sim t'$" the comparison of $actions(t)$ and $actions(t')$ does not take the labels into account)

Besides the relation $\sim_s$ on traces defined by $t \sim_s t'$ *iff* there exists t^2 such that $\mathsf{fst}(t^2) = t$ and $\mathsf{snd}(t^2) = t'$, is an equivalence relation. More precisely:

Proposition C.3

For all traces t_1, t_2, t_3:

1. id is a session matching for t_1 and t_1
2. if π is a session matching for t_1 and t_2 then π^{-1} is a session matching for t_2 and t_1
3. if π is a session matching for t_1 and t_2, and π' for t_2 and t_3, then $\pi' \circ \pi$ is a session matching for t_1 and t_3

2 False attacks and determinacy

In this section we give a detailed proof of the claim of Section 2.2 that false attacks cannot arise for determinate processes, i.e.:

Proposition 5.3 (completeness of equivalence by session for determinate processes)

If P, Q are determinate plain processes such that $P \approx_t Q$ then $P \approx_s Q$.

In the proof, by slight abuse of notation, we may say that an extended process is determinate. We also cast the notion of skeleton to extended processes by writing

$$\mathsf{skel}((\mathcal{P}, \Phi)) = \mathsf{skel}(\mathcal{P}) = \bigcup_{P \in \mathcal{P}} \mathsf{skel}(P)\,,$$

and to traces with

$$\mathsf{skel}(A_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} A_n) = \mathsf{skel}(A_0) \cdot \mathsf{skel}(A_1) \cdot \ldots \cdot \mathsf{skel}(A_n)\,.$$

That is, the skeleton of a trace is the sequence of the skeletons of the processes of which it is composed. Thus, if

$$t : A_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} A_n \qquad\qquad t' : B_0 \xrightarrow{\beta_1} \cdots \xrightarrow{\beta_p} B_p$$

we have $\mathsf{skel}(t) = \mathsf{skel}(t')$ *iff* $n = p$ and for all $i \in [\![0, n]\!]$, $\mathsf{skel}(A_i) = \mathsf{skel}(B_i)$.

Simplifying equivalence First we simplify the problem by forcing the application of (Par) rules in priority in traces.

Definition C.2

If P is a plain process in $\rightsquigarrow$-normal form, we write $\mathbb{T}_\tau(P)$ the set of traces where the rule (Par) is always performed in priority, i.e. where the rules (In) and (Out) are never applied to extended processes $(\mathcal{P}, \Phi)$ such that $\mathcal{P}$ contains a process with a parallel a its root (i.e. a process P such that $|\mathsf{skel}(P)| > 1$).

Proposition C.4

If P, Q are plain processes in $\rightsquigarrow$-normal form such that $\mathsf{skel}(P) = \mathsf{skel}(Q)$:

- $P \sqsubseteq_t Q$ *iff* $\forall t \in \mathbb{T}_\tau(P), \exists t' \in \mathbb{T}_\tau(Q), t \sim t'$
- $P \sqsubseteq_s Q$ *iff* $\forall t \in \mathbb{T}_\tau(P), \exists t^2 \in \mathbb{T}(P, Q), t = \mathsf{fst}(t^2) \sim \mathsf{snd}(t^2)$

Proof. The first point is standard. The proof of the second point can be seen as a corollary of the compression optimisations of equivalence by session (see Section 3.2). □

Definition C.3

An extended process $A = (\{\!\{P_1, \dots, P_n\}\!\}, \Phi)$ is *τ-deterministic* if there is at most one $i \in [\![1, n]\!]$ such that P_i has a parallel operator at its root (i.e. $|\mathsf{skel}(P_i)| > 1$).

The τ-determinism will be an invariant in proofs by induction on the length of traces. More precisely, if A, B are extended processes we call $\mathsf{Inv}(A, B)$ the property stating

(i) A_i, B_i are determinate
(ii) $\mathsf{skel}(A_i) = \mathsf{skel}(B_i)$
(iii) $A_i \sim B_i$
(iv) A_i, B_i are τ-deterministic, and A_i contains a process with a parallel operator at its root (i.e. a process P_i such that $|\mathsf{skel}(P_i)| > 1$) *iff* B_i does.

Equivalence and inclusion We prove that trace equivalence coincides with a notion of trace inclusion strengthened with identical actions and skeleton checks.

Proposition C.5

If P, Q are determinate plain processes in $\rightsquigarrow$-normal form s.t. $\mathsf{skel}(P) = \mathsf{skel}(Q)$, then the following points are equivalent

1 $P \approx_t Q$

2 $\forall t \in \mathbb{T}_\tau(P),\ \exists t' \in \mathbb{T}_\tau(Q),\ \begin{cases} actions(t) = actions(t') \\ \Phi(t) \sim \Phi(t') \\ \mathsf{skel}(t) = \mathsf{skel}(t') \end{cases}$

Proof of 2 ⇒ 1. Given A, B two determinate extended processes we write $\varphi(A, B)$ the property stating that

$$\forall t \in \mathbb{T}_\tau(A),\ \exists t' \in \mathbb{T}_\tau(B),\ \begin{cases} actions(t) = actions(t') \\ \Phi(t) \sim \Phi(t') \\ \mathsf{skel}(t) = \mathsf{skel}(t') \end{cases} .$$

Note that $\varphi(A, B)$ implies $\mathsf{skel}(A) = \mathsf{skel}(B)$ by choosing the empty trace. In particular, to prove 2 ⇒ 1, it sufficies to prove that for all A, B determinate, $\varphi(A, B) \Rightarrow A \sqsubseteq_t B$ and $\varphi(A, B) \Rightarrow \varphi(B, A)$.

The first implication is immediate. As for the second, we prove that for all extended processes A_0, B_0 such that $\varphi(A_0, B_0)$ and $\mathsf{Inv}(A_0, B_0)$, and all

$$t' : B_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} B_n \quad \in \mathbb{T}_\tau(B_0)\,,$$

there exists

$$t : A_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} A_n \quad \in \mathbb{T}_\tau(A_0)\,,$$

s.t. for all $i \in [\![0, n]\!]$, $\mathsf{Inv}(A_i, B_i)$. This is sufficient to conclude as $\mathsf{Inv}(P, Q)$ holds for any determinate plain processes P, Q in $\rightsquigarrow$-normal form s.t. $\mathsf{skel}(P) = \mathsf{skel}(Q)$.

We proceed by induction on n. If $n = 0$ the conclusion is immediate. Otherwise, assume by induction hypothesis that it holds for any trace of length $n - 1$.

▷ *case 1:* $\alpha_1 = \tau$.

We know that B_0 does not contain private channels by determinacy ($\mathsf{Inv}(A_0, B_0)$ Item (i)). Therefore, the transition $B_0 \xrightarrow{\tau} B_1$ is derived by the rule (Par). In particular by $\mathsf{Inv}(A_0, B_0)$ Item (iv), there also exists a transition $A_0 \xrightarrow{\tau} A_1$. The conclusion can now follow from the induction hypothesis applied to A_1, B_1; but to apply it we have to prove that $\varphi(A_1, B_1)$ and $\mathsf{Inv}(A_1, B_1)$ hold.

→ *proof that* $\varphi(A_1, B_1)$.

Let $s \in \mathbb{T}_\tau(A_1)$. Then $(A_0 \xrightarrow{\tau} A_1) \cdot s \in \mathbb{T}_\tau(A_0)$ and by $\varphi(A_0, B_0)$ there exists $(B_0 \xrightarrow{\tau} B_1') \cdot s' \in \mathbb{T}_\tau(B_0)$ such that $\mathit{actions}(s) = \mathit{actions}(s')$, $\Phi(t) \sim \Phi(t')$ and $\mathsf{skel}(t) = \mathsf{skel}(t')$. But by τ-determinism of B_0 we deduce that $B_1 = B_1'$, and $s' \in \mathbb{T}_\tau(B_1)$ satisfies the expected requirements.

→ *proof that* $\mathsf{Inv}(A_1, B_1)$.

(i) A_0 and B_0 are determinate and determinacy is preserved by transitions.
(ii) $\mathsf{skel}(A_1) = \mathsf{skel}(A_0) = \mathsf{skel}(B_0) = \mathsf{skel}(B_1)$
(iii) $A_0 \sim B_0$ and the rule (Par) does not affect the frame.
(iv) A_0 and B_0 are τ-deterministic and τ-determinism is preserved by transitions (w.r.t. $\mathbb{T}_\tau$). Besides due to the $\rightsquigarrow$-normalisation, we know that neither of A_1 nor B_1 contain a parall operator, hence the result.

▷ *case 2:* $\alpha_1 \neq \tau$.

By definition $\mathbb{T}_\tau(B_0)$, we know that the rule (Par) is not applicable to B_0; neither to A_0 by $\mathsf{Inv}(A_0, B_0)$ Item (iv), which means that traces of $\mathbb{T}_\tau(B_0)$ may start by an application of rules (In) or (Out). Using this and the fact that $\mathsf{skel}(A_0) = \mathsf{skel}(B_0)$ ($\mathsf{Inv}(A_0, B_0)$ Item (ii)), we obtain that there exists a transition $A_0 \xrightarrow{\alpha_1} A_1$. The conclusion can now follow from the induction hypothesis applied to A_1, B_1; but to apply it we have to prove that $\varphi(A_1, B_1)$ and $\mathsf{Inv}(A_1, B_1)$ hold.

→ *proof that* $\varphi(A_1, B_1)$.

The argument is the same as its analogue in *case 1*, using the determinacy of B_0 instead of its τ-determinism.

→ *proof that* $\mathsf{Inv}(A_1, B_1)$.

(i) A_0 and B_0 are determinate and determinacy is preserved by transitions.
(ii) By applying $\varphi(A_0, B_0)$ with the trace $t_0 : A_0 \xrightarrow{\alpha_1} A_1$, we obtain a trace $t_0' : B_0 \xrightarrow{\alpha_1} B_1'$ such that $\mathsf{skel}(A_1) = \mathsf{skel}(B_1')$. But by determinacy of B_0, the transition $B_0 \xrightarrow{\alpha_1} B_1$ is the only transition from B_0 that has label α_1, hence $B_1 = B_1'$ and the conclusion.
(iii) Identical proof as that of Item (ii) above, using the fact that $A_1 \sim B_1'$ instead of $\mathsf{skel}(A_1) = \mathsf{skel}(B_1')$.
(iv) Let us write

$$A_0 = (\{\!\{P_0\}\!\} \cup \mathcal{P}, \Phi) \qquad A_1 = (\{\!\{P_1\}\!\} \cup \mathcal{P}, \Phi')$$
$$B_0 = (\{\!\{Q_0\}\!\} \cup \mathcal{Q}, \Psi) \qquad B_1 = (\{\!\{Q_1\}\!\} \cup \mathcal{Q}, \Psi')$$

As we argued already at the beginning of *case 2*, neither $\mathcal{P}$ nor $\mathcal{Q}$ contain processes with parallel operators at their roots. Therefore, we only have to prove that P_1 has a parallel operator at its root *iff* Q_1 does. For cardinality reasons, this a direct corollary of the following points:
– $\mathsf{skel}(P_0) = \mathsf{skel}(Q_0)$ (same action α_1 being executable at topelevel),
– $\mathsf{skel}(A_0) = \mathsf{skel}(B_0)$ (hypothesis $\mathsf{Inv}(A_0, B_0)$), and
– $\mathsf{skel}(A_1) = \mathsf{skel}(B_1)$ (Item (ii) proved above). □

Proof of 1 ⇒ 2. The proof will follow in the steps as the other implication (we construct the trace t' by induction on the length of t while maintaining the invariant Inv).

More formally, we prove that for all extended processes A_0, B_0 such that $A_0 \approx_t B_0$ and $\mathsf{Inv}(A_0, B_0)$, and all

$$t : A_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} A_n \quad \in \mathbb{T}_\tau(A_0)\,,$$

there exists

$$t' : B_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} B_n \quad \in \mathbb{T}_\tau(B_0)\,,$$

s.t. for all $i \in [\![0, n]\!]$, $\mathsf{Inv}(A_i, B_i)$.

We proceed by induction on n. We proceed by induction on n. If $n = 0$ the conclusion is immediate. Otherwise, assume by induction hypothesis that it holds for any trace of length $n - 1$.

▷ *case 1:* $\alpha_1 = \tau$.

Similarly to the converse implication, there exists a transition $B_0 \xrightarrow{\tau} B_1$ (derived by (Par)) and it sufficies to prove that $A_1 \approx_t B_1$ and $\mathsf{Inv}(A_1, B_1)$ hold in order to apply the induction hypothesis and conclude.

→ *proof that* $A_1 \approx_t B_1$.

Let $s \in \mathbb{T}_\tau(A_1)$. Then $(A_0 \xrightarrow{\tau} A_1) \cdot s \in \mathbb{T}_\tau(A_0)$ and since $A_0 \approx_t B_0$ there is $(B_0 \xrightarrow{\tau} B_1') \cdot s' \in \mathbb{T}_\tau(B_0)$ such that

$$(A_0 \xrightarrow{\tau} A_1) \cdot s \sim (B_0 \xrightarrow{\tau} B_1') \cdot s'\,.$$

But by τ-determinism of B_0 we deduce that $B_1 = B_1'$, and thus $s' \in \mathbb{T}_\tau(B_1)$ and $s \sim s'$. This justifies that $A_1 \sqsubseteq_t B_1$, and a symmetric argument can be used for the converse inclusion $B_1 \sqsubseteq_t A_1$.

→ *proof that* $\mathsf{Inv}(A_1, B_1)$.

By the exact same arguments as that of the analogue case in the converse implication.

▷ *case 2:* $\alpha_1 \neq \tau$.

Similarly to the converse implication, there exists a transition $B_0 \xrightarrow{\alpha_1} B_1$ and it sufficies to prove that $A_1 \approx_t B_1$ and $\mathsf{Inv}(A_1, B_1)$ hold in order to apply the induction hypothesis and conclude.

→ *proof that* $A_1 \approx_t B_1$.

The argument is the same as its analogue in *case 1*, using the determinacy of B_0 instead of its τ-determinism.

$\rightarrow$ *proof that* $\mathsf{Inv}(A_1, B_1)$.

This is the proof obligation whose arguments substantially differ from that of the converse implication.

(i) A_0 and B_0 are determinate and determinacy is preserved by transitions.

(ii) We assume by contradiction that $\mathsf{skel}(A_1) \neq \mathsf{skel}(B_1)$. By symmetry, say that $\mathsf{skel}(A_1) \not\subseteq \mathsf{skel}(B_1)$ and let $s \in \mathsf{skel}(A_1) \setminus \mathsf{skel}(B_1)$. By definition of $\mathbb{T}_\tau(A_0)$, we know that the rule (Par) is neither applicable to A_0 nor B_0; in particular, there exists a transition $A_1 \xrightarrow{\alpha} A$ derived from rule (In) or (Out) (the one corresponding to the skeleton s) such that $B_1 \not\xrightarrow{\alpha}$. But by determinacy of B_0, the transition $B_0 \xrightarrow{\alpha_1} B_1$ is the only transition from B_0 that has label α_1. Thus, this yields a contradiction with $A_0 \approx_t B_0$: more precisely the trace $A_0 \xrightarrow{\alpha_1} A_1 \xrightarrow{\alpha} A$ is not matched.

(iii) By determinacy of B_0, the transition $t'_0 : B_0 \xrightarrow{\alpha_1} B_1$ is the only transition from B_0 that has label α_1. In particular, using the hypothesis $A_0 \approx_t B_0$, we obtain that $t'_0 \in \mathbb{T}_\tau(B_0)$ is the only trace such that

$$t_0 : (A_0 \xrightarrow{\alpha_1} A_1) \sim t'_0 \,.$$

In particular $A_1 \sim B_1$.

(iv) Same cardinality argument as the analogue case in the converse implication. □

Session matchings Prop C.5 is the core result of the proof. We now connect it with the equivalence by session by using the characterisation of Appendix 1.

Proposition C.6

Let P, Q two determinate plain processes in $\rightsquigarrow$-normal form and two labelled traces $t \in \mathbb{T}_\tau(A_0)$ and $t' \in \mathbb{T}_\tau(Q)$ such that $\mathit{actions}(t) = \mathit{actions}(t')$ and $\mathsf{skel}(t) = \mathsf{skel}(t')$. Then there exists a session matching for t and t'.

Proof. We prove that for all τ-deterministic, determinate extended processes A_0 and B_0, and

$$t : A_0 \xrightarrow{[\alpha_1]^{\ell_1}} \cdots \xrightarrow{[\alpha_n]^{\ell_n}} A_n \qquad\qquad t' : B_0 \xrightarrow{[\alpha_1]^{\ell'_1}} \cdots \xrightarrow{[\alpha_n]^{\ell'_n}} B_n$$

if $\mathsf{skel}(t) = \mathsf{skel}(t')$, then there exists a session matching for t and t'. We proceed by induction on n. If $n = 0$ the session matching is $\pi : \varepsilon \mapsto \varepsilon$. Otherwise, let us write

$$A_{n-1} = (\{\!\{ [\, P \,]^{\ell_n} \}\!\} \cup \mathcal{P}, \Phi) \qquad\qquad B_{n-1} = (\{\!\{ [\, Q \,]^{\ell'_n} \}\!\} \cup \mathcal{Q}, \Psi)$$

By induction hypothesis, let π be a session matching for the first $n-1$ transitions of t and t'; in particular, the labels of A_{n-1} are in the domain of π.

▷ *case 1:* $\alpha_n \neq \tau$.

In this case we write

$$A_n = (\{\!\{ [\, P' \,]^{\ell_n} \}\!\} \cup \mathcal{P}, \Phi') \qquad\qquad B_n = (\{\!\{ [\, Q' \,]^{\ell'_n} \}\!\} \cup \mathcal{Q}, \Psi')$$

First of all, we observe that $\mathsf{skel}(P) = \mathsf{skel}(Q)$ because the same observable action α_n can be performed at the root of P and Q. In particular, by determinacy (hypothesis), unicity of

the process with a given label (invariant of the labelling procedure), and Item 1 of Definition C.1, we deduce that $\pi(\ell_n) = \ell'_n$.

Therefore by the hypothesis $\mathsf{skel}(A_{n-1}) = \mathsf{skel}(B_{n-1})$, we obtain $\mathsf{skel}(\mathcal{P}) = \mathsf{skel}(\mathcal{Q})$. Hence $\mathsf{skel}(P') = \mathsf{skel}(Q')$ by the hypothesis $\mathsf{skel}(A_n) = \mathsf{skel}(B_n)$. All in all, π is a session matching for the whole traces t and t'.

▷ *case 2:* $\alpha_n = \tau$.

In this case we write

$$P = P_1 \mid \cdots \mid P_k \qquad A_n = (\{\!\{ [\, P_i \,]^{\ell_n \cdot i} \}\!\}_{i=1}^{k} \cup \mathcal{P}, \Phi')$$
$$Q = Q_1 \mid \cdots \mid Q_{k'} \qquad B_n = (\{\!\{ [\, Q_i \,]^{\ell'_n \cdot i} \}\!\}_{i=1}^{k'} \cup \mathcal{Q}, \Psi')$$

Since determinacy excludes private channels, the last transition of t and t' is derived from the rule (Par). By τ-determinism, this means that P and Q are the only processes in A_{n-1} and B_{n-1}, respectively, that contain a parallel operator at their roots. In particular, by Item 1 of Definition C.1, we deduce that $\pi(\ell_n) = \ell'_n$ and $\mathsf{skel}(P) = \mathsf{skel}(Q)$; and thus $k = k'$.

Therefore, there exists a permutation σ of $[\![1, k]\!]$ such that for all $i \in [\![1, k]\!]$, $\mathsf{skel}(P_i) = \mathsf{skel}(Q_{\sigma(i)})$ (although this is not needed for the proof, this permutation appears to be unique by determinacy). Thus if $\pi' : L \to L'$ is the function extending π and such that

$$\forall i \in [\![1, k]\!], \pi'(\ell \cdot i) = \pi(\ell) \cdot \sigma(i)\,,$$

then π' is a session matching for t and t'. □

Altogether Prop C.2, C.5, C.6 justify the following corollary (that actually appears to be stronger than Prop 5.3).

Corollary C.7

If P and Q are determinate plain processes in $\rightsquigarrow$-normal form, $P \approx_t Q$ *iff* $P \sqsubseteq_s Q$.

3 Correctness of POR

In this section we prove the results presented in Section 3.

3.1 Permutability of independent actions

We give the proof of the core correctness argument, namely that traces can be considered up to permutation of independent actions (Proposition 5.15). First we prove it for traces of two actions.

Proposition C.8

If $\alpha \parallel \beta$ and $t : A \overset{\alpha\beta}{\Longrightarrow} B$, then there exists a trace $u : A \overset{\beta\alpha}{\Longrightarrow} B$. It has the property that for all traces $u^2 : A^2 \overset{\beta\alpha}{\Longrightarrow}_{\mathsf{ss}^2} B^2$ such that $\mathsf{fst}(u^2) = u$, there exists $t^2 : A^2 \overset{\alpha\beta}{\Longrightarrow}_{\mathsf{ss}^2} B^2$ such that $\mathsf{fst}(t^2) = t$.

Proof. Since the labels of α and β are incomparable w.r.t. the prefix ordering by independence, the trace t needs have the form

$$A = (\mathcal{P} \cup \mathcal{Q} \cup \mathcal{R}, \Phi) \xrightarrow{\alpha} (\mathcal{P}' \cup \mathcal{Q} \cup \mathcal{R}, \Phi') \xrightarrow{\beta}_{\mathsf{ss}^2} (\mathcal{P}' \cup \mathcal{Q}' \cup \mathcal{R}, \Phi'')$$

with $(\mathcal{P}, \Phi) \xrightarrow{\alpha} (\mathcal{P}', \Phi')$ and $(\mathcal{Q}, \Phi') \xrightarrow{\beta} (\mathcal{Q}', \Phi'')$. Now we construct the trace u, by a case analysis on α and β. In each case, we omit the construction of the trace t^2 that can be inferred easily.

▷ *case 1:* α and β are inputs or τ actions.

In particular $\Phi'' = \Phi' = \Phi$ and it sufficies to choose

$$u : (\mathcal{P} \cup \mathcal{Q} \cup \mathcal{R}, \Phi) \xrightarrow{\beta} (\mathcal{P} \cup \mathcal{Q}' \cup \mathcal{R}, \Phi) \xrightarrow{\alpha} (\mathcal{P}' \cup \mathcal{Q}' \cup \mathcal{R}, \Phi)\,.$$

▷ *case 2:* α is an output and β is an input or a τ action.

In particular $\Phi'' = \Phi' = \Phi \cup \{\mathsf{ax} \mapsto m\}$ with $\mathsf{ax} \notin dom(\Phi)$ and ax does not appear in β. Then it sufficies to choose the trace

$$u : (\mathcal{P} \cup \mathcal{Q} \cup \mathcal{R}, \Phi) \xrightarrow{\beta} (\mathcal{P} \cup \mathcal{Q}' \cup \mathcal{R}, \Phi) \xrightarrow{\alpha} (\mathcal{P}' \cup \mathcal{Q}' \cup \mathcal{R}, \Phi')\,.$$

▷ *case 3:* α is an input or a τ action and β is an output.

Similar to case 2.

▷ *case 4*: α and β are both outputs.

Then $\Phi' = \Phi \cup \{\mathsf{ax} \mapsto m\}$ and $\Phi'' = \Phi' \cup \{\mathsf{ax}' \mapsto m'\}$ with $\mathsf{ax} \neq \mathsf{ax}'$, $\{\mathsf{ax}, \mathsf{ax}'\} \cap dom(\Phi) = \varnothing$. Then we choose

$$\begin{aligned} u : (\mathcal{P} \cup \mathcal{Q} \cup \mathcal{R}, \Phi) &\xrightarrow{\beta} (\mathcal{P} \cup \mathcal{Q}' \cup \mathcal{R}, \Phi \cup \{\mathsf{ax}' \mapsto m'\}) \\ &\xrightarrow{\alpha} (\mathcal{P}' \cup \mathcal{Q}' \cup \mathcal{R}, \Phi'')\,. \end{aligned}$$

□

Then Proposition 5.15 can be obtained by induction on the hypothesis of π permuting independent actions of tr, using Proposition C.8. We actually prove the stronger result:

Proposition C.9

If $t : A \overset{\mathsf{tr}}{\Rightarrow} B$ and π permutes independent actions of tr, then $A \overset{\pi.\mathsf{tr}}{\Longrightarrow} B$. This trace is unique if we take labels into account, and is referred as $\pi.t$. It has the property that for all $u^2 : A^2 \overset{\pi.\mathsf{tr}}{\Longrightarrow}_{\mathsf{ss}^2} B^2$ such that $\mathsf{fst}(u^2) = \pi.t$, there exists $t^2 : A^2 \overset{\mathsf{tr}}{\Rightarrow}_{\mathsf{ss}^2} B^2$ such that $\mathsf{fst}(t^2) = t$.

Proof. The uniqueness of $\pi.t$ is immediate, as a quick induction on the length of traces shows that any labelled trace u is uniquely determined by the action word $actions(u)$ (labels included). We then construct $\pi.t$ by induction on the hypothesis that π permutes independent actions of $actions(t)$. Let us write

$$t : A = A_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} A_n = B\,.$$

If $\pi = id$ it sufficies to choose $\pi.t = t$. Otherwise let us write $\pi = \pi_0 \circ (i \;\; i{+}1)$ with $\alpha_i \parallel \alpha_{i+1}$

and π_0 permutes independent actions of $\mathsf{tr}' = \alpha_p \cdots \alpha_{i-1}\alpha_{i+1}\alpha_i\alpha_{i+2}\cdots\alpha_n$. By Proposition C.8, there exists a trace

$$u : A_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_{i-1}} A_{i-1} \xRightarrow{\alpha_{i+1}\alpha_i} A_{i+1} \xrightarrow{\alpha_{i+2}} \cdots \xrightarrow{\alpha_n} A_n$$

such that for all $u^2 : A^2 \overset{\mathsf{tr}'}{\Longrightarrow}_{\mathsf{ss}^2} B^2$ verifying $\mathsf{fst}(u^2) = u$, there exists $t^2 : A^2 \overset{\mathsf{tr}}{\Longrightarrow}_{\mathsf{ss}^2} B^2$ such that $\mathsf{fst}(t^2) = t$. Then since π_0 permutes independent actions of $\mathsf{tr}' = \mathit{actions}(u)$, it sufficies to choose $\pi.t = \pi_0.u$ by induction hypothesis. □

Then we can easily extend this result to $\equiv_{\mathsf{por}}$.

Proposition C.10

Let $t : A \overset{\mathsf{tr}}{\Longrightarrow} B$ be a trace and $t' \equiv_{\mathsf{por}} t$. Then writing $\mathit{actions}(t') = \mathsf{tr}'$ we have $t' : A \overset{\mathsf{tr}'}{\Longrightarrow} B$ and, for all $u^2 : A^2 \overset{\mathsf{tr}'}{\Longrightarrow} B^2$ such that $t' = \mathsf{fst}(u^2) \sim \mathsf{snd}(u^2)$, there exists $t^2 : A^2 \overset{\mathsf{tr}}{\Longrightarrow} B^2$ such that $t = \mathsf{fst}(t^2) \sim \mathsf{snd}(t^2)$.

Proof. For the sake of reference, let us write $H(t, t')$ the property to prove. We reason by induction on the hypothesis $t \equiv_{\mathsf{por}} t'$.

- ▷ *case 1*: $t' = \pi.t$, π permutes independent actions of t.

  Direct consequence of Proposition C.9.

- ▷ *case 2*: t' is recipe-equivalent to t.

  Let u^2 with $t' = \mathsf{fst}(u^2) \sim \mathsf{snd}(u^2)$. By static equivence, for any recipes ξ_1, ξ_2 such that

  $$\xi_1\Phi(\mathsf{fst}(u^2)) =_E \xi_2\Phi(\mathsf{fst}(u^2)),$$

  we also have

  $$\xi_1\Phi(\mathsf{snd}(u^2)) =_E \xi_2\Phi(\mathsf{snd}(u^2)).$$

  In particular, t^2 can be obtained by operating on the second component of u^2 the same recipe transformations that have been operated to transform $t' = \mathsf{fst}(u^2)$ into t.

- ▷ *case 3*: (transitivity) $H(t, s)$ and $H(s, t')$ for some trace s.

  Let u^2 with $t' = \mathsf{fst}(u^2) \sim \mathsf{snd}(u^2)$. By hypothesis $H(s, t')$ there exists s^2 such that $s = \mathsf{fst}(s^2) \sim \mathsf{snd}(s^2)$. Hence the result by hypothesis $H(t, s)$. □

And finally we have the Proposition 5.16 that is a corollary of this result.

Theorem 5.16 *(correctness of por)*

Let $\mathbb{O}_1^\forall \subseteq \mathbb{O}_2^\forall$ be universal optimisations. We assume that for all $t \in \mathbb{O}_2^\forall$, there exists t_{ext} such that t is a prefix of t_{ext}, and $t' \in \mathbb{O}_1^\forall$ such that $t' \equiv_{\mathsf{por}} t_{ext}$. Then $\mathbb{O}_1^\forall$ is a correct refinement of $\mathbb{O}_2^\forall$.

Proof. Let $\approx_i = \sqsubseteq_i \cap \sqsupseteq_i$ the notion of equivalence induced by $\mathbb{O}_i^\forall$. The inclusion $\approx_2 \subseteq \approx_1$ is immediate. Let us then assume $P \sqsubseteq_1 Q$ and prove $P \sqsubseteq_2 Q$. Let $t \in \mathbb{T}(P) \cap \mathbb{O}_2^\forall$. Without loss of generality, we assume t maximal, i.e. that there are no transitions possible from its last process. Therefore by hypothesis, there exists $t' \equiv_{\mathsf{por}} t$ such that $t' \in \mathbb{O}_1^\forall$. Since $P \sqsubseteq_1 Q$,

there is $u^2 \in \mathbb{T}(P,Q)$ such that

$$t' = \mathsf{fst}(u^2) \sim \mathsf{snd}(u^2)\,.$$

Therefore by Proposition C.10, there exists $t^2 \in \mathbb{T}(P,Q)$ such that $t = \mathsf{fst}(t^2) \sim \mathsf{snd}(t^2)$. □

3.2 Additional results

We provide some utility results on independent permutations of actions. First, about composition of permutations:

Proposition C.11

Let t be a trace, π permuting independent actions of t, and π' permuting independent actions of $\pi.t$. Then $\pi.\pi'.t = (\pi \circ \pi').t$.

Proof. By definition, if $t : A \overset{\mathsf{tr}}{\Rightarrow} B$, $\pi.\pi'.t$ is the unique trace of the form $A \overset{\pi.\pi'.\mathsf{tr}}{\Longrightarrow} B$, and $(\pi \circ \pi').t$ is the unique trace of the form $A \overset{(\pi\circ\pi').\mathsf{tr}}{\Longrightarrow} B$. Hence the result since $\pi.\pi'.\mathsf{tr} = (\pi \circ \pi').\mathsf{tr}$ by definition of a group action. □

This formalises that the group-action properties of $(\pi, \mathsf{tr}) \mapsto \pi.\mathsf{tr}$ carry on to traces. Then, we also discuss the domain extension of permutations. If π is a permutation of $[\![1,n]\!]$, we define π^{+q}_{+p} permutation of $[\![1, n+p+q]\!]$ by

$$\pi^{+q}_{+p}(x) = \begin{cases} p + \pi(x-p) & \text{if } p < x \leqslant n+p \\ x & \text{otherwise} \end{cases}$$

in particular, the following result is immediate:

Proposition C.12 (extension)

If π permutes independent actions of v, $\pi^{+|w|}_{+|u|}$ permutes independent actions of uvw.

3.3 Decomposition into phases

In this section we prove correct the refinement at the very basis of our partial-order reductions, namely that all traces can be decomposed into phases (modulo permutation of independent actions).

Proposition 5.17 (correctness of basic compression)

$\mathbb{O}^{\forall}_{\mathsf{c},b}$ is a correct refinement of $\mathbb{O}^{\forall}_{\mathsf{all}}$.

Proof. By Proposition 5.16, it sufficies to prove that for all traces t that are maximal (i.e. whose last process is irreducible), there exists π permuting independent actions of t such that $\pi.t$ can be decomposed into phases.

We prove this by induction on the length of t. If t is empty the result is immediate: π is the identity and the phase decomposition consists of a unique empty negative block. Otherwise let us write

$$t : (A \xrightarrow{\alpha} B) \cdot t'\,.$$

Note in particular that the trace t' is also maximal. By induction hypothesis, there exists π'

permuting independent actions of t' such that

$$\pi'.t' = b_0^- \cdot b_1^+ \cdot b_1^- \cdot b_2^+ \cdot b_2^- \cdots b_n^+ \cdot b_n^-$$

where each b_i^+ is a positive or null phase, and each b_i^- is a negative phase.

▷ *case 1*: α is an output or a parallel action.

Then $(A \xrightarrow{\alpha} B) \cdot b_0^-$ is a negative phase and it sufficies to choose $\pi = (\pi')^{+0}_{+1}$.

▷ *case 2*: $\alpha = [\,\tau\,]^{\ell_1|\ell_2}$ (internal communication).

Let us write E the multiset of actions of the word $\mathsf{tr}(b_0^-)$. We partition it into $E = F \uplus G$ where

$$F = \{\!\{\beta \in E \mid \alpha \parallel \beta\}\!\}$$
$$G = \{\!\{[\,a\,]^\ell \in E \mid \ell_1 \preccurlyeq_{pref} \ell \text{ or } \ell_2 \preccurlyeq_{pref} \ell\}\!\}$$

where $\preccurlyeq_{pref}$ refers to the prefix ordering on words. This is indeed a partition of E thanks to the invariant that any label appearing in t' is either incomparable with ℓ_1 and ℓ_2, or a suffix of ℓ_1 or ℓ_2. For the same reason, all actions in F are independent of all actions in G: it is therefore straightforward to construct π_0^- permuting independent actions of b_0^- such that

$$\pi_0^-.b_0^- : B \xrightarrow{\mathsf{tr}_F \cdot \mathsf{tr}_G} C \qquad\qquad \mathsf{tr}_F \in F^\star \qquad\qquad \mathsf{tr}_G \in G^\star\,.$$

Then, we let σ permuting independent actions of

$$s = (A \xrightarrow{\alpha} B) \cdot (\pi_0^-.b_0^-)$$

such that $\sigma.s = A \xRightarrow{\mathsf{tr}_F} B' \xrightarrow{\alpha} B'' \xRightarrow{\mathsf{tr}_G} C$. By definition of F and G, we have $polar(B') \neq \infty$. And by the hypothesis that b_0^- is a negative phase, its last process C has not a polarity of $-\infty$ neither. Therefore $A \xRightarrow{\mathsf{tr}_F} B'$ and $B'' \xRightarrow{\mathsf{tr}_G} C$ are negative phases. All in all, it sufficies to choose

$$\pi = \sigma^{+p}_{+0} \circ (\pi_0^-)^{+p}_{+1} \circ (\pi')^{+0}_{+1} \qquad\qquad \text{with } p = \sum_{i=1}^{n} |b_i^+| + |b_i^-|$$

▷ *case 3*: $\alpha = [\,c(\xi)\,]^\ell$.

Let us write

$$A = (\{\!\{[\,c(x).P\,]^\ell\}\!\} \cup \mathcal{P}, \Phi) \qquad\qquad B = (\{\!\{[\,P'\,]^\ell\}\!\} \cup \mathcal{P}, \Phi)$$

If the label ℓ does not appear in $actions(t')$, then by maximality of t it needs be that $polar(P') = 0$ and $(A \xrightarrow{\alpha} B)$ is therefore a positive phase. In particular $\pi'.t$ is already decomposed into phases and it sufficies to choose $\pi = (\pi')^{+0}_{+1}$.

Otherwise assume that ℓ appears in $actions(t')$. We write

$$\mathsf{tr}_i^- = actions(b_i^-) \qquad\qquad \mathsf{tr}_i^+ = actions(b_i^+)$$

We also consider the phase of t' in which the first action of P' is executed, i.e. the first phase b such that ℓ appears in $\mathit{actions}(b)$. Note that, thanks to the invariant that any label appearing in t' is either incomparable or a suffix of ℓ, α is independent of all actions of all phases of t' preceding b.

▷ *case 3a*: $b = b_i^+$ is a positive or null phase.

Then we fix σ permuting independent actions of

$$\alpha \cdot \mathsf{tr} \qquad\qquad \text{with } \mathsf{tr} = \mathsf{tr}_0^- \cdot \mathsf{tr}_1^+ \cdot \mathsf{tr}_1^- \cdots \mathsf{tr}_{i-1}^+ \cdot \mathsf{tr}_{i-1}^-$$

such that, writing $s = (A \xrightarrow{\alpha} B) \cdot b_0^- \cdot b_1^+ \cdot b_1^- \cdots b_{i-1}^+ \cdot b_{i-1}^-$,

$$\sigma.s : A \overset{\mathsf{tr}}{\Longrightarrow} A' \xrightarrow{\alpha} A''\,.$$

If $b = b_i^+$ is a null phase, $A' \xrightarrow{\alpha} A''$ is a positive phase. If b is a positive phase, $(A' \xrightarrow{\alpha} A'') \cdot b$ is a positive phase too. In both cases, it sufficies to choose

$$\pi = \sigma_{+0}^{+p} \circ (\pi')_{+1}^{+0} \qquad\qquad \text{with } p = \sum_{j=i}^{n} |b_j^+| + |b_j^-|$$

▷ *case 3b*: $b = b_i^-$ is a negative phase.

Similarly to case 2, we fix E the multiset of actions appearing in the word tr_i^- and we partition it as $E = F \uplus G$

$$F = \{\beta \in E \mid \alpha \parallel \beta\} \qquad\qquad G = \{[\,a\,]^{\ell'} \in E \mid \ell \preccurlyeq_{pref} \ell'\}\,.$$

And again we let π_i^- permuting tr_i^- such that

$$\pi_i^-.b_i^- : R \overset{\mathsf{tr}_F}{\Longrightarrow} S \overset{\mathsf{tr}_G}{\Longrightarrow} T \qquad\qquad \mathsf{tr}_F \in F^\star \qquad\qquad \mathsf{tr}_G \in G^\star\,.$$

Then we let σ permuting independent actions of

$$\alpha \cdot \mathsf{tr} \cdot \mathsf{tr}_F \qquad\qquad \text{with } \mathsf{tr} = \mathsf{tr}_0^- \cdot \mathsf{tr}_1^+ \cdot \mathsf{tr}_1^- \cdots \mathsf{tr}_i^+$$

such that, writing $s = (A \xrightarrow{\alpha} B) \cdot b_0^- \cdot b_1^+ \cdot b_1^- \cdots b_i^+ \cdot (\pi_i^-.b_i^-)$,

$$\sigma.s : A \overset{\mathsf{tr}}{\Longrightarrow} A' \overset{\mathsf{tr}_F}{\Longrightarrow} A'' \xrightarrow{\alpha} S \overset{\mathsf{tr}_G}{\Longrightarrow} T\,.$$

For the same reason as in case 2, $A' \overset{\mathsf{tr}_F}{\Longrightarrow} A''$ and $S \overset{\mathsf{tr}_G}{\Longrightarrow} T$ are negative phases. It therefore sufficies to choose

$$\pi = \sigma_{+0}^{+p} \circ (\pi_i^-)_{1+|\mathsf{tr}|}^{p} \circ (\pi')_{+1}^{+0} \qquad\qquad \text{with } p = \sum_{j=i+1}^{n} |b_j^+| + |b_j^-| \qquad \square$$

3.4 Improper positive phases

In this section we prove the correctness of the optimisation $\mathbb{O}^{\forall}_{\mathsf{c+i}}$ consisting of delaying improper blocks as much as possible in traces, as introduced in Section 3.3. First we prove that improper

blocks are essentially independent of all blocks following them.

Proposition C.13

Let $t \in \mathbb{O}^{\forall}_{\mathsf{c}}$ of the form $t = b \cdot u$ where $u \in \mathbb{O}^{\forall}_{\mathsf{c}}$ and b is an improper block. Then there exists u' recipe equivalent to u such that b is independent of all blocks of u'.

Proof. By definition of improper blocks Item 2, if $A \overset{\mathsf{tr}}{\Rightarrow} (\mathcal{P}, \Phi)$ is improper then for all $[\,P\,]^L \in \mathcal{P}$ such that $P \neq 0$, the labels of L are prefix-incomparable with all labels of tr. In particular a straightforward induction shows that all actions of b are sequentially independent of all actions of u. Besides let us write by definition

$$b : (\mathcal{P}, \Phi) \overset{\mathsf{tr}}{\Rightarrow} (\mathcal{Q}, \Phi \cup \{\mathsf{ax}_1 \mapsto t_1, \ldots, \mathsf{ax}_n \mapsto t_n\})$$

with ξ_i a recipe such that $\xi_i \Phi =_E t_i$. We then let the substitution

$$\sigma = \{\mathsf{ax}_1 \mapsto \xi_1, \ldots, \mathsf{ax}_n \mapsto \xi_n\}$$

as well as u' the trace obtained by replacing in u all actions of the form $c(\xi)$ by $c(\xi\sigma)$. By definition, u is recipe equivalent to u' and, since no axioms introduced in b appear in u', b is data independent of all blocks of u'. All in all, b is independent of all blocks of u'. □

The main argument is then a substantially-simple but technical induction delaying improper blocks one by one. For that we prove the following auxiliary result about the preservation of (im)properness when permuting independent blocks, since it is not the case in general as discussed in Section 3.4.

Proposition C.14

Let a sequence of blocks $t = b_1 \cdots b_n$ and a transposition $\pi = (i \;\; i{+}1)$ for some $i \in [\![1, n-1]\!]$. We assume that $b_i \parallel b_{i+1}$. Then

1. *Delaying an improper block preserves improperness:* if b_i is improper then the $i{+}1^{\mathsf{th}}$ block of $\pi.t$ is also improper.
2. *Advancing a proper block preserves properness:* if b_{i+1} is proper then the i^{th} block of $\pi.t$ is also proper.
3. *Swapping two improper blocks preserve improperness:* if b_i and b_{i+1} are improper then the i^{th} and $i{+}1^{\mathsf{th}}$ blocks of $\pi.t$ are improper.

Proof. Item 1 follows from the fact that for any ground term t, recipe ξ and frame Φ, if $\xi\Phi =_E t$ then $\xi(\Phi \cup \Phi') =_E t$ for all frames Φ' such that $dom(\Phi) \cap dom(\Phi') = \varnothing$. Item 2 follows from the fact that for any ground term t and frames Φ, Φ' such that $dom(\Phi) \cap dom(\Phi') = \varnothing$, if t is not deducible in $\Phi \cup \Phi'$ then it cannot be deducible in Φ neither. Let us detail more the argument for Item 3, by writing

$$b_i : (\mathcal{P}_0, \Phi_0) \overset{\mathsf{tr}}{\Rightarrow} (\mathcal{P}_1, \Phi_0 \cup \Phi_1)$$
$$b_{i+1} : (\mathcal{P}_1, \Phi_0 \cup \Phi_1) \overset{\mathsf{tr}'}{\Rightarrow} (\mathcal{P}_2, \Phi_0 \cup \Phi_1 \cup \Phi_2)$$

If we write b'_i, b'_{i+1} the i^{th} and $i{+}1^{\mathsf{th}}$ blocks of $\pi.t$, respectively, they have the form

$$b'_i : (\mathcal{Q}_0, \Phi_0) \overset{\mathsf{tr}'}{\Longrightarrow} (\mathcal{Q}_1, \Phi_0 \cup \Phi_2)$$
$$b'_{i+1} : (\mathcal{Q}_1, \Phi_0 \cup \Phi_2) \overset{\mathsf{tr}}{\Rightarrow} (\mathcal{Q}_2, \Phi_0 \cup \Phi_1 \cup \Phi_2)\,.$$

To show that b'_i and b'_{i+1} are improper, it sufficies to verify the Item 3 of the definition since the two other items immediately follow from the fact that b_i and b_{i+1} are improper.

▷ *Proof that b'_i is improper.*

Let $t \in img(\Phi_2)$ and let us construct a recipe ξ such that $\xi\Phi_0 =_E t$. Since b_{i+1} is improper there exists a recipe ξ_0 such that $\xi_0(\Phi_0 \cup \Phi_1) =_E t$. Besides b_i is also improper hence for all $\mathsf{ax} \in dom(\Phi_1)$ there exists a recipe ξ_{ax} such that $\xi_{\mathsf{ax}}\Phi_0 =_E \mathsf{ax}\,\Phi_1$. It thus sufficies to choose

$$\xi = \xi_0\{\mathsf{ax} \mapsto \xi_{\mathsf{ax}} \mid \mathsf{ax} \in dom(\Phi_1)\}\,.$$

▷ *Proof that b'_{i+1} is improper.*

If $t \in img(\Phi_2)$, since b_i is improper there exists a recipe ξ such that $\xi\Phi_0 =_E t$. In particular $\xi(\Phi_0 \cup \Phi_1) =_E t\Phi_1 =_E t$. □

Proposition 5.21 (correctness criterion)

For all $t \in \mathbb{O}^{\forall}_{\mathsf{c}}$, there exists $t' \equiv_{\mathsf{b\text{-}por}} t$ such that $t' \in \mathbb{O}^{\forall}_{\mathsf{c+i}}$ and t' has the same number of improper blocks t.

Proof. Let us decompose t in blocks as $t : b_0^- \cdot b_1 \cdots b_n$. We say that $i \in [\![1, n-1]\!]$ is a *pending index* in t when b_i is an improper block and there exists $j > i$ such that b_j is a proper block. We prove the proposition by induction on the number of pending indexes.

If there are no pending indexes then $t \in \mathbb{O}^{\forall}_{\mathsf{c+i}}$ by definition. Otherwise let p be the minimal pending index in t. By Proposition C.13 there exists v' recipe equivalent to $b_{p+1} \cdots b_n$ such that b is independent of all blocks of v'. Thus the permutation

$$\pi = (n \;\; n{-}1) \circ (n{-}1 \;\; n{-}2) \circ \cdots \circ (p{+}2 \;\; p{+}1) \circ (p{+}1 \;\; p)$$

permutes independent blocks of $s = b_1 \cdots b_p \cdot v'$. Let us prove that $\pi.s$ has less pending indexes than t, and as many improper blocks as t. Once this is established this will conclude the proof: this makes it possible apply the induction hypothesis to $\pi.s$, thus obtaining $t' \equiv_{\mathsf{b\text{-}por}} \pi.s$ such that $t' \in \mathbb{O}^{\forall}_{\mathsf{c+i}}$ and t' has as many improper blocks as $\pi.s$. In particular $t' \equiv_{\mathsf{b\text{-}por}} \pi.s \equiv_{\mathsf{b\text{-}por}} s \equiv_{\mathsf{b\text{-}por}} t$ hence the desired conclusion.

Let us thus compare the number of pending indexes and improper blocks of t and $\pi.s$. First of all, (im)properness is preserved under recipe equivalence, and t therefore has as many pending indexes and improper blocks as s. Let us decompose s and $\pi.s$ in blocks as

$$s = b_0^- \cdot b_1^s \cdots b_n^s \qquad\qquad \pi.s = b_0^- \cdot b_1^{\pi.s} \cdots b_n^{\pi.s}$$

We then let $i \in [\![1, n]\!]$.

▷ *case 1: $i < p$. Let us prove that b_i and $b_i^{\pi.s}$ are proper (in particular i is not a pending index in $\pi.s$).*

This follows from the fact that $b_i^{\pi\cdot s} = b_i$ and b_i is proper (otherwise p would not be the minimal pending index in t).

▷ *case 2: $i = n$. Let us prove that $b_n^{\pi\cdot s}$ is an improper block.*

Since $b_p^s = b_p$ is improper by hypothesis, $n - p$ applications of Proposition C.14 Item 1 show that the $b_p^{\pi\cdot s}$ is improper.

▷ *case 3: $p \leqslant i < n$. Let us prove that $b_i^{\pi\cdot s}$ is improper iff b_{i+1}^s is improper.*

Let us write, if $u \in \mathbb{O}_{\mathsf{c}}^{\forall}$, b_i^u the i^{th} block of u. If $a \leqslant b$ we also define the permutation

$$\pi_{a\to b} = (b \;\; b{-}1) \circ (b{-}1 \;\; b{-}2) \circ \cdots \circ (a{+}2 \;\; a{+}1) \circ (a{+}1 \;\; a)$$

with $\pi_{a\to a} = id$ by convention. As a preliminary result we prove by induction on i that $b_i^{\pi_{p\to i}\cdot s}$ is improper: if $i = p$ we have $b_p^{\pi_{p\to p}\cdot s} = b_p$ which is improper, and if $i > p$ we know that $b_i^{\pi_{p\to i-1}\cdot s}$ is improper by induction hypothesis, hence the result since $\pi_{p\to i} = (i \;\; i{-}1) \circ \pi_{p\to i-1}$ and by using Proposition C.14 Item 1.

We now prove the actual property, i.e. that $b_i^{\pi\cdot s}$ is improper *iff* b_{i+1}^s is improper. Using the preliminary result above we know that $b_i^{\pi_{p\to i}\cdot s}$ is improper. In particular, using Proposition C.14 Items 2, 3, we obtain that $b_i^{\pi_{p\to i+1}}.s$ is improper *iff* $b_{i+1}^{\pi_{p\to i}\cdot s} = b_{i+1}^s$ is improper. The conclusion eventually follows from the fact that $b_i^{\pi\cdot s} = b_i^{\pi_{p\to n}\cdot s} = b_i^{\pi_{p\to i+1}\cdot s}$.

▷ *Conclusion: $\pi.s$ contains less pending indexes than s, and the same number of improper blocks.*

The fact that $\pi.s$ and s have the same number of improper blocks follows from the fact that the bijection $\varphi : [\![1, n]\!] \to [\![1, n]\!]$ defined by

$$\begin{array}{llll} \varphi(i) = i & \text{if } i < p & \varphi(i) = n & \text{if } i = p \\ \varphi(i) = i + 1 & \text{if } p \leqslant i < n & \varphi(i) = p & \text{if } i = n \end{array}$$

verifies $b_i^{\pi\cdot s}$ is improper *iff* $b_{\varphi(i)}^s$ is improper, by the cases 1,2,3.

Let us then show that $\pi.s$ has less pending indexes than s. For that, by case 1, it sufficies to prove that $b_p^{\pi\cdot s} \cdots b_n^{\pi\cdot s}$ contains less pending indexes than $b_p^s \cdots b_n^s$. Let $p \leqslant i < n$ such that $b_i^{\pi\cdot s}$ is improper. By case 3, there exists $j \in [\![i + 1, n - 1]\!]$ such that $b_j^{\pi\cdot s}$ is proper *iff* there exists $j' \in [\![i + 2, n]\!]$ such that $b_{j'}^s$ is proper, that is, *iff* $i + 1$ is a pending index in s. By case 2 this means that $i < n$ is a pending index in $\pi.s$ *iff* $i + 1$ is a pending index in s. Therefore there are as many pending indexes in $b_p^{\pi\cdot s} \cdots b_n^{\pi\cdot s}$ as in $b_{p+1}^s \cdots b_n^s$, i.e. less than in $b_p^s \cdots b_n^s$ since p is a pending index in s by hypothesis. □

The correctness of the optimisation (Proposition 5.22) then simply follows from this proposition and Proposition 5.20.

3.5 High-priority null phases

In this section we prove the correctness of the optimisation based on executing high-priority internal communications in priority in traces as formalised in Section 3.5. First we state and prove the main technical property of high-priority transitions that makes it correct to execute them as soon as they are available:

Proposition C.15

Let $d \in \mathcal{N}$ a high-priority channel in A, and a transition $A \xrightarrow{\alpha} B$ that is not an internal communication on d. Then d is high-priority in B.

Proof. We use the definition and thus let a trace of the form

$$B \overset{\mathsf{tr}}{\Rightarrow} C \qquad\qquad \text{no labels of } \mathbb{L}_B(d) \text{ appear in } \mathsf{tr},$$

and show that $\mathbb{L}_B(d) = \mathbb{L}_C(d)$. By hypothesis the label of α does not belong to $\mathbb{L}_A(d)$ and therefore, since d is high-priority in A and $\mathbb{L}_A(d) = \mathbb{L}_B(d)$ by definition, we obtain the expected conclusion by considering the trace $A \xrightarrow{\alpha} B \overset{\mathsf{tr}}{\Rightarrow} C$. □

Using this result we can prove that whenever a channel is high-priority at some point A in a maximal trace, there needs be a later transition that executes an internal communication on this channel that was already available in A.

Proposition C.16

Let $t \in \mathbb{O}_{\mathsf{c}}^{\forall} \cap \mathbb{T}(A)$ and $d \in \mathcal{N}$ that is high-priority in A. We also assume that t is maximal, i.e. that no transitions are possible at the end of t. Then t can be decomposed into $t = b_0^- \cdot u \cdot b \cdot v$ for some negative phase b_0^- and $u, v \in \mathbb{O}_{\mathsf{c}}^{\forall}$ such that

- b is a block starting with an internal communication on d with action $[\,\tau\,]^{\ell|\ell'}$ with $\{\ell, \ell'\} \subseteq \mathbb{L}_A(d)$
- the block b is independent of all blocks of u.

Proof. We decompose t in phases as follows $t : b_0^- \cdot b_1 \cdots b_n$. We proceed by induction on n the number of blocks of t. Since d is high-priority in A, an internal communication on d is possible in A by definition. In particular since t is maximal, we know that b_1 cannot be the empty trace. Let us therefore perform a case analysis on the first transition of b_1.

▷ *case 1:* t starts with an internal communication on d.

Then it sufficies to choose $u = \varepsilon$, $b = b_1$ and $v = b_2 \cdots b_n$.

▷ *case 2:* t starts with a transition that is either a public input or an internal communication on a channel $e \neq d$.

Let us write $t_0 = (b_0^- \cdot b_1) : A_1 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} A_n$. By applying Proposition C.15 to each transition of t_0, we know that d is also high-priority in each A_i, in particular in A_n. We can therefore apply the induction hypothesis to $b_2 \cdots b_n$. By doing so we get $i \in [\![2, n]\!]$ such that

1. b_i starts with an internal communication on d with action $[\,\tau\,]^{\ell|\ell'}$ with $\{\ell, \ell'\} \subseteq \mathbb{L}_{A_n}(d)$. Hence $\{\ell, \ell'\} \subseteq \mathbb{L}_A(d)$ since $\mathbb{L}_{A_n}(d) = \mathbb{L}_A(d)$ because d is high-priority in $A_1 = A$. This implies, since b_1 contains no internal communications on d by hypothesis, that the first transition of b_i is sequentially independent—and therefore independent—of all actions of b_1. In particular $b_1 \parallel b_i$.
2. for all $j \in [\![2, i-1]\!]$, $b_j \parallel b_i$.

Altogether it sufficies to choose $u = b_1 \cdots b_{i-1}$, $b = b_i$ and $v = b_{i+1} \cdots b_n$. □

Proposition 5.26 (correctness criterion)

Let $t = u \cdot v$ a maximal trace, where $u, v \in \mathbb{O}_{\mathsf{c}}^{\forall}$ and v does not contain any high-priority transitions. Then there exists π permuting independent blocks of u such that $\pi.u \in \mathbb{O}_{\mathsf{0}}^{\forall}$.

Proof. We decompose u and v in blocks

$$u = b_0^- \cdot b_1 \cdots b_n \qquad\qquad v = b_{n+1} \cdots b_m$$

with $b_i : A_i \overset{\mathsf{tr}_i}{\Longrightarrow} A_{i+1}$. First of all we observe that for all $i > n$, there are no high-priority channels in A_i. Indeed using Proposition C.15, a quick induction on the length of s shows that for all maximal traces $s : A \overset{\mathsf{tr}}{\Rightarrow} B$ such that there exists a high-priority channel in A, there exists a high-priority transition in s. In the context of our proposition, this would contradict the hypothesis that v does not contain any high-priority transitions. We then prove the property by induction on the number of processes A_i, $i \in [\![1, n]\!]$, such that there is a high-priority channel in A_i.

If there is no such process then it sufficies to choose $\pi = id$. Otherwise let $p \in [\![1, n]\!]$ be the minimal index such that there exists a high-priority channel in A_p, and $d \in \mathcal{N}$ the minimal such channel w.r.t. $\preccurlyeq_{ch}$. By Proposition C.16 there exists $q \in [\![p, m]\!]$ such that b_q starts with an internal communication on d and for all $p \leqslant r < q$, $b_r \parallel b_q$. If we consider the minimal such index q, we know by Proposition C.15 that d is high-priority in A_q and in particular $q \leqslant n$ by the preliminary remark. Let us thus assume so. Therefore the permutation

$$\pi_0 = (p \;\; p{+}1) \circ (p{+}1 \;\; p{+}2) \circ \cdots \circ (q{-}2 \;\; q{-}1) \circ (q{-}1 \;\; q)$$

permutes independent blocks of u and the p^{th} block of $\pi_0.u$ starts with a high-priority transition. We write u_p the trace composed of the first p blocks of $\pi_0.u$. We have $u_p \in \mathbb{O}^{\forall}_{\mathsf{0}}$: no high-priority transitions are possible in the first $p-1$ blocks of u_p by minimality of p, and the p^{th} block starts with a high-priority transition.

We then let u_{n-p} the remaining $n-p$ blocks of $\pi_0.u$, i.e. the trace such that $\pi_0.u = b_0^- \cdot u_p \cdot u_{n-p}$. We then apply the induction hypothesis to $u_{n-p} \cdot v$ (which is indeed a maximal trace) which gives π_1 permuting independent blocks of u_{n-p} such that $\pi_1.u_{n-p} \in \mathbb{O}^{\forall}_{\mathsf{0}}$. It therefore sufficies to choose $\pi = {\pi_1}^{+0}_{+p} \circ \pi_0$ (notation of the extension lemma, Proposition C.12). □

Corollary 5.27 (correctness of high-priority transitions)
$\mathbb{O}^{\forall}_{\mathsf{c+i^*+0}}$ is a correct refinement of $\mathbb{O}^{\forall}_{\mathsf{c+i^*}}$.

Proof. We use the characterisation of Proposition 5.20. Let $t \in \mathbb{O}^{\forall}_{\mathsf{c+i^*}}$ and t_{ext} an arbitrary maximal extension of t (obtained by executing transitions after t as long as it is possible). By Propositions 5.21, 5.24 there exists $t_{\mathsf{i^*}} \equiv_{\mathsf{b\text{-}por}} t_{ext}$ such that $t_{\mathsf{i^*}} \in \mathbb{O}^{\forall}_{\mathsf{c+i^*}}$. Let us decompose $t_{\mathsf{i^*}}$ as

$$t_{\mathsf{i^*}} = b_0^- \cdot u \cdot v$$

where b_0^- is a negative phase, $u \in \mathbb{O}^{\forall}_{\mathsf{c}}$ only contains proper blocks and $v \in \mathbb{O}^{\forall}_{\mathsf{c}}$ only contains improper blocks. Note that $t_{\mathsf{i^*}}$ is maximal since $t_{\mathsf{i^*}} \equiv_{\mathsf{b\text{-}por}} t_{ext}$ and t_{ext} is maximal. Recall also that improper blocks never contain high-priority transitions by definition. There we can apply Proposition 5.26 to obtain π permuting independent actions of u such that $\pi.u \in \mathbb{O}^{\forall}_{\mathsf{0}}$. Hence $b_0^- \cdot \pi.u \cdot v \in \mathbb{O}^{\forall}_{\mathsf{c+i^*+0}}$, keeping in mind Proposition 5.23. □

3.6 Reduction of independent blocks

Now we prove the correctness of the optimisation $\mathbb{O}^{\forall}_{\mathsf{por}}$ introduced in Section 3.6. We recall that we assume an ordering $\preccurlyeq$ on words of actions such that two words of independent actions are always strictly comparable. In particular we can rephrase more simply the definition of the fact that a block cannot follow an other one:

Proposition C.17

Let two blocks $b \parallel b'$. If b cannot follow b' then $\mathit{actions}(b) \prec \mathit{actions}(b')$, and if b' starts with a high-priority transition then b too and on the same private channel.

Proof. By definition if the block b cannot follow the block b' then
1 $\neg(\mathit{actions}(b') \prec \mathit{actions}(b))$; and
2 if b' starts with a high-priority transition then b too; and
3 b and b' do not start with a high-priority transition on different channels.
Hence the result, keeping in mind the assumption on $\preccurlyeq$ that $b \parallel b'$ implies $\mathit{actions}(b) \prec \mathit{actions}(b')$ or $\mathit{actions}(b') \prec \mathit{actions}(b)$. □

We write $\preccurlyeq_{lex}$ the lexicographic extension of $\preccurlyeq$ to traces of same number of blocks, i.e. if $t = b_1 \cdots b_n$ and $t' = b'_1 \cdots b'_n$,

$$t \preccurlyeq_{lex} t' \quad \textit{iff} \quad \mathit{actions}(b_1) \prec \mathit{actions}(b'_1) \ \text{ or } \ \begin{cases} \mathit{actions}(b_1) \preccurlyeq \mathit{actions}(b'_1) \\ b_2 \cdots b_n \preccurlyeq_{lex} b'_2 \cdots b'_n \end{cases}$$

By convention, two traces with a different number of blocks are incomparable w.r.t. $\preccurlyeq_{lex}$. The core argument to prove the correctness of the optimisation is that $\mathbb{O}^{\forall}_{\mathsf{r}}$ contains minimal traces among those obtainable by permutation of independent actions:

Proposition C.18

Let $u = (b_0^- \cdot b_1 \cdots b_n) \in \mathbb{O}^{\forall}_0$ and $i \in [\![2, n]\!]$ such that $\neg\mathsf{Minimal}(b_1 \cdots b_{i-1}, b_i)$. We assume i minimal among indexes with this property and write $\pi = (i-1 \quad i)$. Then π permutes independent actions of u, $\pi.u \prec_{lex} u$ and $p \in \mathbb{O}^{\forall}_0$ where p consists of the first $i-1$ blocks of $\pi.u$.

Proof. By definition of the predicate $\mathsf{Minimal}$, $b_{i-1} \parallel b_i$, $\mathsf{tr}(b_i) \prec \mathsf{tr}(b_{i-1})$ and b_i cannot follow b_{i-1}. Note that the eventuality $\neg\mathsf{Minimal}(b_1 \cdots b_{i-2}, b_{i-1})$ has been excluded by minimality of i. In particular π permutes independent blocks of u and $\pi.u \prec_{lex} u$. Thus let b'_j the j^{th} block of $\pi.u$, $j \in [\![1, i-1]\!]$ and let us show that it starts with a high-priority transition if one is available. If $j < i-1$ then $\pi(j) = j$, hence $b'_j = b_j$. The conclusion thus follows from the assumption $u \in \mathbb{O}^{\forall}_0$. Otherwise $j = i-1$ and we write A such that $b_{i-1} \in \mathbb{T}(A)$. We recall that b_i cannot follow b_{i-1}; therefore by Proposition C.17 we are in one of the following cases.

▷ *case 1:* b_{i-1} does not start with a high-priority transition

In particular the assumption that $u \in \mathbb{O}^{\forall}_0$ ensures that no high-priority transitions are possible from A which appears to be the initial process of the block b'_j, hence the conclusion.

▷ *case 2:* b_{i-1} and b_i start with a high-priority internal communication on a channel d

In particular since $u \in \mathbb{O}^{\forall}_0$, d is the minimal high-priority channel among those available

in A, and the block b'_j (which takes the same transitions as b_i) starts with a high-priority. □

Corollary C.19

Let $t \in \mathbb{O}^{\forall}_{\mathsf{c}}$ a maximal trace of the form $t = u \cdot v$ where $u \in \mathbb{O}^{\forall}_{\mathsf{0}} \setminus \mathbb{O}^{\forall}_{\mathsf{r}}$, and $v \in \mathbb{O}^{\forall}_{\mathsf{c}}$ does not contain any high-priority transitions. Then there exists π permuting independent actions of u such that $\pi.u \prec_{lex} u$ and $\pi.u \in \mathbb{O}^{\forall}_{\mathsf{0}}$.

Proof. Consider the block decomposition $u = b_0^- \cdot b_1 \cdots b_n$. By hypothesis $u \in \mathbb{O}^{\forall}_{\mathsf{0}} \setminus \mathbb{O}^{\forall}_{\mathsf{r}}$ and we can therefore fix $i \in [\![2, n]\!]$ the minimal index such that $\neg\mathsf{Minimal}(b_1 \cdots b_{i-1}, b_i)$. By Proposition C.18, $\pi = (i-1 \;\; i)$ permutes independent blocks of u, $\pi.u \prec_{lex} u$ and the first $i-1$ blocks of $\pi.u$ form a trace of $\mathbb{O}^{\forall}_{\mathsf{0}}$.

Besides let s consisting of the last $n-i+1$ blocks of $\pi.u$. By hypothesis $s \cdot v$ is maximal and v does not contain high-priority transitions: in particular by Proposition 5.26 there exists σ permuting independent blocks of s such that $\sigma.s \in \mathbb{O}^{\forall}_{\mathsf{0}}$. Then we define, using the notations of the extension lemma (Proposition C.12), $\tau = \sigma^{+0}_{+i-1} \circ \pi$. It permutes independent blocks of u, $\tau.u \prec_{lex} u$ and $\tau.u \in \mathbb{O}^{\forall}_{\mathsf{0}}$. □

Proposition C.20

Let $t \in \mathbb{O}^{\forall}_{\mathsf{c}}$ a maximal trace of the form $t = u \cdot v$ where $u \in \mathbb{O}^{\forall}_{\mathsf{0}}$, and $v \in \mathbb{O}^{\forall}_{\mathsf{c}}$ does not contain any high-priority transitions. Then there exists π permuting independent blocks of u such that $\pi.u \in \mathbb{O}^{\forall}_{\mathsf{0}} \cap \mathbb{O}^{\forall}_{\mathsf{r}}$. Besides $\pi.u \prec_{lex} u$ if $u \notin \mathbb{O}^{\forall}_{\mathsf{r}}$.

Proof. Let us consider the set of traces

$$U = \mathbb{O}^{\forall}_{\mathsf{0}} \cap \{\pi.u \mid \pi \text{ permutes independent blocks of } t\}.$$

The set U is finite (its size is bounded by $|u|!$). In particular the ordering $\preccurlyeq_{lex}$ is well-founded on U, i.e. there exist no infinite decreasing sequences of elements of U w.r.t $\prec_{lex}$. Let us thus show by well-founded induction on u' that for all traces of the form $u' \cdot v$, $u' \in U$, there exists π permuting independent blocks of u' such that $\pi.u' \in \mathbb{O}^{\forall}_{\mathsf{r}}$. If $u' \in \mathbb{O}^{\forall}_{\mathsf{r}}$ it sufficies to choose $\pi = id$. Otherwise by Proposition C.19 there exists π_0 permuting independent blocks of u' such that $\pi_0.u' \prec_{lex} u'$ and $\pi_0.u' \in U$. Hence the result by induction hypothesis applied to $\pi_0.u'$. □

We can eventually prove the correctness of $\mathbb{O}^{\forall}_{\mathsf{por}}$. It essentially relies on the characterisation of correctness provided by Proposition 5.20, applying two times Proposition C.20. The decreasing argument w.r.t. the lexicographic extension of $\preccurlyeq$ is a simple consequence of the fact that the proof is performed by well founded induction w.r.t. this ordering.

Proposition 5.28 (correctness criterion)

For all maximal traces $t \in \mathbb{O}^{\forall}_{\mathsf{c+i^*+0}}$, there exists π permuting independent blocks of t such that $\pi.t \in \mathbb{O}^{\forall}_{\mathsf{por}}$. Besides $\pi.t \prec_{lex} t$ if $t \notin \mathbb{O}^{\forall}_{\mathsf{por}}$, with $\preccurlyeq_{lex}$ the lexicographic extension of $\preccurlyeq$.

Proof. Let us decompose such a trace t into $t : b^- \cdot t_p \cdot t_i$ where b^- is a negative phase, $t_p \in \mathbb{O}_{\mathsf{0}}^{\forall}$ only contains proper blocks and $t_i \in \mathbb{O}_{\mathsf{c}}^{\forall}$ only contains improper blocks. Besides, by maximality of t_i, no high-priority channels appear in any extended process of t_i (otherwise a block of t_i would start with an internal communication by Proposition C.16, impossible for improper blocks). Therefore $t_i \in \mathbb{O}_{\mathsf{0}}^{\forall}$. We can thus apply Proposition C.20 with $u = t_i$ and $v = \varepsilon$ and we obtain π_i permuting independent blocks of t_i such that $\pi_i.t_i \in \mathbb{O}_{\mathsf{0}}^{\forall} \cap \mathbb{O}_{\mathsf{r}}^{\forall}$ and $\pi_t.t_i \preccurlyeq_{lex} t_i$.

We observe that $\pi_i.t_i$ is also maximal and does not contain any high-priority transitions. By applying Proposition C.20 again, but with $u = t_p$ and $v = \pi_i.t_i$, we obtain π_p permuting independent blocks of t_p such that $\pi_p.t_p \in \mathbb{O}_{\mathsf{0}}^{\forall} \cap \mathbb{O}_{\mathsf{r}}^{\forall}$ and $\pi_t.t_p \preccurlyeq_{lex} t_p$. In particular, using the notations of the extension lemma (Proposition C.12),

$$\pi = \pi_{p}{}_{+0}^{+|t_i|} \circ \pi_{i}{}_{+|t_p|}^{+0}$$

permutes independent actions of t and

$$\pi.t = b_0^- \cdot \pi_p.t_p \cdot \pi_i.t_i \,.$$

There are only improper blocks in $\pi_i.t_i$ by Proposition C.14 Item 3, and therefore $\pi.t \in \mathbb{O}_{\mathsf{c+i^*}}^{\forall}$ by Proposition 5.23. All in all, $\pi.t \in \mathbb{O}_{\mathsf{por}}^{\forall}$ and $\pi.t \preccurlyeq_{lex} t$. □

Corollary 5.29 (correctness of lexicographic reduction)
$\mathbb{O}_{\mathsf{por}}^{\forall}$ is a correct refinement of $\mathbb{O}_{\mathsf{c+i^*+0}}^{\forall}$.

Proof. Let $t \in \mathbb{O}_{\mathsf{c+i^*+0}}^{\forall}$ and t_{ext} an arbitrary, maximal extension of t obtained by executing as many transitions of t as possible. By Propositions 5.21, 5.24, 5.26 there exists $\bar{t} \equiv_{\mathsf{b\text{-}por}} t_{ext}$ such that $\bar{t} \in \mathbb{O}_{\mathsf{c+i^*+0}}^{\forall}$. Then by Proposition 5.28 there exists $u \equiv_{\mathsf{b\text{-}por}} \bar{t}$ such that $u \in \mathbb{O}_{\mathsf{por}}^{\forall}$. Hence the conclusion by Proposition 5.20. □

4 Correctness of symmetries

In this section we prove the correctness of the optimisations presented in Section 4, i.e. the reduction by symmetries.

4.1 Preliminary notations and results

First of all we introduce some notions that will be at the core of most proofs of correctness of our reductions by symmetry. Intuitively when there is a symmetry between two processes P_a and P_b, we *relabel* them (that is, we swap their labels) and the resulting process is structurally equivalent to the initial one. Then we use the fact that structurally equivalent processes produce *almost identical* traces, meaning that their can be matched equivalently. We formalise these two notions in the following paragraphs.

Relabelling We first define an action of the group of label permutations. It acts on extended (twin) processes, actions and traces by relabelling. Formally if π is a permutation

of labels, we write

$$A\pi \qquad A^2\pi \qquad \alpha\pi \qquad t\pi \qquad t^2\pi$$

respectively the extended process A, extended twin process A^2, labelled action α, trace t and extended twin trace t^2 where each label $\ell \cdot \ell'$ has been replaced by $\pi(\ell) \cdot \ell'$. For this transformation to be well defined, we always consider that π is *consistent* (w.r.t. the object it is applied to) which means that

1. the set $supp(\pi) = \{\ell \mid \pi(\ell) \neq \ell\}$ only contains pairwise incomparable labels w.r.t. the prefix ordering
2. if applied to $A = (\{\!\{[\,P_1\,]^{\ell_1}, \ldots, [\,P_n\,]^{\ell_n}\}\!\}, \Phi)$ then for all $\ell \in supp(\pi)$, there exists $i \in [\![1, n]\!]$ such that ℓ is a prefix of ℓ_i
3. if applied to $t \in \mathbb{T}(A)$ with $A = (\{\!\{[\,P_1\,]^{\ell_1}, \ldots, [\,P_n\,]^{\ell_n}\}\!\}, \Phi)$ then π is consistent w.r.t. A
4. if applied to an extended twin process or an extended twin trace, π is consistent w.r.t. its first projection.

We recall that extended twin processes are only labelled on their first projection. This operation has a lot of trivial but important properties that we will use implicitly in incoming proofs:

Proposition C.21

With the notations above:

1. $\mathsf{fst}(t^2\pi) = \mathsf{fst}(t^2)\pi$ and $\mathsf{snd}(t^2\pi) = \mathsf{snd}(t^2)$
2. Hence: $t = \mathsf{fst}(t^2) \sim \mathsf{snd}(t^2)$ *iff* $t\pi = \mathsf{fst}(t^2\pi) \sim \mathsf{snd}(t^2\pi)$
3. if $\sigma \in S_n$ and π is the permutation such that $\pi(\ell_i) = \ell_{\sigma(i)}$, $A\pi = \sigma^{-1}.A$.
4. if X is any of the objects above, $Xid = X$ and $X\pi\pi' = X(\pi' \circ \pi)$.
5. if π is consistent w.r.t. A, $A \xrightarrow{\alpha} B$ then π is consistent w.r.t. B.

Structural equivalence Then we list several core properties associated to structural equivalence. The first one is the trivial observation that it is preserved by the action of permutations:

Proposition C.22

If $A \equiv B$ consist of n parallel subprocesses and $\pi \in S_n$, $\pi.A \equiv \pi.B$.

An important corollary is that the function $(\pi, A) \mapsto \pi.A$ is a group action on the set of extended processes quotiented by $\equiv$. In particular its stabilisers are permutation groups:

Corollary C.23

For all extended processes A consisting of n processes, $\mathsf{Stab}(A)$ is a subgroup of S_n.

An other property is that structural equivalence preserves static equivalence and skeletons. This is simply because static equivalence is preserved by bijective renaming of private names.

Proposition C.24

If $A \equiv B$ then $A \sim B$, and if $P \equiv_E Q$ then $\mathsf{skel}(P\wr) = \mathsf{skel}(Q\wr)$.

The goal of our reductions by symmetry is to identify subprocesses that will behave identically, so that the execution of one is prioritised over the others; we formalise this by a notion that identifies traces that are identical up to their labels and public channels. Formally we write $X \simeq_{Ch} Y$ when X and Y are two identical objets up to renaming of public channels, i.e. $X = Y\varrho$ for ϱ permutation of $\mathcal{F}_0$; then we say that two traces u, v are *almost identical*, written $u \cong v$, when there exists a session matching σ for u and v (notion introduced in Appendix 1) and ϱ an α-renaming such that

$$u : A_0 \xrightarrow{[\,a_1\,]^{\ell_1}} \cdots \xrightarrow{[\,a_n\,]^{\ell_n}} A_n$$
$$v : (B_0 \xrightarrow{[\,b_1\,]^{\ell_1\sigma}} \cdots \xrightarrow{[\,b_n\,]^{\ell_n\sigma}} B_n)\varrho$$

where for all i, $a_i \simeq_{Ch} b_i$ and A_i, B_i are of the form

$$A_i = (\{\!|[\,P_1\,]^{\ell'_1}, \ldots, [\,P_m\,]^{\ell'_m}|\!\}, \Phi)$$
$$B_i \simeq_{Ch} (\{\!|[\,Q_1\,]^{\ell'_1\sigma}, \ldots, [\,Q_m\,]^{\ell'_m\sigma}|\!\}, \Phi')$$

with $P_j \equiv_E Q_j$ for all j and $\Phi =_E \Phi'$. We sometimes make the session matching explicit by writing $u \overset{\sigma}{\cong} v$. Almost identity is an equivalence relation on traces, which is mostly justified by Proposition C.3. We now identify some transformations that produce almost identical traces and we give the core property that we expect such traces to verify.

Proposition C.25

If π is consistent w.r.t. t then $t \overset{\pi}{\cong} t\pi$.

The proof of this proposition is straightforward ($\equiv_E$ and $\simeq_{Ch}$ can be replaced by syntactic equalities). An other property is:

Proposition C.26

If $u \overset{\pi}{\cong} v$ and ϱ is a permutation of $\mathcal{F}_0$ then $u \overset{\pi}{\cong} v\varrho$. Besides if u and v have the same initial extended process B, $\pi_{|L} = id$ for L the set of labels of B, and $t : A \overset{\mathsf{tr}}{\Rightarrow} B$, then $t \cdot u \overset{\bar{\pi}}{\cong} t \cdot v$ where $\bar{\pi}$ is the extension of π that coincides with π on $dom(\pi)$ and is the identity on the labels of t.

But the most important property is that structurally equivalent processes have almost identical traces (again with the identity renaming of channels).

Proposition C.27

Let A_0, B_0 two extended processes such that $A_0 \equiv B_0$, and L the set of labels appearing in A_0 and B_0 (which is the same). Then for all traces $u \in \mathbb{T}(A_0)$, there exists $v \in \mathbb{T}(B_0)$ such that $u \overset{\pi}{\cong} v$ with $\pi_{|L} = id$ and $u \sim v$.

Proof. Intuitively the trace v is constructed by mirroring all transitions of u in B_0. The technical part is that structural equivalence includes associative-commutative reordering of parallel operators, making the labels of u and v different after the first transition. Let us

write $u : A_0 \xrightarrow{[a_1]^{\ell_1^u}} \cdots \xrightarrow{[a_n]^{\ell_n^u}} A_n$ and

$$A_0 = (\{\!\{ [P_1]^{\ell_1^0}, \ldots, [P_m]^{\ell_m^0} \}\!\}, \Phi)$$
$$B_0 = (\{\!\{ [Q_1]^{\ell_1^0}, \ldots, [Q_m]^{\ell_m^0} \}\!\}, \Phi)\varrho$$

for some α-renaming ϱ and $P_i \equiv_E Q_i$ for all i. We show by induction on n that there exists a trace

$$v : B_0 \xrightarrow{[a_1]^{\ell_1^u \pi}} \cdots \xrightarrow{[a_n]^{\ell_n^u \pi}} B_n$$

for some session matching π for u and v such that $\pi_{|L} = id$, and A_i, B_i are of the form

$$A_i = (\{\!\{ [R_1]^{\ell_1}, \ldots, [R_m]^{\ell_m} \}\!\}, \Phi_i)$$
$$B_i = (\{\!\{ [S_1]^{\ell_1}, \ldots, [S_m]^{\ell_m} \}\!\}, \Phi_i')\varrho$$

with $R_j \equiv_E S_j$ for all j and $\Phi_i =_E \Phi_i'$.

For $n = 0$ it sufficies to choose v the empty trace and $\pi = id$. For $n > 0$ we write $u = u_0 \cdot u'$ with $u_0 : A_0 \xrightarrow{\alpha} A_1$ and perform a case analysis on the rule from which the transition u_0 is derived. Below we write $L(A)$ the set of labels appearing in a process A.

▷ *case 1:* rules (In), (Out) or (Comm)

We write $v_0 : B_0 \xrightarrow{\alpha} B_1$, the transition obtained by performing the same action as u_0 in B_0 at the same label. In particular we apply the induction hypothesis to u' from A_1, B_1, which gives $v' \in \mathbb{T}(B_1)$ and π a session matching for u' and v'. Then π is also a session matching for t and s and $\pi_{|L(A_0)} = \pi_{|L(A_1)} = id$ and it sufficies to choose $v = v_0 \cdot v'$.

▷ *case 2:* rule (Par)

Let us use the following notations

$$\begin{aligned} u_0 : A_0 &= (\{\!\{ [\textstyle\prod_{i=1}^p P_i]^{\ell} \}\!\} \cup \{\!\{ [P_i]^{\ell_i} \}\!\}_{i=p+1}^q, \Phi) \\ &\xrightarrow{\tau} (\{\!\{ [P_i]^{\ell \cdot i} \}\!\}_{i=1}^p \cup \{\!\{ [P_i]^{\ell_i} \}\!\}_{i=p+1}^q, \Phi) \\ B_0 &= (\{\!\{ [Q]^{\ell} \}\!\} \cup \{\!\{ [Q_i]^{\ell_i} \}\!\}_{i=p+1}^q, \Phi)\varrho \end{aligned}$$

with $\prod_{i=1}^p P_i \equiv_E Q$ and for all $i \in [\![p+1, n]\!]$, $P_i \equiv_E Q_i$. By Proposition C.24 and since Q is in $\rightsquigarrow$-normal form by definition, we have $Q = \prod_{i=1}^p Q_i$ where, for some permutation $\sigma \in S_p$, $Q_i \equiv_E P_{\sigma(i)}$ for all $i \in [\![1, p]\!]$. In particular by applying rule (Par) in B_0 at label ℓ we obtain a transition $v_0 : B_0 \xrightarrow{\tau} B_1$ such that we can apply the induction hypothesis from $A_1, B_1\pi_0$, where π_0 is the label permutation defined by $\pi_0(\ell \cdot i) = \ell \cdot \sigma(i)$ and $supp(\pi_0) \subseteq \{\ell \cdot 1, \ldots, \ell \cdot p\}$. Note that $B_1\pi_0$ is well defined since π_0 is consistent w.r.t. B_1. Let us thus apply the induction hypothesis to the processes $A_1, B_1\pi_0$ and the trace u'. We obtain a trace $v' \in \mathbb{T}(B_1\pi_0)$ and a session matching π' for u' and v'. To conclude it sufficies to choose $v = v_0 \cdot v'\pi_0^{-1}$ and π defined by $\pi(\ell) = \ell$ and

$$\begin{aligned} &\pi(\ell \cdot i \cdot \ell') = \pi_0^{-1}(\ell \cdot i) \cdot \ell'' && \text{where } \pi'(\ell \cdot i \cdot \ell') = \ell \cdot i \cdot \ell'' \\ &\pi(\ell') = \pi'(\ell') && \text{if } \ell \text{ not a prefix of } \ell' \end{aligned}$$

π is well defined and a session matching for u and v because π' is a session matching for u' and v' such that $\pi'_{|L(A_1)} = id$. □

Finally, almost identical traces preserve POR properties. Indeed if $u \overset{\sigma}{\cong} v$, a permutation π permutes independent blocks of u *iff* it permutes independent blocks of v and, in this case, $\pi.u \overset{\sigma}{\cong} \pi.v$. In addition, since the frames of almost identical traces are equal modulo theory and renaming of private names, a block is proper *iff* an almost identical block is. All in all:

Proposition C.28

If $u \cong v$ then $u \in \mathbb{O}^{\forall}_{\mathsf{c}+\mathsf{i}^*}$ *iff* $v \in \mathbb{O}^{\forall}_{\mathsf{c}+\mathsf{i}^*}$.

4.2 Universal symmetries

Permutation groups Let us first prove that the sets Sym_1 and Sym_2 (notation of the optimisation) are subgroups of S_n.

In the next two theorems we fix the notations of the optimisation Sym_1 and Sym_2 depend of. We therefore consider a trace $t \in \mathbb{T}(P)$ whose final process A consists of n processes.

Proposition C.29

Sym_1 is a subgroup of S_n.

Proof. Since $\mathsf{Stab}(A)$ is a group, it sufficies to prove Sym_1^Q is a subgroup of S_n. We recall that it is the set of permutations $\pi \in S_n$ such that for all traces $t^2 : (P,Q) \overset{\mathsf{tr}}{\Rightarrow} A^2$ such that $t = \mathsf{fst}(t^2) \sim \mathsf{snd}(t^2)$, there exists a trace $s^2 : (P,Q) \overset{\mathsf{tr}}{\Rightarrow} \pi.A^2$ such that $t = \mathsf{fst}(s^2)$. This set contains the identity (it sufficies to choose $s^2 = t^2$). Therefore by Proposition 5.12 it sufficies to prove that Sym_1^Q is closed by composition.

Let $\pi, \sigma \in \mathsf{Sym}_1^Q$. In particular there is $s^2 : (P,Q) \overset{\mathsf{tr}}{\Rightarrow} \pi.A^2$ such that $t = \mathsf{fst}(s^2)$. Since s^2 has the same final frames as t^2, we also have $\mathsf{fst}(s^2) \sim \mathsf{snd}(s^2)$. Therefore since $\pi' \in \mathsf{Sym}_1^Q$, there exists $u^2 : (P,Q) \overset{\mathsf{tr}}{\Rightarrow} \sigma.\pi.A^2 = (\sigma \circ \pi).A^2$ such that $\mathsf{fst}(u^2) = t$, hence $\pi \circ \sigma \in \mathsf{Sym}_1^Q$. □

Proposition C.30

Sym_2 is a subgroup of S_n.

Proof. Rephrasing the definition, $\pi \in \mathsf{Sym}_2$ *iff* there exists ϱ permutation of $\mathcal{F}_0$ such that $\pi.A \equiv A\varrho$ and, for all twin traces $t^2 : (P,Q) \overset{\mathsf{tr}}{\Rightarrow} A^2$ such that $t = \mathsf{fst}(t^2) \sim \mathsf{snd}(t^2)$, we have $\pi.\mathsf{snd}(A^2) \equiv \mathsf{snd}(A^2)\varrho$. Similarly to Sym_1, it sufficies to prove that Sym_2 is closed by composition to conclude by Proposition 5.12, since $id \in \mathsf{Sym}_2$. Thus let $\pi, \pi' \in \mathsf{Sym}_2$ and ϱ, ϱ' permutations of $\mathcal{F}_0$ such that

1. $\pi.A \equiv A\varrho$ and $\pi'.A \equiv A'\varrho'$
2. for all twin traces $t^2 : (P,Q) \overset{\mathsf{tr}}{\Rightarrow} A^2$ such that $t = \mathsf{fst}(t^2) \sim \mathsf{snd}(t^2)$, $\pi.\mathsf{snd}(A^2) \equiv \mathsf{snd}(A^2)\varrho$ and $\pi'.\mathsf{snd}(A^2) \equiv \mathsf{snd}(A^2)\varrho'$.

We prove that $\pi \circ \pi' \in \mathsf{Sym}_2$. First of all

$$(\pi \circ \pi').A \equiv \pi.(A\varrho') = (\pi.A)\varrho' \equiv A\varrho\varrho' .$$

Then let $t^2 : (P, Q) \overset{\mathsf{tr}}{\Rightarrow} A^2$ such that $t = \mathsf{fst}(t^2) \sim \mathsf{snd}(t^2)$. With the same reasoning as above we obtain $(\pi \circ \pi').\mathsf{snd}(A^2) \equiv \mathsf{snd}(A^2)\varrho\varrho'$, hence the conclusion. □

Correctness of the reduction by symmetry First of all we prove that the optimisation consisting of discarding transitions based on the analysis of Sym. At first we only study the refinement of $\mathbb{O}^{\forall}_{\mathsf{c+i}^*}$ and tackle later the question of the compatibility with high-priority transitions and with the lexicographic reduction. We use again the notations of the optimisation and, if $\pi \in \mathsf{Sym}$ and $L = \{\ell_i\}_{i=1}^n$, we let $\bar{\pi}$ the label permutation defined by $supp(\bar{\pi}) \subseteq L$ and $\bar{\pi}(\ell_i) = \ell_{\pi(i)}$.

Proposition C.31

Let $\pi \in \mathsf{Sym}$. We also let a trace $u \in \mathbb{O}^{\forall}_{\mathsf{c+i}^*}$ of the form

$$u = t \cdot (A \xrightarrow{[a]^\ell} B) \cdot v$$

We assume that for all traces $u' \cong u$ of the form

$$u' = t \cdot (A \xrightarrow{[a']^{\ell\bar{\pi}}} B') \cdot v' \qquad (\star)$$

there exists $s \in \mathbb{T}(Q)$ with $u' \sim s$ and a session matching for u' and s. Then there exists $s \in \mathbb{T}(Q)$ with $u \sim s$ and a session matching for u and s.

Proof. We decompose the proof in three parts, isolating the characterisation based on Sym_1, the one based on Sym_2, and how to compose the two.

***Part 1**: Proof in the case $\pi \in \mathsf{Sym}_1$.* Let $u \in \mathbb{T}(P)$ a trace with the same notations as the statement of the theorem. We write

$$u_0 : A \xrightarrow{[a]^\ell} B$$

The permutation $\bar{\pi}$ is consistent w.r.t. A and the trace $(u_0 \cdot v)\bar{\pi}$ is therefore well defined, and of the form

$$(u_0 \cdot v)\bar{\pi} : (\pi^{-1}.A \xrightarrow{[a]^{\ell\bar{\pi}}} B) \cdot v\bar{\pi}$$

But $\pi \in \mathsf{Sym}_1$ and in particular $\pi \in \mathsf{Stab}(A)$. Since $\mathsf{Stab}(A)$ is a group we deduce, by definition, that $\pi^{-1}.A \equiv A$. Hence by Proposition C.27 there exists $w \in \mathbb{T}(A)$ such that $w \sim u_0 \cdot v$ and $(u_0 \cdot v)\bar{\pi} \overset{\sigma}{\cong} w$ for some σ such that $\sigma_{|L} = id$. In particular $u \cong t \cdot w$ by Proposition C.26 and $t \cdot w$ is therefore of the form $(\star)$. Hence by hypothesis there exists $s' \in \mathbb{T}(Q)$ such that $s' \sim t \cdot w$ and σ' a session matching for $t \cdot w$ and s'. We also let σ'_t and σ'_w the restrictions of σ' to the labels of t and w, respectively. If we decompose $s' = s'_t \cdot s'_w$ with $|s'_t| = |t|$ then σ'_t (resp. σ'_w) is a session matching for t and s'_t (resp. w and s'_w). In particular, $\sigma'_w \circ \sigma$ is a session matching for $(u_0 \cdot v)\bar{\pi}$ and s'_w. Thus $\phi = \sigma_w \circ \sigma \circ \bar{\pi}$ is a session matching for $u_0 \cdot v$ and s'_w.

Besides by hypothesis $\pi^{-1} \in \mathsf{Sym}_1^Q$ because Sym_1 is a group; in terms of session matchings this assumption can be rephrased as follows:

for all $z : Q \overset{\mathsf{tr}}{\Rightarrow} Z$ and ω session matching for t and z, there exists $z' : Q \overset{\mathsf{tr}}{\Rightarrow} Z$ and ω'

session matching for t and z' such that $\omega'_{|L(Z)} = \omega_{|L(Z)} \circ \bar{\pi}$, where $L(Z)$ is the set of labels of Z.

In particular for $z = s'_t$ and $\omega = \sigma'_t$ we obtain $s''_t \in \mathbb{T}(Q)$ and σ''_t session matching for t and s''_t such that for all $i \in [\![1, n]\!]$,

$$\begin{aligned} \sigma''_t(\ell_i) &= \sigma'_t(\ell_{\pi(i)}) \\ &= \sigma'_w(\ell_{\pi(i)}) && \text{(because } \sigma'_t \text{ and } \sigma'_w \text{ coincide on } L) \\ &= \phi(\ell_i) && \text{(because } \sigma_{|L} = id) \end{aligned}$$

This justifies that the following mapping ψ of domain $dom(\sigma''_t) \cup dom(\phi)$ is well defined

$$\psi_{|dom(\sigma''_t)} = \sigma''_t \qquad\qquad \psi_{|dom(\phi)} = \phi$$

All in all it sufficies to choose $s = s''_t \cdot s'_w$ since ψ is a session matching for u and s, and $s \sim u$ because

$$actions(s) = actions(s'_t) \cdot actions(w) = actions(t) \cdot actions(u_0 \cdot v) = actions(u) \qquad \square$$

***Part 2**: Proof in the case $\pi \in \mathsf{Sym}_2$.* We use the same initial notations for u and u_0 as in the previous Part, with in particular

$$(u_0 \cdot v)\bar{\pi} : (\pi^{-1}.A \xrightarrow{[\,a\,]^{\ell\bar{\pi}}} B) \cdot v\bar{\pi}$$

But by hypothesis $\pi \in \mathsf{Sym}_2$, or equivalently $\pi^{-1} \in \mathsf{Sym}_2$ since Sym_2 is a group. Therefore there exists ϱ permutation of $\mathcal{F}_0$ such that $\pi^{-1}.A \equiv A\varrho$. Hence by Proposition C.27 there exists $w \in \mathbb{T}(A\varrho)$ such that $w \sim u_0 \cdot v$ and $u_0 \cdot v \overset{\sigma_a}{\cong} w$ for some σ_a such that $\sigma_{a|L} = id$. In particular $u \cong t \cdot w\varrho^{-1}$ by Proposition C.26 and $t \cdot w\varrho^{-1}$ is therefore of the form $(\star)$ (and its first action is $[\,a'\varrho^{-1}\,]^{\ell\bar{\pi}}$). Hence by hypothesis there exists $s' \in \mathbb{T}(Q)$ such that $s' \sim t \cdot w\varrho^{-1}$ and σ' a session matching for $t \cdot w\varrho^{-1}$ and s'. We also let σ'_t and σ'_w the restrictions of σ' to the labels of t and w, respectively. Decomposing $s' = s'_t \cdot s'_w$ with $|s'_t| = |t|$, σ'_t (resp. σ'_w) is a session matching for t and s'_t (resp. $w\varrho^{-1}$ and s'_w). Hence we have established so far:

- σ'_t is a session matching for t and s'_t
- $\sigma'_w \circ \sigma_a \circ \bar{\pi}$ is a session matching for $u_0 \cdot v$ and $s'_w\varrho$

However this is not sufficient to construct a session matching for $u = t \cdot u_0 \cdot v$, among other things because the last extended process of s'_t (let us write it Z) is not the same as the first extended process of $s'_w\varrho$ (which is $Z\varrho$). This is where we use the hypothesis $\pi^{-1} \in \mathsf{Sym}_2^Q$, which implies that $\pi^{-1} \in \mathsf{Stab}_\varrho(Z)$. In this instance of $\mathsf{Stab}_\varrho(Z)$, the implicit ordering of the labels of Z is the one mirroring the ordering on the labels of A: namely the labels of Z are ordered $\sigma'_w(\ell_1), \sigma'_w(\ell_2) \ldots, \sigma'_w(\ell_n)$. In particular in our context, the hypothesis $\pi^{-1} \in \mathsf{Stab}_\varrho(Z)$ means:

$$(\pi^{-1}.(Z\sigma'^{-1}_w))\sigma'_w \equiv Z\varrho$$

or more succinctly:

$$Z\phi \equiv Z\varrho^{-1} \qquad\qquad \text{with } \phi = \sigma'_w \circ \bar{\pi}^{-1} \circ \sigma'^{-1}_w$$

Hence by applying Proposition C.27 to the trace $s'_w\phi$, we obtain $s''_w \in \mathbb{T}(Z\varrho^{-1})$ such that $s''_w \sim s'_w$ and σ_z a session matching for $s'_w\phi$ and s''_w such that $\sigma_{z|L(Z)} = id$ for $L(Z) = \{\sigma'_w(\ell) \mid \ell \in L\}$ the set of labels of Z. Therefore $\sigma_z \circ \phi$ is a session matching for $s'_w\varrho$ and $s''_w\varrho \in \mathbb{T}(Z)$. To conclude we choose $s = s'_t \cdot s''_w\varrho$. To define the session matching for u and s we consider

$$\psi_1 = \sigma'_t \qquad\qquad \psi_2 = \sigma_z \circ \phi \circ \sigma'_w \circ \sigma_a \circ \bar{\pi} = \sigma_z \circ \sigma'_w \circ \bar{\pi}^{-1} \circ \sigma_a \circ \bar{\pi}$$

We have established that ψ_1 is a session matching for t and s'_t, and that ψ_2 is a session matching for $u_0 \cdot v$ and s''_w. Therefore a session matching for u and s would be the matching ψ such that $dom(\psi) = dom(\psi_1) \cup dom(\psi_2)$ and $\psi_{dom(\psi_1)} = \psi_1$ and $\psi_{dom(\psi_2)} = \psi_2$. This function can be defined if ψ_1 and ψ_2 coincide on $dom(\psi_1) \cap dom(\psi_2) = L$. And indeed, for all $i \in [\![1, n]\!]$,

$$\begin{aligned}
\psi_2(\ell_i) &= \sigma_z \circ \sigma'_w \circ \bar{\pi}^{-1} \circ \sigma_a(\ell_{\pi(i)}) && \text{(by definition)}\\
&= \sigma_z \circ \sigma'_w \circ \bar{\pi}^{-1}(\ell_{\pi(i)}) && \text{(because } \sigma_{a|L} = id\text{)}\\
&= \sigma_z \circ \sigma'_w(\ell_i)\\
&= \sigma'_w(\ell_i) && \text{(because } \sigma_{z|L(Z)} = id\text{)}\\
&= \psi_1(\ell_i) && \text{(because } \sigma_{t'|L} = \sigma_{w'|L}\text{)}
\end{aligned}$$

Finally we also verify that $s \sim u$, in particular $actions(s) = actions(u)$:

$$actions(s) = actions(t)\cdot actions(s'_w\varrho) = actions(t)\cdot actions(w) = actions(t)\cdot actions(u_0\cdot v) = actions(u)$$

□

***Part 3**: General case.* That is, $\pi = \pi_1 \circ \cdots \circ \pi_n$ for some $\pi_1, \ldots, \pi_n \in \mathsf{Sym}_1 \cup \mathsf{Sym}_2$. For succinctness if $w \in \mathbb{T}(P)$ we write $H(w)$ the property

> *There exists $s \in \mathbb{T}(Q)$ such that $actions(s) = actions(w)$ and a session matching for w and s.*

In particular the property we attempt to prove is that for all $\pi \in \mathsf{Sym}$, assuming that $H(u')$ for all $u' \cong u$ of the form $(\star)$, we have $H(u)$. We proceed by induction on a proof that $\pi \in \mathsf{Sym}$.

▷ *case 1: $\pi = id$.*

Since Sym_1 is a group, it contains the identity and this case is already captured by the *Part 1* of the proof.

▷ *case 2: $\pi = \phi \circ \psi$ with $\phi \in \mathsf{Sym}_1 \cup \mathsf{Sym}_2$, $\psi \in \mathsf{Sym}$.*

To apply the induction hypothesis to ψ (and thus conclude the proof) we have to prove that $H(u'')$ for all traces $u'' \cong u$:

$$u'' : t \cdot (A \xrightarrow{[a'']^{\ell\psi}} B'') \cdot v'' .$$

Let u'' be such a trace. Using either **Part 1** or **Part 2** (depending on whether $\phi \in \mathsf{Sym}_1$ or

$\phi \in \mathsf{Sym}_2$) we know that it sufficies to prove that $H(u')$ for all $u' \cong u''$ of the form

$$u' : t \cdot (A \xrightarrow{[a']^{\ell\bar{\pi}}} B') \cdot v' \,.$$

Since $\cong$ is transitive all such u' also verify $u' \cong u$, hence the conclusion by hypothesis. □

This is the core technical lemma to justify the correctness of our universal symmetry.

Compatibility with POR So far we have provided the core argument to prove that $\mathbb{O}^{\forall}_{\mathsf{sym}}$ is a correct refinement of $\mathbb{O}^{\forall}_{\mathsf{c+i^*}}$. However it is more complex to prove it compatible with $\mathbb{O}^{\forall}_{\mathsf{por}}$, i.e. with high-priority transitions and the lexicographic reduction. In essence we will have to reuse the core corectness arguments developed in Appendices 3.5 and 3.6 to prove their correctness, but with the handicap of applying symmetries at the same time.

Proposition 5.30

$\mathbb{O}^{\forall}_{\mathsf{por}} \cap \mathbb{O}^{\forall}_{\mathsf{sym}}$ is a correct refinement of $\mathbb{O}^{\forall}_{\mathsf{por}}$.

Proof. Since the reasoning will be the same anyway, we prove the stronger result that $\mathbb{O}^{\forall}_{\mathsf{sym}} \cap \mathbb{O}^{\forall}_{\mathsf{por}}$ is a correct refinement of $\mathbb{O}^{\forall}_{\mathsf{c+i^*+0}}$. We let $\sqsubseteq_{\mathsf{sym}}$ and $\sqsubseteq_{\mathsf{0}}$ the notions of inclusion respectively induced by these two optimisations and show that they coincide. The inclusion $\sqsubseteq_{\mathsf{0}} \subseteq \sqsubseteq_{\mathsf{sym}}$ is immediate. Regarding the converse inclusion, let P, Q two processes such that $P \sqsubseteq_{\mathsf{sym}} Q$ and let us prove that $P \sqsubseteq_{\mathsf{0}} Q$. For that we prove that for all traces $u \in \mathbb{T}(P) \cap \mathbb{O}^{\forall}_{\mathsf{c+i^*+0}}$, there exists $u^2 \in \mathbb{T}(P, Q)$ such that $u = \mathsf{fst}(u^2) \sim \mathsf{snd}(u^2)$. We proceed by well founded induction w.r.t. the ordering $\preccurlyeq_{lex}$ on compressed traces.

▷ *case 1:* $u \notin \mathbb{O}^{\forall}_{\mathsf{por}}$

We consider an arbitrary maximal extension $u' \in \mathbb{O}^{\forall}_{\mathsf{c}}$ obtained by executing arbitrary blocks after u as long as possible. Then by Propositions 5.21, 5.24, 5.26 there exists $\bar{u} \equiv_{\mathsf{b\text{-}por}} u'$ such that $\bar{u} \in \mathbb{O}^{\forall}_{\mathsf{c+i^*+0}}$. Since $\bar{u}$ is maximal, by Proposition 5.28 there exists π permuting independent blocks of $\bar{u}$ such that $\pi.\bar{u} \in \mathbb{O}^{\forall}_{\mathsf{por}} \subseteq \mathbb{O}^{\forall}_{\mathsf{c+i^*+0}}$ and $\pi.\bar{u} \prec_{lex} u$. In particular by applying the induction hypothesis to $\pi.\bar{u}$ we obtain u^2 such that $\pi.\bar{u} = \mathsf{fst}(u^2) \sim \mathsf{snd}(u^2)$. Hence the conclusion by Proposition C.10, since $\pi.\bar{u} \equiv_{\mathsf{b\text{-}por}} u$.

▷ *case 2:* $u \in \mathbb{O}^{\forall}_{\mathsf{por}} \cap \mathbb{O}^{\forall}_{\mathsf{sym}}$

Then the conclusion follows from the hypothesis $P \sqsubseteq_{\mathsf{sym}} Q$.

▷ *case 3:* $u \in \mathbb{O}^{\forall}_{\mathsf{por}} \setminus \mathbb{O}^{\forall}_{\mathsf{sym}}$

Let us write $u = u_1 \cdot b \cdot u_2$ with $u_1, u_2 \in \mathbb{O}^{\forall}_{\mathsf{c}}$ and $b \in \mathbb{T}(A)$ a block such that $b \notin \mathbb{O}^{\forall}_{\mathsf{sym}}$. Let $\ell_1, \ldots, \ell_k$ be the labels A and ℓ the label of its first action.

▷ *case 3a:* the first action of b is an input.

We write $\ell = \ell_{i_0}$. Referring to the notations of the definition of $\mathbb{O}^{\forall}_{\mathsf{sym}}$ we let $\pi \in \mathsf{Sym}$ such that

$$\pi(i_0) = \min \mathsf{Orb}(i_0) \,.$$

The conclusion will follow from Proposition C.31 provided we manage to comply with its hypothesis. Thus let $u' \cong u$ a trace of the following form

$$u' : u_1 \cdot b' \cdot u'_2$$

where the first action of $b' \in \mathbb{T}(A)$ is an input on the label $\ell_{\pi(i_0)}$. We recall that $u' \in \mathbb{O}^{\forall}_{\mathsf{c+i^*}}$ by Proposition C.28. Besides by hypothesis $u_1 \cdot b \in \mathbb{O}^{\forall}_{\mathsf{c+i^*+0}}$, therefore no high-priority transitions are available at the start of b, i.e. in A (since the first transition of b is an input and can therefore not be high-priority). Therefore $u_1 \cdot b' \in \mathbb{O}^{\forall}_{\mathsf{c+i^*+0}}$. On the other hand there are no guarantees that high-priority transitions are respected in u'_2; however by Proposition 5.26 we can fix π permuting independent blocks of u'_2 such that $\pi.u'_2 \in \mathbb{O}^{\forall}_{\mathsf{c+i^*+0}}$. In particular if $u'' = u_1 \cdot b'\pi.u'_2$ we have $u'' \in \mathbb{O}^{\forall}_{\mathsf{c+i^*+0}}$ but also and $u'' \prec_{lex} u$ because $\mathit{actions}(b') \prec \mathit{actions}(b)$ by definition of π. By the induction hypothesis applied to u'' we therefore obtain u^2 such that $u'' = \mathsf{fst}(u^2) \sim \mathsf{snd}(u^2)$. Using Proposition C.10 we then obtain v^2 such that $u' = \mathsf{fst}(v^2) \sim \mathsf{snd}(v^2)$. Hence the expected conclusion by Proposition C.1.

▷ *case 3b:* the first action of b is an internal communication.

We write $\ell = \ell_{i_0} \mid \ell_{i_1}$ where the label of the input is ℓ_{i_0}. Referring to the notations of the definition of $\mathbb{O}^{\forall}_{\mathsf{sym}}$ we let $\pi \in \mathsf{Sym}$ such that

$$(\pi(i_0), \pi(i_1)) = \min \mathsf{Orb}(i_0, i_1)\,.$$

Like the previous case we want to conclude by using Proposition C.31. Let $u' \cong u$ a trace of the following form

$$u' : u_1 \cdot b' \cdot u'_2$$

where the first action of b' is on the label $\ell_{\pi(i_0)} \mid \ell_{\pi(i_1)}$. We recall that by definition $\mathsf{Orb}(i_0, i_1) \subseteq \mathit{IO}$ and therefore that an internal communication at label $\ell_{\pi(i_0)} \mid \ell_{\pi(i_1)}$ respects high-priority transitions. In particular $u_1 \cdot b' \in \mathbb{O}^{\forall}_{\mathsf{c+i^*+0}}$ (using Proposition C.28 again). Besides, using a similar reasoning as in the case 3a we also obtain that $u' \in \mathbb{O}^{\forall}_{\mathsf{c+i^*}}$ and a π permuting independent blocks of u'_2 such that $u'' = u_1 \cdot b' \cdot \pi.u'_2 \in \mathbb{O}^{\forall}_{\mathsf{c+i^*+0}}$. The conclusion of the reasoning is then similar, applying the induction hypothesis to u'' and then Propositions C.10 and C.1. □

4.3 Existential symmetries

In this section we prove the existential optimisation:

Proposition 5.32

$\mathbb{O}^{\exists}_{\mathsf{sym}}$ is a correct refinement of $\mathbb{O}^{\exists}_{\mathsf{all}}$.

However although the formalism of twin traces makes the formalisation of the optimisation more conceit as a simple restriction of rule (Match), we found it easier to prove it correct using the formalism of session matchings; that is, the optimisation rather consists of reducing the potential number of session matchings considered for a trace. Let us thus first state it in this paradigm.

Let $t \in \mathbb{T}(P)$ a trace and an instance of rule (Par) in t:

$$(\{\!\{[\textstyle\prod_{i=1}^{n} P_i\,]^{\ell}\}\!\} \cup \mathcal{P}, \Phi) \xrightarrow{\tau} (\{\!\{[\,P_i\,]^{\ell \cdot i}\}\!\} \cup \mathcal{P}, \Phi)$$

We also let $t' \in \mathbb{T}(Q)$ such that $\mathit{actions}(t) = \mathit{actions}(t')$ and assume that there exists a session matching σ for t and t'. In particular t' contains a unique application of rule (Par) of the form

$$(\{\!\{[\textstyle\prod_{i=1}^{n} Q_i\,]^{\sigma(\ell)}\}\!\} \cup \mathcal{Q}, \Psi) \xrightarrow{\tau} (\{\!\{[\,Q_i\,]^{\sigma(\ell)\cdot\pi(i)}\}\!\} \cup \mathcal{Q}, \Psi) = \pi^{-1}.A$$

for some $\pi \in S_n$ and $\sigma(\ell \cdot i) = \sigma(\ell) \cdot \pi(i)$. Then we write

$$\pi \sim \pi' \quad \textit{iff} \quad \exists u \in \mathsf{Stab}(A), \pi' = \pi \circ u$$

and say that σ is *well formed on* ℓ if π is minimal in its equivalence class for $\sim$ (w.r.t. an arbitrary fixed ordering on S_n). We say that σ is *well formed* when it is well formed on all labels ℓ on which an application of rule (Par) has been performed in t. In particular we obtain the following characterisation of $\mathbb{O}^{\exists}_{\mathsf{sym}}$ by Proposition C.1:

Proposition C.32

The following statements are equivalent for all processes P, Q

1. $\forall t \in \mathbb{T}(P), \exists t^2 \in \mathbb{T}(P,Q) \cap \mathbb{O}^{\exists}_{\mathsf{sym}}, t = \mathsf{fst}(t^2) \sim \mathsf{snd}(t^2)$
2. for all $t \in \mathbb{T}(P)$, there exists $t' \in \mathbb{T}(Q)$ and σ a well formed session matching for t and t' such that $t \sim t'$

In the remaining of the section we therefore focus on proving that the second item is equivalent to $P \sqsubseteq_s Q$. For that we rely again on the results of Section 4.1 in this appendix.

Proposition C.33

The following statements are equivalent for all processes P, Q

1. $P \sqsubseteq_s Q$
2. for all $t \in \mathbb{T}(P)$, there exists $t' \in \mathbb{T}(Q)$ and σ a well formed session matching for t and t' such that $t \sim t'$

Proof. The implication $2 \Rightarrow 1$ is immediate. Conversely we assume 1, let $t \in \mathbb{T}(P)$ and thus $t' \in \mathbb{T}(Q)$ such that $t \sim t'$ and σ_0 a session matching for t and t'. We now construct $s \in \mathbb{T}(Q)$ and σ a well formed session matching for t and s. We proceed by (decreasing) induction on the number of transitions of t before the first (Par) transition on a label ℓ the matching σ_0 is not well formed on. If there are none σ_0 is well formed and it sufficies to choose $s = t'$ and $\sigma = \sigma_0$. Otherwise let ℓ be the first such label. We write $t = t_0 \cdot p \cdot t_1$ with p the transition

$$(\{\!\{[\prod_{i=1}^n P_i]^\ell\}\!\} \cup \mathcal{P}, \Phi) \xrightarrow{\tau} (\{\!\{[P_i]^{\ell \cdot i}\}\!\} \cup \mathcal{P}, \Phi)\,.$$

Similarly we write $t' = t'_0 \cdot p' \cdot t'_1$ with p' the transition

$$(\{\!\{[\prod_{i=1}^n Q_i]^{\sigma_0(\ell)}\}\!\} \cup \mathcal{Q}, \Psi) \xrightarrow{\tau} (\{\!\{[Q_i]^{\sigma_0(\ell)\cdot\pi(i)}\}\!\} \cup \mathcal{Q}, \Psi) = \pi^{-1}.A\,.$$

We let $u \in \mathsf{Stab}(A)$ such that $\pi_0 = \pi \circ u$ is minimal in its equivalence class for $\sim$. By definition of $\mathsf{Stab}(A)$ we have in particular $\pi^{-1}.A \equiv \pi_0^{-1}.A$. Therefore by Proposition C.27 we know that there exists $s_1 \in \mathbb{T}(\pi_0^{-1}.A)$ and a session matching σ_1 for t'_1 and s_1 such that $t'_1 \sim s_1$ and $\sigma_1(\ell') = \ell'$ for all labels $\ell' \in L$ the set of labels appearing in A. In particular we consider the trace

$$s' = t'_0 \cdot ((\{\!\{[\prod_{i=1}^n Q_i]^{\sigma_0(\ell)}\}\!\} \cup \mathcal{Q}, \Psi) \xrightarrow{\tau} \pi_0^{-1}.A) \cdot s_1$$

and σ' the label permutation defined by

$$\begin{aligned} \sigma'(\ell') &= \sigma_0(\ell') && \text{if } \ell' \text{ label of } t_0 \cdot p \\ \sigma'(\ell') &= \sigma_1 \circ \sigma_0(\ell') && \text{if } \ell' \text{ label of } t_1 \end{aligned}$$

Since σ_0 and σ_1 coincide on L, σ' is well defined a session matching for t and s'. Besides it is well formed on ℓ (and on all labels appearing before in the trace t). Hence we can apply the induction hypothesis to s' and σ' which gives the expected trace s and session matching σ. □

Appendix D:

Proofs of Chapter 6

Summary.

In this appendix we prove the technical results used in Chapter 6 for the termination and complexity of the decision procedure for trace equivalence and labelled bisimilarity.

1 Termination

1.1 For mgs

The termination of the computation of most general solutions mostly relied on the following result, yet to be proved:

Proposition 6.2 (decrease of unused first-order terms during constraint solving)

Let $\mathcal{C}^e$ be an extended constraint system such that $\mathcal{C}^e\{\!\!\} = \mathcal{C}^e$ and the invariants $\mathsf{Inv}_{wf}(\mathcal{C}^e)$ and $\mathsf{Inv}_{sound}(\mathcal{C}^e)$ hold. Then let $\mathcal{C}^e \xrightarrow{\Sigma}\{\!\!\} \mathcal{C}^{e\prime} \neq \bot$. If this transition is derived with:

1 Rule (MGS-Conseq): $|unused^1(\mathcal{C}^{e\prime})| \leqslant |unused^1(\mathcal{C}^e)|$

2 Rules (MGS-Res) or (MGS-Cons): $|unused^1(\mathcal{C}^{e\prime})| < |unused^1(\mathcal{C}^e)|$

Proof. Consider first the simplification rule (MGS-Unif) and the ones from Figure 4.1. They typically apply protocol term substitutions on the constraint system (they also effect recipe disequations that are irrelevant in $unused^1(\mathcal{C}^e)$). Note that the applied substitution is always generated from terms already in the constraint system. As such $\mu^1(\mathcal{C}^e{\downarrow}) = \mu^1(\mathcal{C}^e)$ and so $\Phi(\mathcal{C}^e{\downarrow})\mu^1(\mathcal{C}^e{\downarrow}) = \Phi(\mathcal{C}^e)\mu^1(\mathcal{C}^e)$, $\mathsf{K}(\mathcal{C}^e{\downarrow})\mu^1(\mathcal{C}^e{\downarrow}) = \mathsf{K}(\mathcal{C}^e)\mu^1(\mathcal{C}^e)$ and $\mathsf{D}(\mathcal{C}^e{\downarrow})\mu^1(\mathcal{C}^e{\downarrow}) = \mathsf{D}(\mathcal{C}^e)\mu^1(\mathcal{C}^e)$. Thus, we directly obtain that $|unused^1(\mathcal{C}^e{\downarrow})| \leqslant |unused^1(\mathcal{C}^e)|$.

Let us look at Rules (MGS-Conseq), (MGS-Cons) and (MGS-Res) and let us consider $\mathcal{C}^e \xrightarrow{\Sigma} \mathcal{C}'^e$. The rule (MGS-Conseq) does not modify the protocol terms of the constraint systems by apply a recipe substitution. However, we show an invariant on the constraint systems that any $\xi, \zeta \in subterms_{\mathsf{c}}(\mathcal{C}^e)$ are consequence of $\mathsf{K}\cup\mathsf{D}$ as well as any of their subterms (see Definition B.1 in Appendix). Thus, we deduce from the definition of $subterms_{\mathsf{c}}(\mathcal{C}^e)$ that $subterms_{\mathsf{c}}(\mathcal{C}^e)\Sigma \subseteq subterms_{\mathsf{c}}(\mathcal{C}'^e)$. To conclude that $|unused^1(\mathcal{C}'^e)| \leqslant |unused^1(\mathcal{C}^e)|$, we rely on the technical Proposition B.5; in other words, if $(\xi, t) \in \mathsf{Conseq}(\mathsf{K}(\mathcal{C}^e)\mu^1(\mathcal{C}^e) \cup \mathsf{D}(\mathcal{C}^e)\mu^1(\mathcal{C}^e))$ then $(\xi\Sigma, t) \in \mathsf{Conseq}(\mathsf{K}(\mathcal{C}'^e)\mu^1(\mathcal{C}'^e) \cup \mathsf{D}(\mathcal{C}'^e)\mu^1(\mathcal{C}'^e))$. Since $subterms_{\mathsf{c}}(\mathcal{C}^e)\Sigma \subseteq subterms_{\mathsf{c}}(\mathcal{C}'^e)$, we conclude that $|unused^1(\mathcal{C}'^e)| \leqslant |unused^1(\mathcal{C}^e)|$.

By applying the same reasonning for the rule (MGS-Res), we can also show that $unused^1(C'^e) \leqslant unused^1(\mathcal{C}^e)$. However, we can even show that this inequality is strict. Indeed, using the same the notation in the rule (MGS-Res), this rule is only applied if $\mathcal{C}^e = \mathcal{C}^e{\downarrow}$ and for all $\xi \in subterms_{\mathsf{c}}(\mathcal{C}^e) \setminus \{X\}$, $(\xi, u) \notin \mathsf{Conseq}(\mathsf{K} \cup \mathsf{D})$. Note that $\mathcal{C}^e = \mathcal{C}^e{\downarrow}$ implies that $\mathsf{K}\mu^1 = \mathsf{K}$ and $\mathsf{D}\mu^1 = \mathsf{D}$. Moreover, it also implies that $u \in unused^1\mathcal{C}^e$. However, in $\mathcal{C}'^e$, we have that $(\xi, u\mu_1(\mathcal{C}'^e)) \in \mathsf{Conseq}(\mathsf{K}(\mathcal{C}'^e)\mu^1(\mathcal{C}'^e) \cup \mathsf{D}(\mathcal{C}'^e)\mu^1(\mathcal{C}'^e))$. Moreover, we show another invariant on the constraint system (see Definition B.1 in Appendix) that ensures us that $X \in subterms_{\mathsf{c}}(\mathcal{C}^e)$ and so $\xi \in subterms_{\mathsf{c}}(\mathcal{C}'^e)$. Hence, we obtain that $u\mu^1(\mathcal{C}'^e) \notin unused^1(\mathcal{C}'^e)$ allowing us to conclude that $unused^1(C'^e) < unused^1(C^e)$. By applying the same reasoning, we can also show that $unused^1(C'^e) < unused^1(C^e)$ when the rule (MGS-Cons) is applied. □

1.2 Exponential measure

Another argument left pending is that each component of the measure except the last one can be bounded by an exponential in the DAG size of the parameters of the problem. We give a bound for each of them, in particular relying on the bound on $unused^1$ proved in the body of thesis.

1. $M_1(\Gamma) \leqslant |P, Q|_{\mathsf{dag}}$:
   by definition.
2. $M_2(\Gamma) \leqslant |P, Q|_{\mathsf{dag}}$:
   the number of non-deducibility facts is bounded by the maximal number of internal communications possible in a trace, itself bounded by the sizes of the processes.
3. $M_3(\Gamma) \leqslant (|P|_{\mathsf{dag}}\,|E|_{\mathsf{dag}})^{|P|_{\mathsf{dag}}} + (|Q|_{\mathsf{dag}}\,|E|_{\mathsf{dag}})^{|Q|_{\mathsf{dag}}}$:
   The measure corresponds to the number of symbolic transitions possible from P and Q for a given symbolic trace, hence the bound. Notice that the part $|E|_{\mathsf{dag}}^{|P|_{\mathsf{dag}}}$ is due to the computation of the most general unifiers modulo E in the symbolic transitions.
4. $M_4(\Gamma) \leqslant 9\,|P, Q, E|_{\mathsf{dag}}^3$:
   It sufficies to observe that $\mathsf{set}_{\mathsf{K}}(\mathcal{C}^e) \leqslant unused^1(\mathcal{C}^e)$ and to use the bound proved in the body of the thesis.
5. $M_5(\Gamma) \leqslant M_3(\Gamma)$:
   Trivial.
6. $M_6(\Gamma) \leqslant M_3(\Gamma) \times |E|_{\mathsf{dag}}^{|E|_{\mathsf{dag}}} \times (18\,|P, Q, E|_{\mathsf{dag}})^{27|P,Q,E|_{\mathsf{dag}}^3}$:
   Bounding the size of $|\mathsf{set}_{\mathrm{Rew}}(\mathcal{C}^e)|$ can easily be done: the number of $\psi \in \mathsf{K}$ possible is bounded by $|\mathsf{K}|$, itself bounded by $|\mathsf{set}_{\mathsf{K}}(\mathcal{C}^e)|$. The number of rewrite rules, position p and $\psi_0 \in \mathsf{RewF}(\xi, \ell \to r, p)$ only depends on the rewrite systems and can be bounded by $|E|_{\mathsf{dag}}^{|E|_{\mathsf{dag}}}$. Note that the exponential comes mainly from the number of possible positions in ℓ. We already know that the number of most general solutions is bounded by $(|\mathsf{K}(\mathcal{C}^e)| + 1)^{unused^1(\mathcal{C}^e)}$. Combining with all previous results, and with the rough approximation $9\,|P, Q, E|_{\mathsf{dag}}^3 + 1 \leqslant 18\,|P, Q, E|_{\mathsf{dag}}^3$, we obtain the above bound.
7. $M_7(\Gamma) \leqslant |E| \times M_3(\Gamma)$:
   To bound this number, we need to recall that we always apply the case distinction rules with the priority ordering (Sat) $<$ (Rew). Thus, when we apply a rule (Rew), there is no unsolved deduction formula in any of the extended constraint systems (otherwise we should have applied the rule (Sat)). It means this measure is bounded by the number

of deduction formulas produced by one instance of (Rew). By definition, we know that $|\mathsf{RewF}(\xi, \ell \to r, p)| \leqslant |\mathcal{R}|$ (one formula per rewrite rule). Thus, the rule (Rew) generates at most $|E| \times M_3(\Gamma)$ deduction formulas.

8 $M_8(\Gamma) \leqslant M_3(\Gamma) \times 2\,|E|_{\mathsf{dag}}\,(|P,Q|_{\mathsf{dag}})^2(1+|E|_{\mathsf{dag}})^2$:
The application conditions stipulate that the rule can be applied either (a) on two deduction facts of $\mathsf{K}(\mathcal{C}_i^e)$, or (b) on one deduction fact of $\mathsf{K}(\mathcal{C}_i^e)$ in combination with a construction function symbol.
Note that even though the rule also consider the existence of a most general solution $\Sigma \in mgs(\mathcal{C}_i^e[\mathsf{E}^1 \mapsto \mathsf{E}^1 \wedge \mathsf{hyp}(\psi{:}(\Sigma_0, \mathcal{C}_i^e))])$, the number of applications of the rule (Eq) will not depend on the number of possible most general solutions. Indeed, consider the case (a) where the rule is applied on two deduction fact $(\xi_1 \vdash^? u_1), (\xi_2 \vdash^? u_2) \in \mathsf{K}(\mathcal{C}_i^e)$. Thus, an equality formula with $\xi_1\Sigma =_f^? \xi_2\Sigma$ as head will be added in $\mathsf{F}(\mathcal{C}_i^e{:}\Sigma)$. However, in the application conditions of the rule, we also require that *for all* $(\forall S.\, \varphi \Rightarrow H) \in \mathsf{F}(\mathcal{C}_i^e)$, $H \neq (\xi_1 =_f^? \xi_2)$. Thus, a new application of the rule (Eq) on $\mathcal{C}_i^e{:}\Sigma$ with the same (up to instantiation of Σ) deductions facts from $\mathsf{K}(\mathcal{C}_i^e{:}\Sigma)$ will be prevented.
The same situation occurs in case (b) with the condition *for all* $(\forall S.\, \varphi \Rightarrow \zeta_1 =_f^? \zeta_2) \in \mathsf{F}(\mathcal{C}_i^e)$, $\zeta_1 = \xi_1$ *implies* $root(\zeta_2) \neq \mathsf{f}$. We therefore conclude that the rule (Eq) can be applied only once per pair of deduction facts in K and once per deduction fact in K and function symbol in $\mathcal{F}_\mathsf{c}$.

9 $M_9(\Gamma) \leqslant M_3(\Gamma)$:
Unsolved equality formulas can be generated by two rules: the case distinction rule (Eq) or the simplification Rule Vect-add-Formula. However, once again because of the priority order (Sat) $<$ (Eq), the two rules cannot be triggered simultaneously and the rule (Eq) is only triggered when there is no unsolved equality formulas. Note that due to the condition $\forall i.\forall(\forall S.\, \varphi \Rightarrow \zeta_1 =_f^? \zeta_2) \in \mathsf{F}_i$, $\zeta_1 \neq \xi$ *or* $\zeta_2 \neq \xi$ in Rule Vect-add-Formula, two instances of the Rule Vect-add-Formula with different recipes ζ (e.g. if u_1 can be deducible with two different recipes) cannot be applied sequentially. Thus, at any given moment, there is at most one unsolved equality formula per extended constraint system of Γ, hence the bound.

1.3 Bounding the increase of second order terms

In Section 2, Proposition 6.7, we gave a bound on the increase of the size of most general solutions when applying the rule (Sat). We give here the arguments to extend to the other case distinction rules. For that it sufficies to generalise this property to a more general set of substitution Σ:

Definition D.1

Let $\mathcal{C}^e$ be an extended constraint system. Let Σ be a second-order substitution. We say that $\Sigma \in \mathsf{CompatSubs}(\mathcal{C}^e)$ if $dom(\Sigma) \subseteq vars^2(\mathsf{D}(\mathcal{C}^e))$ and for all $X \in dom(\Sigma)$, $X\Sigma \in \mathsf{Conseq}(\mathsf{K}(\mathcal{C}^e) \cup \mathsf{D}' \cup D_\Sigma)$ where $\mathsf{D}' = \{X \vdash^? u \in \mathsf{D}(\mathcal{C}^e) \mid X \notin dom(\Sigma)\}$ and $D_\Sigma = \{X \vdash^? x \mid x \text{ fresh and } X \in vars^2(\Sigma) \setminus vars^2(\mathcal{C}^e)\}$.

Intuitively, $\mathsf{CompatSubs}(\mathcal{C}^e)$ represents the recipe substitutions Σ that can be applied be applied to the constraint system $\mathcal{C}^e$, i.e. $\mathcal{C}^e{:}\Sigma$, and such that the recipes in the of Σ would be consequence of $\mathcal{C}^e{:}\Sigma$. Note that $mgs(\mathcal{C}^e) \subseteq \mathsf{CompatSubs}(\mathcal{C}^e)$.

By applying Proposition B.5, we can show that:

$$\text{for all } \Sigma \in \mathsf{CompatSubs}(\mathcal{C}^e), |unused^1(\mathcal{C}^e{:}\Sigma)| \leqslant |unused^1(\mathcal{C}^e)| \tag{D.1}$$

Note that in a set of symbolic processes two extended constraint systems $\mathcal{C}_1^e, \mathcal{C}_2^e$ always have the same *recipe structure* (Invariant Inv_{str}), i.e. $|\Phi(\mathcal{C}_1^e)| = |\Phi(\mathcal{C}_2^e)|$, $vars^2(\mathcal{C}_1^e) = vars^2(\mathcal{C}_2^e)$ and $\{\xi \mid (\xi \vdash^? u) \in \mathsf{K}(\mathcal{C}_1^e)\} = \{\xi \mid (\xi \vdash^? u) \in \mathsf{K}(\mathcal{C}_2^e)\}$. Thus, we deduce that $\mathsf{CompatSubs}(\mathcal{C}_1^e) = \mathsf{CompatSubs}(\mathcal{C}_2^e)$. Therefore, we can conclude that for any simplification and case distinction rules, $|unused^1(\mathcal{C}^e)|$ never increase for all extended constraint systems in a set of extended symbolic processes.

2 Complexity lower bounds

2.1 Advanced winning strategies

Before starting the proofs, we present some characterizations of observational (in)equivalence in order to make the incoming proofs easier to handle.

▶ *Remark*
The results of this section (2.1) also apply to the extended syntax and semantics of Chapter 6, Section 4.1. ◀

For the defender The transitions of the semantics which are deterministic and silent are not essential to equivalence proofs as they do not interfere substantially with them. We introduce below a refined proof technique to rule them out.

Definition D.2 (simplification)
A multiset of closed plain processes $\mathbb{S}$ is silent in an extended process $(\mathcal{P}, \Phi)$ when for all transitions $(\mathcal{P} \cup \mathbb{S}, \Phi) \xrightarrow{\alpha} (\mathcal{Q}, \Phi')$, it holds that $\mathcal{Q} = \mathcal{P}' \cup \mathbb{S}$ with $(\mathcal{P}, \Phi) \xrightarrow{\alpha} (\mathcal{P}', \Phi')$ and $\mathbb{S}$ silent in $(\mathcal{P}', \Phi')$. Then we define $\rightsquigarrow$ (simplification relation) the relation on extended processes defined by the following inference rules:

$$\frac{\mathbb{S} \text{ silent in } (\mathcal{P}, \Phi)}{(\mathcal{P} \cup \mathbb{S}, \Phi) \rightsquigarrow (\mathcal{P}, \Phi)} \quad \text{(S-SIL)}$$

$$\frac{c \in \mathcal{N} \qquad \mathsf{msg}t \qquad c \notin names\mathcal{P}, \Phi}{(\mathcal{P} \cup \{\!\{\overline{c}\langle t\rangle.P, c(x).Q\}\!\}, \Phi) \rightsquigarrow (\mathcal{P} \cup \{\!\{P, Q\{x \mapsto t\}\}\!\}, \Phi)} \quad \text{(S-COMM)}$$

$$\frac{A \xrightarrow{\tau} B \text{ by rules NULL, PAR, THEN, ELSE}}{A \rightsquigarrow B} \quad \text{(S-NPTE)}$$

In other words, we write $A \rightsquigarrow B$ when B is obtained from A by removing some silent process or applying a deterministic (in the sense of the confluence lemma below) instance of the transition relation $\xrightarrow{\tau}$. We call $\rightsquigarrow_{\mathsf{pi}}$ the restriction of $\rightsquigarrow$ to the rule (S-NPTE). Their reflexive transitive closures are denoted τ and $\overset{\star}{\rightsquigarrow}_{\mathsf{pi}}$ respectively as usual.

Lemma D.1

If $A \rightsquigarrow B$ (by some rule ϱ_{sil} of the definition of $\rightsquigarrow$) and $A \xrightarrow{\alpha} C$ (by some rule ϱ_{c} of the semantics), then either $B = C$ and $\alpha = \tau$, or there exists D such that $C \rightsquigarrow D$ (by rule ϱ_{sil}) and $B \xrightarrow{\alpha} D$ (by rule ϱ_{c}).

Proof. We make a case analysis on the rule used to obtain the reduction $A \rightsquigarrow B$:

- **case 1** : by rule (S-SIL):
  Then we write $A = (\mathcal{P} \cup \mathbb{S}, \Phi)$, $B = (\mathcal{P}, \Phi)$. By definition of silent processes, the reduction $A \xrightarrow{\alpha} C$ hence gives $C = (\mathcal{P}' \cup \mathbb{S}, \Phi')$ where $(\mathcal{P}, \Phi) \xrightarrow{\alpha} (\mathcal{P}', \Phi') = D$ and $\mathbb{S}$ silent in D. In particular D gives the expected conclusion.
- **case 2** : by rule (S-COMM) or (S-NPTE):
  Then either $B = C$ and the conclusion is immediate, or $B \neq C$ and a quick analysis of the rules of the semantics gives $\mathcal{P}, Q_B, Q'_B, Q_C, Q'_C, \Phi, \Phi'$ such that:

$$A = (\{\!\{Q_B, Q_C\}\!\} \cup \mathcal{P}, \Phi) \quad B = (\{\!\{Q'_B, Q_C\}\!\} \cup \mathcal{P}, \Phi) \qquad C = (\{\!\{Q_B, Q'_C\}\!\} \cup \mathcal{P}, \Phi')$$
$$(\{\!\{Q_B\}\!\}, \Phi) \rightsquigarrow (\{\!\{Q'_B\}\!\}, \Phi) \qquad (\{\!\{Q_C\}\!\}, \Phi) \xrightarrow{\alpha} (\{\!\{Q'_C\}\!\}, \Phi')$$

  and we conclude by choosing $D = (\{\!\{Q'_B, Q'_C\}\!\} \cup \mathcal{P}, \Phi')$. □

Corollary D.2

If $A \tau B$ and $A \xrightarrow{\alpha} C$ then either $B \tau C$ and $\alpha = \tau$, or there exists D such that $C \tau D$ and $B \xrightarrow{\alpha} D$.

Proof. By a straightforward induction on the number of steps of the reduction $A \tau B$. □

Corollary D.3

$\rightsquigarrow_{\mathsf{pi}}$ is convergent.

Proof. The termination of $\rightsquigarrow_{\mathsf{pi}}$ follows from the termination of the whole calculus. As for the local confluence (which sufficies by Newmann's lemma), we observe that by Lemma D.1, if $A \rightsquigarrow_{\mathsf{pi}} B$ and $A \rightsquigarrow_{\mathsf{pi}} C$ then either $B = C$, or there is D such that $B \rightsquigarrow_{\mathsf{pi}} D$ and $C \rightsquigarrow_{\mathsf{pi}} D$: in particular, $B \overset{\star}{\rightsquigarrow}_{\mathsf{pi}} E$ and $C \overset{\star}{\rightsquigarrow}_{\mathsf{pi}} E$ for some $E \in \{C, D\}$. □

In particular, all extended processes A have a unique normal form w.r.t. $\rightsquigarrow_{\mathsf{pi}}$ which will be written $A_{\downarrow_{\mathsf{pi}}}$. This notation is lifted to multiset of processes, writing $\mathcal{P}_{\downarrow_{\mathsf{pi}}}$ (which is consistent since $\rightsquigarrow_{\mathsf{pi}}$ does not modify the frame). With all of this, we eventually gathered all the ingredients to introduce our characterization of bisimilarity:

Definition D.3 (bisimulation up to $\rightsquigarrow$)

A symmetric relation $\mathcal{R}$ on extended processes is then said to be bisimulation up to $\rightsquigarrow$, or a bisimulation up to simplification, when:

- $\mathcal{R} \subseteq \sim$;
- for all extended processes A, B such that $A \mathcal{R} B$, and for all transitions $A \xrightarrow{\alpha} A'$, there

exists $B \stackrel{\alpha}{\Rightarrow} B'$ such that $A'\tau\mathcal{R} \overset{\star}{\backsim} B'$.

Proposition D.4

For all extended processes A and B, $A \approx_l B$ *iff* there exists a bisimulation up to simplification $\mathcal{R}$ such that $A\tau\mathcal{R} \overset{\star}{\backsim} B$.

Proof. The forward implication follows from the fact that $\approx_l$ is a bisimulation up to simplification (by reflexivity of τ). For the converse, let us consider $\mathcal{R}$ a bisimulation up to $\rightsquigarrow$ and prove that it is contained in $\approx_l$. In order to do that, it sufficies to show that:

- $\tau\mathcal{R} \overset{\star}{\backsim}$ is symmetric;
- $(\tau\mathcal{R} \overset{\star}{\backsim}) \subseteq \sim$;
- for all extended processes A, B such that $A\tau\mathcal{R} \overset{\star}{\backsim} B$, if $A \stackrel{\alpha}{\rightarrow} A'$ then there exists $B \stackrel{\alpha}{\Rightarrow} B'$ such that $A'\tau\mathcal{R} \overset{\star}{\backsim} B'$.

These three properties indeed justify that $(\tau\mathcal{R} \overset{\star}{\backsim}) \subseteq \approx_l$ by definition, hence the expected conclusion as $\mathcal{R} \subseteq \tau\mathcal{R} \overset{\star}{\backsim}$ by reflexivity of τ. Yet it appears that the first two points directly follows from the properties of $\mathcal{R}$ and the reflexivity of τ, and we thus only need to prove the third point. Let us therefore consider the following hypotheses and notations:

$$A\tau C \ \mathcal{R} \ D \overset{\star}{\backsim} B \qquad\qquad A \stackrel{\alpha}{\rightarrow} A'$$

and let us exhibit B' such that $B \stackrel{\alpha}{\Rightarrow} B'$ and $A'\tau\mathcal{R} \overset{\star}{\backsim} B'$. Let us consider the two cases induced by the application of Corollary D.2:

- **case 1** : $A'\tau C$ and $\alpha = \tau$
  Then we can choose $B' = B$.
- **case 2** : there exists C' such that $A'\tau C'$ and $C \stackrel{\alpha}{\rightarrow} C'$
  Consequently, since $\mathcal{R}$ is a bisimulation up to simplification, there is D' such that $D \stackrel{\alpha}{\Rightarrow} D'$ and $C'\tau\mathcal{R} \overset{\star}{\backsim} D'$. Then we remark that for all extended processes B_1, B_2, B_3:
  ▷ a transition $B_1 \rightsquigarrow B_2$ with rules (S-NPTE) or (S-COMM) implies $B_1 \stackrel{\tau}{\rightarrow} B_2$;
  ▷ if $B_1 \rightsquigarrow B_2$ with rule (S-SIL) and $B_2 \stackrel{\beta}{\rightarrow} B_3$, then $B_1 \stackrel{\beta}{\rightarrow}\rightsquigarrow B_3$.
  In particular since $B\tau D \stackrel{\alpha}{\Rightarrow} D'$, we have $B \stackrel{\alpha}{\Rightarrow} D''\tau D'$ for some D''. Hence the conclusion by choosing $B' = D''$. □

For the attacker When taking the negation of labelled bisimilarity, we essentially obtain a set of rules for a game whose states are pairs of processes (A, B): an attacker selects a transition and a defender answers by selecting a equivalently-labelled sequence of transitions in the other process.

Definition D.4 (labelled attack)

A relation $\mathcal{S}$ on extended processes is called a labelled attack when for all A, B such that $A\mathcal{S}B$, it holds that:

1 either: $A \not\sim B$
2 or: $\exists A \stackrel{\alpha}{\rightarrow}\overset{\mathsf{tr}}{\Rightarrow} A', \ \forall B \overset{\alpha.\mathsf{tr}}{\Longrightarrow} B', \ A'\mathcal{S}B'$
3 or: $\exists B \stackrel{\alpha}{\rightarrow}\overset{\mathsf{tr}}{\Rightarrow} B', \ \forall A \overset{\alpha.\mathsf{tr}}{\Longrightarrow} A', \ A'\mathcal{S}B'$

Note that labelled attacks are not the direct translation of the above intuition since they allow the attacker to choose several transitions in a row; this intuitively entails no loss of generality since it is equivalent to the attacker selecting some transitions non-adaptatively (i.e. independently of the answer of the defender). Here is the formal statement of correctness:

Proposition D.5

For all extended processes A and B, $A \not\approx_l B$ *iff* there exists a labelled attack $\mathcal{S}$ such that $A\mathcal{S}B$.

Proof. The forward implication is immediate since $\not\approx_l$ is a labelled attack (we can even choose $\mathsf{tr} = \varepsilon$ everytime). Let then $\mathcal{S}$ be a labelled attack such that $A\mathcal{S}B$ and let us prove that $\mathcal{S} \subseteq \not\approx_l$. More precisely, we prove that $\mathcal{S} \subseteq \mathcal{S}' \subseteq \not\approx_l$ for some relation $\mathcal{S}'$. We will construct $\mathcal{S}'$ in such a way that for all A, B extended processes, $A\mathcal{S}'B$ entails:

(i) either: $A \not\sim B$;

(ii) or: $\exists A \xrightarrow{\alpha} A',\ \forall B \stackrel{\alpha}{\Rightarrow} B',\ A'\mathcal{S}B'$;

(iii) or: $\exists B \xrightarrow{\alpha} B',\ \forall A \stackrel{\alpha}{\Rightarrow} A',\ A'\mathcal{S}B'$

The inclusion $\mathcal{S}' \subseteq \not\approx_l$ is indeed clear if this property is verified, hence the expected conclusion provided such a relation $\mathcal{S}'$. We concretely define it as the smallest relation on extended processes saturated by the following inference rules:

$$\frac{A\mathcal{S}B}{A\mathcal{S}'B} \quad (\textsc{Axiom})$$

$$\frac{A\mathcal{S}'B \qquad A \xrightarrow{\alpha} A' \xrightarrow{\alpha'}\stackrel{\mathsf{tr}}{\Rightarrow} A'' \qquad \forall B \stackrel{\alpha.\alpha'.\mathsf{tr}}{\Longrightarrow} B'', A''\mathcal{S}'B'' \qquad B \stackrel{\alpha}{\Rightarrow} B'}{A'\mathcal{S}'B'} \quad (\textsc{Dec-L})$$

$$\frac{A\mathcal{S}'B \qquad B \xrightarrow{\alpha} B' \xrightarrow{\alpha'}\stackrel{\mathsf{tr}}{\Rightarrow} B'' \qquad \forall A \stackrel{\alpha.\alpha'.\mathsf{tr}}{\Longrightarrow} A'', A''\mathcal{S}'B'' \qquad A \stackrel{\alpha}{\Rightarrow} A'}{A'\mathcal{S}'B'} \quad (\textsc{Dec-R})$$

In particular, note that $\mathcal{S} \subseteq \mathcal{S}'$ thanks to the rule (Axiom). As for the two other rules (Dec-L) and (Dec-R), they intuitively decompose sequences $\xrightarrow{\alpha}\stackrel{\mathsf{tr}}{\Rightarrow}$ into atomic transitions in order to switch from points *2.* or *3.* of Definition D.4 to points (ii) or (iii).

Let then A and B be two extended processes such that $A\mathcal{S}'B$. We consider a proof-tree of $A\mathcal{S}'B$ in the inference system above and perform a case analysis on the rule at its root:

- **case 1** (Axiom): $A\mathcal{S}B$.
  As $\mathcal{S}$ is a labelled attack, we apply the case analysis of Definition D.4:
  - ▷ **case 1.a** : $A \not\sim B$.
    Then (i) is satisfied.
  - ▷ **case 1.b** : there exists $A \xrightarrow{\alpha} A' \stackrel{\mathsf{tr}}{\Rightarrow} A''$ such that $A''\mathcal{S}B''$ for all $B \stackrel{\alpha.\mathsf{tr}}{\Longrightarrow} B''$.
    In particular, keeping in mind that $\mathcal{S} \subseteq \mathcal{S}'$ due to the rule (Axiom), we have $A''\mathcal{S}'B''$ for all $B \stackrel{\alpha.\mathsf{tr}}{\Longrightarrow} B''$. Let us then show that the transition $A \xrightarrow{\alpha} A'$ satisfies (ii). We therefore have to show that $A'\mathcal{S}'B'$ for all $B \stackrel{\alpha}{\Rightarrow} B'$. If $A' = A''$ then the result follows from the hypothesis. Otherwise let us write $A \xrightarrow{\alpha} A' \xrightarrow{\alpha'}\stackrel{\mathsf{tr}'}{\Rightarrow} A''$ where $\alpha'.\mathsf{tr}' = \mathsf{tr}$ and

the rule (DEC-L) justifies that $A'\mathcal{S}'B'$ for all $B \stackrel{\alpha}{\Rightarrow} B'$.

▷ **case 1.c** : there exists $B \stackrel{\alpha}{\to} B' \stackrel{\mathsf{tr}}{\Rightarrow} B''$ such that $B''\mathcal{S}A''$ for all $A \stackrel{\alpha.\mathsf{tr}}{\Longrightarrow} A''$.
Analogous, targeting (iii) instead of (ii) and replacing (DEC-L) by (DEC-R).

- **case 2** (DEC-L): there are A_0, A_1, A_2, B_0, B_2, α, α', tr, such that $A_0\mathcal{S}'B_0$, $B_0 \stackrel{\alpha}{\Rightarrow} B$, $A_0 \stackrel{\alpha}{\to} A \stackrel{\alpha'}{\to} A_1 \stackrel{\mathsf{tr}}{\Rightarrow} A_2$, and $\forall\ B_0 \stackrel{\alpha.\alpha'.\mathsf{tr}}{\Longrightarrow} B_2, A_2\mathcal{S}'B_2$.
Let us show that the transition $A \stackrel{\alpha'}{\to} A_1$ satisfies (ii). We therefore have to show that $A_1\mathcal{S}'B_1$ for all $B \stackrel{\alpha'}{\Rightarrow} B_1$. If $A_1 = A_2$ then the result follows from the hypothesis. Otherwise we write $A \stackrel{\alpha'}{\to} A_1 \stackrel{\alpha''}{\to}\stackrel{\mathsf{tr}'}{\Rightarrow} A_2$ where $\alpha''.\mathsf{tr}' = \mathsf{tr}$ and the rule (DEC-L) justifies that $A_1\mathcal{S}'B_1$ for all $B \stackrel{\alpha}{\Rightarrow} B_1$.
- **case 3** (DEC-R): Analogous to case 2. □

2.2 Correctness of the encodings (Section 4.1)

Now we prove that the translation $[\![\cdot]\!]$ of the extended semantics is correct:

Lemma D.6

Let $\approx_t^+$ and $\approx_l^+$ be the notions of trace equivalence and labelled bisimilarity over the extended calculus (the flag $^+$ being omitted outside of this lemma). For all extended processes $A = (\mathcal{P}, \Phi)$, the translation $[\![A]\!] = ([\![\mathcal{P}]\!], \Phi) = (\{\!\{[\![P]\!] \mid P \in \mathcal{P}\}\!\}, \Phi)$ can be computed in polynomial time, $A \approx_t^+ [\![A]\!]$ and $A \approx_l^+ [\![A]\!]$.

But first of all, a (trivial) observation about the free variables of a translated process:

Lemma D.7

For all plain processes P and all first-order substitution σ, $[\![P\sigma]\!] = [\![P]\!]\,\sigma$.

We will use this lemma implicitly in the remaining of this section. Besides, as we have the inclusion of relations $\approx_l \subseteq \approx_t$, we only need to prove the observational-equivalence statement of Lemma D.6. We recall that we use notations $A_{\downarrow_{\mathsf{pi}}}$ and $\mathcal{P}_{\downarrow_{\mathsf{pi}}}$ to refer to normal forms w.r.t. $\rightsquigarrow_{\mathsf{pi}}$ (see Corollary D.3).

Proposition D.8

We consider $\mathcal{R}$ the symmetric closure of:

$$\{(C, [\![C]\!]_{\downarrow_{\mathsf{pi}}}) \mid C \text{ extended process such that } C = C_{\downarrow_{\mathsf{pi}}}\}$$

$\mathcal{R}$ is a bisimulation up to simplification.

Proof. $\mathcal{R}$ is symmetric by definition and is trivially included in $\sim$. Let then $(A, B) \in \mathcal{R}$ and $A \stackrel{\alpha}{\to} A'$ and let us exhibit B' such that $B \stackrel{\alpha}{\Rightarrow} B'$ and $A'\mathcal{R}B'$. We perform a case analysis on the rule triggerring the transition $A \stackrel{\alpha}{\to} A'$:

- **case 1** (rules NULL, PAR, THEN, ELSE):
This case cannot arise as A is in normal form w.r.t. $\rightsquigarrow_{\mathsf{pi}}$ by definition of $\mathcal{R}$.

- **case 2** (rule In): $\alpha = \xi(\zeta)$ for some $\xi, \zeta \in \mathcal{T}(\mathcal{F}, \mathcal{F}_0 \cup dom\Phi)$ and:

$$\begin{aligned} A &= (\mathcal{P} \cup \{\!\{u(x).P\}\!\}, \Phi) && \text{with } \mathsf{msg}u,\ \mathsf{msg}\xi\Phi \text{ and } \xi\Phi{\downarrow} = u{\downarrow} \\ A' &= (\mathcal{P} \cup \{\!\{P\{x \mapsto \zeta\Phi{\downarrow}\}\}\!\}, \Phi) && \text{with } \mathsf{msg}\zeta\Phi \end{aligned}$$

  Then, by a case analysis on the hypothesis $A\mathcal{R}B$:
  - ▷ **case 2.a** : $B = [\![A]\!]_{\downarrow_{\mathsf{pi}}}$
    Then we can write:

$$B = ([\![\mathcal{P}]\!]_{\downarrow_{\mathsf{pi}}} \cup \{\!\{u(x).\,[\![P]\!]\}\!\}, \Phi)$$

    and we conclude by remarking that $B \xrightarrow{\xi(\zeta)} B' = ([\![\mathcal{P}]\!]_{\downarrow_{\mathsf{pi}}} \cup \{\!\{[\![P]\!]\,\{x \mapsto \zeta\Phi{\downarrow}\}\}\!\}, \Phi)$ and:

$$A'\tau A'_{\downarrow_{\mathsf{pi}}}\ \mathcal{R}\ [\![A'_{\downarrow_{\mathsf{pi}}}]\!]_{\downarrow_{\mathsf{pi}}} = [\![A']\!]_{\downarrow_{\mathsf{pi}}} = ([\![\mathcal{P}]\!]_{\downarrow_{\mathsf{pi}}} \cup \{\!\{[\![P]\!]\,\{x \mapsto \zeta\Phi{\downarrow}\}\}\!\}_{\downarrow_{\mathsf{pi}}}, \Phi) \overset{\star}{\leftsquigarrow} B'$$

  - ▷ **case 2.b** : $A = [\![B]\!]_{\downarrow_{\mathsf{pi}}}$ (and $B = B_{\downarrow_{\mathsf{pi}}}$)
    Note that $u = \xi\Phi{\downarrow}$ cannot be one of the names introduced by the translation $[\![\cdot]\!]$: these names can indeed not appear in Φ since they are chosen private and fresh and since the semantics cannot introduce new private names. In particular we can write:

$$B = (\mathcal{Q} \cup \{\!\{u(x).Q\}\!\}, \Phi) \qquad \text{with } [\![Q]\!] = P \text{ and } [\![\mathcal{Q}]\!]_{\downarrow_{\mathsf{pi}}} = \mathcal{P}$$

    and we conclude by writing $B \xrightarrow{\xi(\zeta)} B' = (\mathcal{Q} \cup \{\!\{Q\{x \mapsto \zeta\Phi{\downarrow}\}\}\!\}, \Phi)$ and:

$$A'\tau([\![\mathcal{Q}]\!]_{\downarrow_{\mathsf{pi}}} \cup \{\!\{[\![Q]\!]\,\{x \mapsto \zeta\Phi{\downarrow}\}\}\!\}_{\downarrow_{\mathsf{pi}}}, \Phi) = [\![B']\!]_{\downarrow_{\mathsf{pi}}} = [\![B'_{\downarrow_{\mathsf{pi}}}]\!]_{\downarrow_{\mathsf{pi}}}\ \mathcal{R}\ B'_{\downarrow_{\mathsf{pi}}} \overset{\star}{\leftsquigarrow} B'$$

- **case 3** (rule Out): $\alpha = \overline{\xi}\langle \mathsf{ax}_n \rangle$ for some $\xi, \in \mathcal{T}(\mathcal{F}, \mathcal{F}_0 \cup dom\Phi)$, $\mathsf{ax}_n \in \mathcal{AX}$ and:

$$\begin{aligned} A &= (\mathcal{P} \cup \{\!\{\overline{u}\langle t \rangle.P\}\!\}, \Phi) && \text{with } \mathsf{msg}u,\ \mathsf{msg}t,\ \mathsf{msg}\xi\Phi \text{ and } \xi\Phi{\downarrow} = u{\downarrow} \\ A' &= (\mathcal{P} \cup \{\!\{P\}\!\}, \Phi') && \text{where } \Phi' = \Phi \cup \{\mathsf{ax}_n \mapsto t{\downarrow}\} \text{ and } n = |\Phi| + 1 \end{aligned}$$

  Then, by a case analysis on the hypothesis $A\mathcal{R}B$:
  - ▷ **case 3.a** : $B = [\![A]\!]_{\downarrow_{\mathsf{pi}}}$
    Then we can write:

$$B = ([\![\mathcal{P}]\!]_{\downarrow_{\mathsf{pi}}} \cup \{\!\{\overline{u}\langle t \rangle.\,[\![P]\!]\}\!\}, \Phi)$$

    and we conclude by remarking that $B \xrightarrow{\overline{\xi}\langle \mathsf{ax}_n \rangle} B' = ([\![\mathcal{P}]\!]_{\downarrow_{\mathsf{pi}}} \cup \{\!\{[\![P]\!]\}\!\}, \Phi)$ and:

$$A'\tau A'_{\downarrow_{\mathsf{pi}}}\ \mathcal{R}\ [\![A'_{\downarrow_{\mathsf{pi}}}]\!]_{\downarrow_{\mathsf{pi}}} = [\![A']\!]_{\downarrow_{\mathsf{pi}}} = ([\![\mathcal{P}]\!]_{\downarrow_{\mathsf{pi}}} \cup \{\!\{[\![P]\!]\}\!\}_{\downarrow_{\mathsf{pi}}}, \Phi) \overset{\star}{\leftsquigarrow} B'$$

▷ **case 3.b** : $A = [\![B]\!]_{\downarrow_{\mathsf{pi}}}$ (and $B = B_{\downarrow_{\mathsf{pi}}}$)
For the same reason as in case 2.b, we can write:

$$B = (\mathcal{Q} \cup \{\!\{\overline{u}\langle t\rangle.Q\}\!\}, \Phi) \qquad \text{with } [\![Q]\!] = P \text{ and } [\![\mathcal{Q}]\!]_{\downarrow_{\mathsf{pi}}} = \mathcal{P}$$

and we conclude by writing $B \xrightarrow{\overline{\xi}\langle \mathsf{ax}_n\rangle} B' = (\mathcal{Q} \cup \{\!\{Q\}\!\}, \Phi)$ and:

$$A'\tau([\![\mathcal{Q}]\!]_{\downarrow_{\mathsf{pi}}} \cup \{\!\{[\![Q]\!]\}\!\}_{\downarrow_{\mathsf{pi}}}, \Phi) = [\![B']\!]_{\downarrow_{\mathsf{pi}}} = [\![B'_{\downarrow_{\mathsf{pi}}}]\!]_{\downarrow_{\mathsf{pi}}} \mathcal{R}\ B'_{\downarrow_{\mathsf{pi}}} \overset{\star}{\leftrightsquigarrow} B'$$

- **case 4** (rule (Comm)): $\alpha = \tau$ and:

$$A = (\mathcal{P} \cup \{\!\{\overline{u}\langle t\rangle.P, v(x).Q\}\!\}, \Phi) \qquad \text{with } \mathsf{msg}u,\ \mathsf{msg}v,\ \mathsf{msg}t \text{ and } u{\downarrow} = v{\downarrow}$$
$$A' = (\mathcal{P} \cup \{\!\{P, Q\{x \mapsto t\}\}\!\}, \Phi)$$

Then, by a case analysis on the hypothesis $A\mathcal{R}B$:
▷ **case 4.a** : $B = [\![A]\!]_{\downarrow_{\mathsf{pi}}}$
Then we can write:

$$B = ([\![\mathcal{P}]\!]_{\downarrow_{\mathsf{pi}}} \cup \{\!\{\overline{u}\langle t\rangle.\,[\![P]\!]\,, v(x).\,[\![Q]\!]\}\!\}, \Phi)$$

and we conclude by remarking that $B \xrightarrow{\tau} B' = ([\![\mathcal{P}]\!]_{\downarrow_{\mathsf{pi}}} \cup \{\!\{[\![P]\!]\,, [\![Q]\!]\,\{x \mapsto t\}\}\!\}, \Phi)$ and:

$$A'\tau A'_{\downarrow_{\mathsf{pi}}}\ \mathcal{R}\ [\![A'_{\downarrow_{\mathsf{pi}}}]\!]_{\downarrow_{\mathsf{pi}}} = [\![A']\!]_{\downarrow_{\mathsf{pi}}} = ([\![\mathcal{P}]\!]_{\downarrow_{\mathsf{pi}}} \cup \{\!\{[\![P]\!]_{\downarrow_{\mathsf{pi}}}, [\![Q]\!]\,\{x \mapsto t\}_{\downarrow_{\mathsf{pi}}}\}\!\}, \Phi) \overset{\star}{\leftrightsquigarrow} B'$$

▷ **case 4.b** : $A = [\![B]\!]_{\downarrow_{\mathsf{pi}}}$ (and $B = B_{\downarrow_{\mathsf{pi}}}$) and a term w such that $w{\downarrow} = u{\downarrow}$ appears in B (syntactically)
In particular$\downarrow$ u is not a fresh name introduced by the translation $[\![\cdot]\!]$ and we can therefore write:

$$B = (\mathcal{P}' \cup \{\!\{\overline{u}\langle t\rangle.P', v(x).Q'\}\!\}, \Phi) \qquad \text{with } [\![\mathcal{P}']\!]_{\downarrow_{\mathsf{pi}}} = \mathcal{P},\ [\![P']\!] = P \text{ and } [\![Q']\!] = Q$$

and we conclude by writing $B \xrightarrow{\tau} B' = (\mathcal{P}' \cup \{\!\{P', Q'\{x \mapsto t\}\}\!\}, \Phi)$ and:

$$A'\tau([\![\mathcal{P}']\!]_{\downarrow_{\mathsf{pi}}} \cup \{\!\{[\![P']\!]_{\downarrow_{\mathsf{pi}}}, [\![Q']\!]\,\{x \mapsto t\}_{\downarrow_{\mathsf{pi}}}\}\!\}, \Phi) = [\![B']\!]_{\downarrow_{\mathsf{pi}}} = [\![B'_{\downarrow_{\mathsf{pi}}}]\!]_{\downarrow_{\mathsf{pi}}} \mathcal{R}\ B'_{\downarrow_{\mathsf{pi}}} \overset{\star}{\leftrightsquigarrow} B'$$

▷ **case 4.c** : $A = [\![B]\!]_{\downarrow_{\mathsf{pi}}}$ (and $B = B_{\downarrow_{\mathsf{pi}}}$) and there exists no term w appearing in B such that $w{\downarrow} = u{\downarrow}$ (syntactically)
Then u is a fresh name introduced by $[\![\cdot]\!]$. We consider the two disjoint cases where it was introduced for the translation of a sum or a circuit:

– If $u \in \mathcal{N}$ is a fresh name generated in order to translate a sum of B, or rephrased more formally:

$$\begin{aligned} B &= (\mathcal{Q} \cup \{\!\{P' + Q'\}\!\}, \Phi) \\ A &= (\llbracket \mathcal{Q} \rrbracket_{\downarrow_{\mathsf{pi}}} \cup \{\!\{\overline{u}\langle u\rangle, u(x).\, \llbracket P' \rrbracket, u(x).\, \llbracket Q' \rrbracket\}\!\}, \Phi) && \text{where } x \in \mathcal{X}^1 \text{ but } x \notin vars\, P', Q' \\ A' &= (\llbracket \mathcal{Q} \rrbracket_{\downarrow_{\mathsf{pi}}} \cup \{\!\{\llbracket R \rrbracket, u(x).\, \llbracket S \rrbracket\}\!\}, \Phi) && \text{where } R, S \in \{\!\{P', Q'\}\!\},\ R \neq S \end{aligned}$$

We let $B' = (\mathcal{Q} \cup \{\!\{R\}\!\}, \Phi)$ and remark that $B \xrightarrow{\tau} B'$ by the rule Choice (if $R = P'$ and $S = Q'$, or $R = Q'$ and $S = P'$). Besides, let us observe that the name u does not appear in $\llbracket \mathcal{Q} \rrbracket_{\downarrow_{\mathsf{pi}}}$, $\llbracket R \rrbracket$ nor Φ by construction of $\llbracket \cdot \rrbracket$ and that $u(x).\, \llbracket S \rrbracket$ is therefore easily seen to be silent in $A'' = (\llbracket \mathcal{Q} \rrbracket_{\downarrow_{\mathsf{pi}}} \cup \{\!\{\llbracket R \rrbracket\}\!\}, \Phi)$. In particular it entails that $A' \rightsquigarrow A''$ by the rule (S-sil) which gives the conclusion:

$$A' \rightsquigarrow A'' \tau (\llbracket \mathcal{Q} \rrbracket_{\downarrow_{\mathsf{pi}}} \cup \{\!\{\llbracket R \rrbracket\}\!\}_{\downarrow_{\mathsf{pi}}}, \Phi) = \llbracket B' \rrbracket_{\downarrow_{\mathsf{pi}}} = \llbracket B'_{\downarrow_{\mathsf{pi}}} \rrbracket_{\downarrow_{\mathsf{pi}}} \mathcal{R}\ B'_{\downarrow_{\mathsf{pi}}} \overset{\star}{\leftsquigarrow} B'$$

– $u \in \mathcal{N}$ is a fresh name generated in order to translate a $\mathtt{Choose}(x)$: this case can be handle analogously to the previous one.

– If $u \in \mathcal{N}$ is a fresh name generated in order to translate a circuit of B, or formally:

$$\begin{aligned} B &= (\mathcal{Q} \cup \{\!\{\vec{x} \leftarrow \Gamma(\vec{b}).P'\}\!\}, \Phi) \\ A &= (\llbracket \mathcal{Q} \rrbracket_{\downarrow_{\mathsf{pi}}} \cup \{\!\{ \left\llbracket \vec{x} \leftarrow \Gamma(\vec{b}).P' \right\rrbracket \}\!\}, \Phi) \end{aligned}$$

We call $(c_i)_i$ the private fresh names introduced by the translation $\left\llbracket \vec{x} \leftarrow \Gamma(\vec{b}).P' \right\rrbracket$, stressing that none of them appears in $\llbracket \mathcal{Q} \rrbracket_{\downarrow_{\mathsf{pi}}}$, $\llbracket P' \rrbracket$ nor Φ. Here $u \in \{c_i\}_i$ and we can therefore write:

$$A' = (\llbracket \mathcal{Q} \rrbracket_{\downarrow_{\mathsf{pi}}} \cup \{\!\{Q'\}\!\}, \Phi) \qquad \text{where } (\{\!\{ \left\llbracket \vec{x} \leftarrow \Gamma(\vec{b}).P' \right\rrbracket \}\!\}, \Phi) \xrightarrow{\alpha} (\{\!\{Q'\}\!\}, \Phi)$$

If the sequence $\vec{b}$ contains a term which is not a message or does not reduce to a boolean, then one easily obtain that $(\{\!\{Q'\}\!\}, \Phi) \overset{\star}{\rightsquigarrow}_{\mathsf{pi}} (\mathbb{S}, \Phi)$ where $\mathbb{S}$ is silent in $(\llbracket \mathcal{Q} \rrbracket_{\downarrow_{\mathsf{pi}}}, \Phi)$ (for that we assume, w.l.o.g. that each input of Γ goes through at least one gate). Hence since $\{\!\{\vec{x} \leftarrow \Gamma(\vec{b}).P'\}\!\}$ is also silent in $(\mathcal{Q}, \Phi)$, we conclude with $B' = (\mathcal{Q}, \Phi)$.
Otherwise assume that $\mathsf{msg}\vec{b}$ and $\vec{b}{\downarrow} \subseteq \mathbb{B}$. Then one easily obtain by induction on the number of gates of Γ that $(\{\!\{Q'\}\!\}, \Phi) \overset{\star}{\rightsquigarrow}_{\mathsf{pi}} (\{\!\{\llbracket P' \rrbracket\, \{\vec{x} \mapsto \Gamma(\vec{b})\}\}\!\}, \Phi)$ and we conclude by choosing $B' = (\mathcal{Q} \cup \{\!\{P'\{\vec{x} \mapsto \Gamma(\vec{b})\}\}\!\}, \Phi)$.

▷ **case 5** (rules Choice, Choose-0, Choose-1 or Valuate):
The arguments of each of these cases are analogous to priorly-met subcases. □

In particular note that $A \tau A_{\downarrow_{\mathsf{pi}}}\ \mathcal{R}\ \llbracket A_{\downarrow_{\mathsf{pi}}} \rrbracket_{\downarrow_{\mathsf{pi}}} = \llbracket A \rrbracket_{\downarrow_{\mathsf{pi}}} \overset{\star}{\leftsquigarrow} \llbracket A \rrbracket$ for all extended processes A, hence Lemma D.6.

2.3 Reduction

Before concretly proving the pending lemmas, let us introduce some notations and prove intermediary results about static equivalence. We fix a private nonce $s \in \mathcal{N}$ and define the following frames given a protocol term t:

$$\Phi_t = \{\mathsf{ax}_1 \mapsto \mathsf{h}(t,s),\ \mathsf{ax}_2 \mapsto \mathsf{h}(1,s)\}$$
$$\Phi_t^{\mathsf{N}} = \{\mathsf{ax}_1 \mapsto \mathsf{h}_{\mathsf{N}}(t,s),\ \mathsf{ax}_2 \mapsto \mathsf{h}(1,s)\}$$
$$\Phi_t^{\mathbb{B}} = \{\mathsf{ax}_1 \mapsto \mathsf{h}_{\mathbb{B}}(t,s),\ \mathsf{ax}_2 \mapsto \mathsf{h}(1,s)\}$$

One shall observe that the whole deal with our reduction is about which instances of these three frames are reachable in which conditions. Hence first we prove a lemma investigating the static equivalence between some of them:

Lemma D.9

Let t be a message in normal form ($t = t{\downarrow}$). The following properties hold:

(i) $\Phi_t \sim \Phi_0$ *iff* $t \neq 1$

(ii) $\Phi_t^{\mathsf{N}} \sim \Phi_0$ *iff* $root(t) \neq \mathsf{Node}$

(iii) $\Phi_t^{\mathbb{B}} \sim \Phi_0$ *iff* $t \notin \mathbb{B}$

Proof. We prove the three equivalences together by double implication.

$\Rightarrow$ We prove the three properties by contraposition. We naturally proceed by exhibiting ground recipes ξ, ζ witnessing the non-static-equivalence goal:

- (i) We assume $t = 1$ and we choose $\xi = \mathsf{ax}_1$ and $\zeta = \mathsf{ax}_2$: the conclusion follows from $\xi\Phi_t{\downarrow} = \mathsf{h}(1,s) = \zeta\Phi_t{\downarrow}$ and $\xi\Phi_0{\downarrow} = \mathsf{h}(0,s) \neq \mathsf{h}(1,s) = \zeta\Phi_0{\downarrow}$.
- (ii) We assume $t = \mathsf{Node}(t_1,t_2)$ and we choose $\xi = \mathsf{TestNode}(\mathsf{ax}_1)$: the conclusion follows from $\mathsf{msg}\xi\Phi_t^{\mathsf{N}}$ and $\neg\mathsf{msg}\xi\Phi_0^{\mathsf{N}}$.
- (iii) We assume $t \in \{0,1\}$ and we choose $\xi = \mathsf{TestBool}(\mathsf{ax}_1)$: the conclusion follows from $\mathsf{msg}\xi\Phi_t^{\mathbb{B}}$ and $\neg\mathsf{msg}\xi\Phi_0^{\mathbb{B}}$.

$\Leftarrow$ The key point is an observation about rewriting critical pairs (not specific to $\mathcal{R}$):

if: u is a term;

if: σ is a substitution such that for any $m \in img(\sigma)$ there exists no rule $\ell \to r$ of $\mathcal{R}$ such that m is unifiable with a subterm $u \in subterms\ell - \mathcal{X}$;

then: for any rewriting sequence $u\sigma \to^\star s$, it holds that $s = u'\sigma$ for some $u \to^\star u'$ (where u' is in normal form *iff* s is in normal form. In particular $(u\sigma){\downarrow} = (u{\downarrow})\sigma$).

One shall note that any frame Φ investigated by the lemma (Φ_t when $t \neq 1$, Φ_t^{N} when $root t \neq \mathsf{Node}$ and $\Phi_t^{\mathbb{B}}$ when $t \notin \mathbb{B}$) verifies the second hypothesis. As a consequence, if ξ is a ground recipe such that $axioms\xi \subseteq dom(\Phi)$, then $\mathsf{msg}\xi\Phi$ *iff* $\mathsf{msg}\xi\Phi_0$ (*iff* $\forall\zeta \in subterms\xi,\ \zeta{\downarrow} \in \mathcal{T}(\mathcal{F}_{\mathsf{c}}, \mathcal{F}_0 \cup \mathcal{AX})$). This settles the first item of the definition of static equivalence. As for the second item, let us fix two ground recipes ξ, ζ such that $\mathsf{msg}\xi\Phi$ and $\mathsf{msg}\xi\Phi_0$. Let us then prove that $\xi\Phi{\downarrow} = \zeta\Phi{\downarrow}$ *iff* $\xi\Phi_0{\downarrow} = \zeta\Phi_0{\downarrow}$ by induction on $(\xi{\downarrow}, \zeta{\downarrow})$. Note that we will intensively (and implicitly) use the fact that $\xi\Phi{\downarrow} = (\xi{\downarrow})\Phi$ (same for ζ and/or Φ_0).

- **case 1** : $\xi{\downarrow} = f(\xi_1, \ldots, \xi_n)$ and $\zeta{\downarrow} = f(\zeta_1, \ldots, \zeta_p)$ with $f, g \in \mathcal{F}_{\mathsf{c}}$.
  If $f = g$ then the result follows from induction hypothesis and if $f \neq g$ the conclusion is

immediate ($\xi\Phi\downarrow \neq \zeta\Phi\downarrow$ and $\xi\Phi_0\downarrow \neq \zeta\Phi_0\downarrow$).

- **case 2** : $\xi\downarrow \in \mathcal{AX}$ and $\zeta\downarrow \in \mathcal{AX}$.
  If $\xi = \zeta$ the conclusion is immediate and so is it when $\xi \neq \zeta$ since $\Phi(\mathsf{ax}_1) \neq \Phi(\mathsf{ax}_2)$ and $\Phi_0(\mathsf{ax}_1) \neq \Phi_0(\mathsf{ax}_2)$.
- **case 3** : $\xi\downarrow \in \mathcal{AX}$ and $\zeta\downarrow = f(\zeta_1, \ldots, \zeta_p)$ with $f \in \mathcal{F}_\mathsf{c}$.
  We argue that $\xi\Phi\downarrow \neq \zeta\Phi\downarrow$ and $\xi\Phi_0\downarrow \neq \zeta\Phi_0\downarrow$. Either of the two equalities being verified would indeed imply that $s \in \{\zeta_2\Phi, \zeta_2\Phi_0\}$: this is impossible as ζ_2 is a ground recipe in normal form and $s \in \mathcal{N}$ (one easily shows that $\zeta_2\Phi$ and $\zeta_2\Phi_0$ are either public names, constants, or termes of height 1 or more).
  As $\mathsf{msg}\xi\Phi$ and $\mathsf{msg}\xi\Phi_0$, the preliminary observation justifies that $\xi\downarrow$ and $\zeta\downarrow$ function symbols are all constructor. In particular no other cases than the three above need to be considered, which concludes the proof. □

With this lemma in mind, the proofs of Propositions 6.16 and 6.17 become quite straightforward:

Proposition 6.16 (correctness of the tree checker)

Let x be a message which is not a complete binary tree of height n with boolean leaves. Then there exists a reduction $\mathtt{CheckTree}(x) \overset{\varepsilon}{\Rightarrow} (\{\!\{P\}\!\}, \varnothing)$ such that $P \approx_l \overline{c}\langle \mathsf{h}(0,s)\rangle.\, \overline{c}\langle \mathsf{h}(1,s)\rangle$.

Proof. Let x be a message which is not a complete binary tree of height n whose leaves are booleans.

▷

•

case 1 : there exists a position $\vec{\pi} \in \mathbb{B}^\star$ such that $|\vec{\pi}| = i \in [\![0, n-1]\!]$ and $root x_{|\vec{\pi}} \neq \mathsf{Node}$.
The result follows from Lemma D.9 after writing the following sequence of transitions:

$$\begin{aligned}\mathtt{CheckTree}(x) &\overset{\varepsilon}{\Rightarrow} \mathtt{Choose}(p_1, \ldots, p_i).\ \overline{c}\langle \mathsf{h_N}(x_{|p_1 \cdots p_i}, s)\rangle.\ \overline{c}\langle \mathsf{h}(1,s)\rangle \\ &\overset{\varepsilon}{\Rightarrow} \overline{c}\langle \mathsf{h_N}(x_{|\vec{\pi}}, s)\rangle.\ \overline{c}\langle \mathsf{h}(1,s)\rangle\end{aligned}$$

case 2 : there exists a position $\vec{\pi} \in \mathbb{B}^n$ such that $x_{|\vec{\pi}} \notin \mathbb{B}$.
The result follows from Lemma D.9 after writing the following sequence of transitions:

$$\begin{aligned}\mathtt{CheckTree}(x) &\overset{\varepsilon}{\Rightarrow} \mathtt{Choose}(p_1, \ldots, p_n).\ \overline{c}\langle \mathsf{h_N}(x_{|p_1 \cdots p_n}, s)\rangle.\ \overline{c}\langle \mathsf{h}(1,s)\rangle \\ &\overset{\varepsilon}{\Rightarrow} \overline{c}\langle \mathsf{h_B}(x_{|\vec{\pi}}, s)\rangle.\ \overline{c}\langle \mathsf{h}(1,s)\rangle\end{aligned}$$

□

Proposition 6.17 (correctness of the sat checker)

Let x be a complete binary tree of height n whose leaves are booleans, and val_x be the valuation mapping the variable number i of $[\![\Gamma]\!]_\varphi$ to $x_{|p_1 \cdots p_n} \in \mathbb{B}$ where $p_1 \cdots p_n$ is the binary representation of i (i.e., $i = \sum_{k=1}^{n} p_k 2^{k-1}$). If val_x does not satisfy $[\![\Gamma]\!]_\varphi$ then there exists $\mathtt{CheckSat}(x) \overset{\varepsilon}{\Rightarrow} P$ such that $P \approx_l \overline{c}\langle \mathsf{h}(0,s)\rangle.\, \overline{c}\langle \mathsf{h}(1,s)\rangle$.

Proof. Let x be a complete binary tree of height n whose leaves are booleans, that is to say, a message such that $x_{|\vec{p}} \in \mathbb{B}$ for all $\vec{p} \in \mathbb{B}^n$. Naming $x_0, \ldots, x_{2^n-1}$ the variables of $[\![\Gamma]\!]_\varphi$ in this order, val_x refers to the valuation mapping x_i to $x_{|\vec{p}}$ where $\vec{p}$ is the binary representation of i (of size n with padding head 0's).

Let us now assume that val_x does not satisfy $[\![\Gamma]\!]_\varphi$. In particular there exists a clause of $[\![\Gamma]\!]_\varphi$, say the i^{th} clause with $i = \sum_{k=1}^{m} \pi_k 2^{k-1}$, which is falsified by val_x. In particular, if the three variable of this clause are called x_1, x_2, x_3 with respective negation bits b_1, b_2, b_3, the following formula is evaluated to false (i.e. $\mathsf{0}$):

$$\bigvee_{i=1}^{3} (\ b_j = \mathsf{val}_x(x_j)\ )$$

Therefore, by choosing the sequence $\pi_1, \ldots, \pi_m$ to instanciate the initial $\mathtt{Choose}(p_1, \ldots, p_m)$ of $\mathtt{CheckSat}(x)$, we obtain the following sequence of transitions, which concludes the proof:

$$\mathtt{CheckSat}(x) \stackrel{\varepsilon}{\Rightarrow} \overline{c}\langle \mathsf{h}(\mathsf{0}, s)\rangle.\ \overline{c}\langle \mathsf{h}(\mathsf{1}, s)\rangle \qquad \square$$

We finally gathered all the ingredients needed to prove the main lemma:

Proposition 6.18 (correctness of the reduction)

$[\![\Gamma]\!]_\varphi$ is satisfiable *iff* $A \not\approx_t B$ *iff* $A \not\approx_l B$.

Proof. We name the three properties as follows: (i) $[\![\Gamma]\!]_\varphi$ is satisfiable, (ii) $A \not\approx_t B$ and (iii) $A \not\approx_l B$. We reach the conclusion by the following chain of implications.

$(i) \Rightarrow (ii)$: Let us consider a valuation satisfying φ and let t be a message such that val_t is equal to this valuation. Then since the trace $B \xrightarrow{c(t).\overline{c}\langle \mathsf{ax}_1\rangle.\overline{c}\langle \mathsf{ax}_2\rangle} (\varnothing, \Phi_0)$ cannot be matched in A by Lemma D.9, we obtain $B \not\sqsubseteq_t A$. Hence the (ii).

$(ii) \Rightarrow (iii)$: Follows from the inclusion of relations $\approx_l \subseteq \approx_t$.

$\neg(i) \Rightarrow \neg(iii)$: Let us consider $\mathcal{R}$ the smallest reflexive symmetric relation on extended processes such that:

1. $A \mathcal{R} B$
2. $A'(x) \mathcal{R} B'(x)$ for all message x, where $A = c(x).A'(x)$ and $B = c(x).B'(x)$
3. $P_i \mathcal{R} P'_i$, where:

$$P_i = (\{\!\{\overline{c}\langle t_{i+1}\rangle \ldots \overline{c}\langle t_p\rangle\}\!\}, \{\mathsf{ax}_1 \mapsto t_1, \ldots, \mathsf{ax}_i \mapsto t_i\})$$
$$P'_i = (\{\!\{\overline{c}\langle t'_{i+1}\rangle \ldots \overline{c}\langle t'_p\rangle\}\!\}, \{\mathsf{ax}_1 \mapsto t'_1, \ldots, \mathsf{ax}_i \mapsto t'_i\})$$

and where $\{\mathsf{ax}_1 \mapsto t_1, \ldots, \mathsf{ax}_p \mapsto t_p\} \sim \{\mathsf{ax}_1 \mapsto t'_1, \ldots, \mathsf{ax}_p \mapsto t'_p\}$

It easily follows from Lemma D.9,6.16,6.17 that $\mathcal{R}$ is a bisimulation up to $\rightsquigarrow$, hence the conclusion. $\square$

Appendix E:
Proofs of Chapter 7

Summary.

In this appendix we prove the technical results used in Chapter 7 surveying the complexity of equivalence properties in the applied pi calculus.

1 co-NEXP hardness of equivalences in the bounded fragment

In this section we prove various statements of coNEXP-hardness (see Theorem 6.19). They can be seen as extensions of the results of [CKR18a] (that studied the complexity of TraceEquiv and Bisimilarity). First we show that the reduction of [CKR18a] can be performed without private channels and with a minimal theory, at least for trace equivalence and equivalence by session.

> ***Theorem E.1***
>
> For a theory limited to symmetric encryption and pairs, TraceEquiv and SessEquiv are coNEXP-hard for bounded processes whose channels are constants.

We also show how this lower bound can be extended to the positive fragment by using a slightly larger theory. Regarding Bisimilarity our proof required private channels but no else branches:

> ***Theorem E.2***
>
> For a theory limited to symmetric encryption and pairs, Bisimilarity is coNEXP-hard for bounded positive processes.

We prove all these results by reduction from SuccinctSAT, a NEXP problem that we already described in the body of the paper (see the proof sketch of Theorem 7.8). We thus let φ a formula with 2^m clauses and 2^n variables $x_0, \ldots, x_{2^n-1}$ and Γ a boolean circuit encoding this formula. In each proof we construct two bounded processes P and Q that match the hypotheses of the theorem statement and such that P and Q are equivalent *iff* φ is unsatisfiable.

1.1 Proof of Theorem E.1

Construction Intuitively the processes P and Q first wait for an input x from the attacker which is expected to be a valuation of the 2^n variables of φ (in practice a binary tree of height n modelled using nested pairs). Then the processes non-deterministically chooses a branch of x (i.e. a sequence of $\{0,1\}^n$), a clause of φ (i.e. a sequence of $\{0,1\}^m$) and executes one of the following three actions (P can execute either of the three, and Q only the first two):

1. extract the selected branch from x, then simulate a dummy evaluation of the selected clause, and eventually output three messages that are statically-equivalent to three fresh nonces *iff* the branch extraction failed.
2. simulate a dummy extraction of the selected branch, then evaluate the selected clause w.r.t. the valuation encoded by x, and eventually output three messages that are statically-equivalent to three fresh nonces *iff* the clause has been falsified.
3. simulate a dummy extraction of the selected branch and a dummy evaluation of the selected clause, and eventually output three fresh nonces.

In the end P and Q are equivalent *iff* for all terms $x \in \mathcal{T}(\mathcal{F}, \mathcal{F}_0)$ either x is not a binary tree of height n, or it is one but the valuation it encodes falsifies a clause of φ. All in all P and Q are equivalent *iff* φ is unsatisfiable.

Formalisation We consider the following atomic data

constants :

c	channel of the process
$b_0, \ldots, b_{n+2}$	modelling $[\![0, n+2]\!]$

names :

$k, k_1, \ldots, k_{n+m}$	encr. keys for non-deterministic choices
$r_1, \ldots, r_{4n}$	encr. keys for extraction requests
$s_1, \ldots, s_{4n}$	encr. keys for dummy extraction requests
$a_1, \ldots, a_{4n}$	encr. keys for extraction results
α, β, γ	fresh nonces

The processes P and Q to be proved equivalent are then of the following form

$$P = c(x).(\mathit{Choice} \mid \mathit{Extract} \mid \mathit{Eval} \mid c(y).P'(x,y))$$
$$Q = c(x).(\mathit{Choice} \mid \mathit{Extract} \mid \mathit{Eval} \mid c(y).Q'(x,y))$$

where

$$P'(x,y) = \left(\begin{array}{l} [\mathsf{sdec}(y,k) = b_{n+2}]\, \mathit{Print} \mid \\ [\mathsf{sdec}(y,k) = b_{n+1}]\, \mathit{CheckSat}(x) \mid \\ \prod_{i=0}^{n} [\mathsf{sdec}(y,k) = b_i]\, \mathit{CheckTree}_i(x) \end{array} \right)$$

$$Q'(x,y) = \left(\begin{array}{l} [\mathsf{sdec}(y,k) = b_{n+2}]\, \mathit{CheckSat}(x) \mid \\ [\mathsf{sdec}(y,k) = b_{n+1}]\, \mathit{CheckSat}(x) \mid \\ \prod_{i=0}^{n} [\mathsf{sdec}(y,k) = b_i]\, \mathit{CheckTree}_i(x) \end{array} \right)$$

The processes $P'(x,y)$ and $Q'(x,y)$ execute one of their $n+3$ branches (or none) depending on the input y, expectedly forwarded from *Choice*. Referring to the intuition of the construction provided in the previous paragraph, the process *Print* corresponds to Item 3, *CheckSat* to Item 2, and the collection of processes $\mathit{CheckTree}_i$ to Item 1 (each $\mathit{CheckTree}_i$ is dedicated to the verification of the depth i of the branch). In particular the various branch extractions of the tree x (resp. the evaluations of Γ and of the clauses of φ) that need to be performed by the different processes are simulated by interactions with the process *Extract* (resp. with the process *Eval*).

Formally the processes are defined as follows. For conceitness we use pattern notations in the definitions of the other processes below, recall Section 2.1. All the patterns used below can easily be encoded within the positive fragment of the calculus. We recall in particular that, for any fixed term k, it is possible to test that a term is of the form $\mathsf{senc}(y,k)$ (in the constructor-destructor semantics, the conditional $[\mathsf{sdec}(x,k) = \mathsf{sdec}(x,k)]$ succeeds *iff* x is of the form $\mathsf{senc}(y,k)$ for some term y). We also use a wildcard notation (_) for input variables that will not be used afterwards.

Definition of *Choice* We first define a process *Choice* that will serve as an oracle for performing the $n+m+1$ non-deterministic choices the executions of P and Q require.

$$\mathit{Choice} = \prod_{i=0}^{n+2} \mathsf{senc}(b_i,k) \mid \prod_{i=1}^{n+m} (\mathsf{senc}(b_0,k_i) \mid \mathsf{senc}(b_1,k_i))$$

Other processes select a boolean b non-deterministically by inputting $\mathsf{senc}(b,k_i)$ for some i (and each k_i should be used for only one such non-deterministic choice).

Definition of *Extract* Then we define the process performing branch extraction from a binary tree modelled by nested pairs.

$$\begin{aligned}
&\mathit{Extract} = \textstyle\prod_{i=1}^{4n} E_i \mid E_i^{\mathit{dummy}} \\
&\text{with } E_i = c(\mathsf{senc}(\langle\langle x_0, x_1\rangle, z\rangle, r_i)).\, \textstyle\prod_{j=0}^{1} [z = b_j]\, \overline{c}\langle \mathsf{senc}(x_j, a_i)\rangle \\
&E_i^{\mathit{dummy}} = c(\mathsf{senc}(x, s_i)).\, \overline{c}\langle \mathsf{senc}(x, a_i)\rangle
\end{aligned}$$

That is, *Extract* is able to answer to $4n$ pair-extraction requests, potentially including some dummy requests where the argument is returned unchanged. In the other processes, if $t, t_1, \ldots, t_n$ are terms and $1 \leqslant i \leqslant 3n+1$, the branch extractions are written

$$u_n \leftarrow \mathit{Extr}_i(t, t_1, \ldots, t_n).\, P$$

instead of:

$$\begin{aligned}
&\overline{c}\langle \mathsf{senc}(\langle t, t_1\rangle, r_i)\rangle.\, c(\mathsf{senc}(u_1, a_i)). \\
&\overline{c}\langle \mathsf{senc}(\langle u_1, t_2\rangle, r_{i+1})\rangle.\, c(\mathsf{senc}(u_2, a_{i+1})). \\
&\qquad \vdots \\
&\overline{c}\langle \mathsf{senc}(\langle u_{n-1}, t_n\rangle, r_{i+n-1})\rangle.\, c(\mathsf{senc}(u_n, a_{i+n-1})).\, P
\end{aligned}$$

On the other hand, the dummy extractions are written

$$\mathit{DummyExtr}_i^n.\, P$$

instead of

$$\begin{aligned}&\overline{c}\langle\mathsf{senc}(b_0, s_i)\rangle.\, c(\mathsf{senc}(\_, a_i)).\\ &\qquad\vdots\\ &\overline{c}\langle\mathsf{senc}(b_0, s_{i+n-1})\rangle.\, c(\mathsf{senc}(\_, a_{i+n-1})).\, P\end{aligned}$$

Definition of *Eval* Moving on to the process *Eval*, a gate of a circuit is seen as a tuple (e_1, e_2, f, e_3, e_4) where e_1, e_2 are the input edges, e_3, e_4 are the output edges, and $f : \mathbb{B}^2 \to \mathbb{B}$ is the boolean function computed by the gate where $\mathbb{B} = \{b_0, b_1\}$ models the set of booleans. Given a circuit C we let $G(C)$ the set of its gates and, for each edge e of C, we associate a fresh name k_e. We thus define

$$\begin{aligned}[\![C]\!] &= \textstyle\prod_{g\in G(C)} [\![g]\!]\\ [\![(e_1, e_2, f, e_3, e_4)]\!] &= c(\mathsf{senc}(x, k_{e_1})).\, c(\mathsf{senc}(y, k_{e_2})).\\ &\qquad \textstyle\prod_{b,b'\in\mathbb{B}} [x = b][y = b']\, \overline{c}\langle o_3\rangle.\, \overline{c}\langle o_4\rangle\end{aligned}$$

with $o_i = \mathsf{senc}(f(b, b'), k_{e_i})$. We then let

$$\mathit{Eval} = [\![\Gamma_1]\!] \mid [\![\Gamma_2]\!] \mid [\![\Gamma_3]\!] \mid [\![\Gamma_v]\!]$$

where $\Gamma_1, \Gamma_2, \Gamma_3$ are three fresh copies of Γ (with fresh edges) and Γ_v a circuit computing the boolean function

$$\left\{\begin{array}{ccc}\mathbb{B}^6 & \to & \mathbb{B}\\ (x_1, x_1', x_2, x_2', x_3, x_3') & \mapsto & (x_1 = x_1' \vee x_2 = x_2' \vee x_3 = x_3')\end{array}\right.$$

For the sake of succinctness, when an other processes interacts with *Eval* to, say, compute the a circuit C whose input edges (resp. output edges) are $e_1, \ldots, e_p$ (resp. $f_1, \ldots, f_q$), we write

$$x_1, \ldots, x_q \leftarrow C(t_1, \ldots, t_p).\, P$$

instead of:

$$\begin{aligned}&\overline{c}\langle\mathsf{senc}(t_1, e_1)\rangle \ldots \overline{c}\langle\mathsf{senc}(t_p, e_p)\rangle.\\ &c(\mathsf{senc}(x_1, f_1)) \ldots c(\mathsf{senc}(x_q, f_q)).\, P\end{aligned}$$

Process *CheckTree*$_i$ We now move on to the process verifying that the initial input x provided by the attacker is indeed a binary tree of height n.

$$\begin{aligned}\mathit{CheckTree}_i(x) = \;& c(\mathsf{senc}(v_1, k_1)) \ldots c(\mathsf{senc}(v_i, k_i)).\\ & c(\mathsf{senc}(\_, k_{i+1})) \ldots c(\mathsf{senc}(\_, k_{n+m})).\\ & x_i \leftarrow \mathit{Extr}_1(x, v_1, \ldots, v_i).\\ & \mathit{DummyExtr}_{i+1}^{n-i}.\\ & \_ \leftarrow \Gamma_1(b_0, \ldots, b_0).\\ & \_ \leftarrow \Gamma_2(b_0, \ldots, b_0).\\ & \_ \leftarrow \Gamma_3(b_0, \ldots, b_0).\\ & \mathit{DummyExtr}_{n+1}^{n}.\\ & \mathit{DummyExtr}_{2n+1}^{n}.\\ & \mathit{DummyExtr}_{3n+1}^{n}.\\ & \_ \leftarrow \Gamma_v(b_0, \ldots, b_0).\\ & R_i(x_i)\end{aligned}$$

where the process $R_i(t)$ is defined by

$$\begin{aligned} \text{if } i < n,\ R_i(t) = {} & \mathsf{if}\ \mathsf{fst}(t) = \mathsf{fst}(t)\ \mathsf{then}\ \overline{c}\langle\alpha\rangle.\,\overline{c}\langle\beta\rangle.\,\overline{c}\langle\beta\rangle \\ & \mathsf{else}\ \overline{c}\langle\alpha\rangle.\,\overline{c}\langle\beta\rangle.\,\overline{c}\langle\gamma\rangle \\ R_n(t) = {} & \overline{c}\langle\mathsf{senc}(t,\alpha)\rangle.\,\overline{c}\langle\mathsf{senc}(b_0,\alpha)\rangle.\,\overline{c}\langle\mathsf{senc}(b_1,\alpha)\rangle \end{aligned}$$

The process *CheckTree*$_i$ selects non-deterministically a position $p \in \{0,1\}^i$ in the tree (modelled by the variables $v_1, \ldots, v_i$) and extracts the corresponding node x_i of x by interacting with *Extract*. Then the process R_i, $i < n$ (resp $i = n$), outputs three messages that are indistinguishable from three fresh nonces *iff* x_i is ill-formed, i.e. if x_i is not a pair (resp. not in $\mathbb{B}$). The rest of the process only consists of dummy operations so that *CheckTree*$_i$ performs the same number of actions than *Print* and *Eval*.

Definition of *CheckSat* Next, assuming that the initial input x passes the test of all *CheckTree*$_i$, we define the process verifying that x encodes a valuation that satisfies the formula φ.

$$\begin{aligned} \mathit{CheckSat}(x) = {} & c(\mathsf{senc}(\_, k_1)) \ldots c(\mathsf{senc}(\_, k_n)). \\ & c(\mathsf{senc}(v_1, k_{n+1})) \ldots c(\mathsf{senc}(v_m, k_{n+m})). \\ & \mathit{DummyExtr}_1^n. \\ & \omega_1, x_1, \ldots, x_n \leftarrow \Gamma_1(v_1, \ldots, v_m, b_0, b_0). \\ & \omega_2, y_1, \ldots, y_n \leftarrow \Gamma_2(v_1, \ldots, v_m, b_0, b_1). \\ & \omega_3, z_1, \ldots, z_n \leftarrow \Gamma_3(v_1, \ldots, v_m, b_1, b_0). \\ & \omega'_1 \leftarrow \mathit{Extr}_{n+1}(x, x_1, \ldots, x_n). \\ & \omega'_2 \leftarrow \mathit{Extr}_{2n+1}(x, y_1, \ldots, y_n). \\ & \omega'_3 \leftarrow \mathit{Extr}_{3n+1}(x, z_1, \ldots, z_n). \\ & \omega \leftarrow \Gamma_v(\omega_1, \omega'_1, \omega_2, \omega'_2, \omega_3, \omega'_3). \\ & \overline{c}\langle\mathsf{senc}(\omega, \alpha)\rangle.\,\overline{c}\langle\mathsf{senc}(b_1, \alpha)\rangle.\,\overline{c}\langle\beta\rangle \end{aligned}$$

The process *CheckSat* selects non-deterministically one of the 2^m clauses of φ (which modelled by the variables $v_1, \ldots, v_m$) and then computes $\omega \in \mathbb{B}$ the valuation of this clause w.r.t. the valuation encoded by x. For that it evaluates the three copies of Γ to retrieve the three variables and negation bits of the clause, and performs branch extractions to retrive the valuations $\omega'_1, \omega'_2, \omega'_3$ of the three variables. Then the final three outputs are statically equivalent to three fresh nonces *iff* the clause has been falsified, i.e. $\omega = b_0$.

Definition of *Print* We finally define a process that serves as a baseline for comparison in the equivalence proof: it only performs dummy operations and eventually output three fresh nonces.

$$
\begin{aligned}
\mathit{Print} = \;& c(\mathsf{senc}(\_, k_1)) \dots c(\mathsf{senc}(\_, k_{n+m})).\\
& \mathit{DummyExtr}_1^n.\\
& \_ \leftarrow \Gamma_1(b_0, \dots, b_0).\\
& \_ \leftarrow \Gamma_2(b_0, \dots, b_0).\\
& \_ \leftarrow \Gamma_3(b_0, \dots, b_0).\\
& \mathit{DummyExtr}_{n+1}^n.\\
& \mathit{DummyExtr}_{2n+1}^n.\\
& \mathit{DummyExtr}_{3n+1}^n.\\
& \_ \leftarrow \Gamma_v(b_0, \dots, b_0).\\
& \overline{c}\langle\alpha\rangle.\, \overline{c}\langle\beta\rangle.\, \overline{c}\langle\gamma\rangle
\end{aligned}
$$

Correctness of the construction The correctness of this reduction is summed up by the fact that the following three points are equivalent:

(i) φ is unsatisfiable
(ii) P and Q are equivalent by session
(iii) P and Q are trace equivalent

▷ Proof of (i) $\Longrightarrow$ (ii).

Partial-order reductions have been developed in [CKR19] to make the verification of equivalence by session of bounded processes easier. Formally we let $\mathbb{O}^\forall$ the set of all traces t that have the following properties:

1. t never applies the rules (In) and (Comm) when either the rules (Par) and (Out) are applicable.
2. given an arbitrary fixed total ordering on instances of rules (Par) and (Out), t never applies an instance of these rules when a lower instance (w.r.t. this total ordering) is applicable.
3. given an application of (In) in t of the form

$$(\{\!\{c(x).P\}\!\} \cup \mathcal{P}, \Phi) \xrightarrow{c(\xi)} (\{\!\{d(y).Q\}\!\} \cup \mathcal{P}, \Phi)$$

for $d \in \mathcal{F}_0$, then the next transition in t (if any) after $\rightsquigarrow$-normalisation is an instance of rule (In) of the form

$$(\{\!\{d(y).Q\}\!\} \cup \mathcal{P}, \Phi) \xrightarrow{d(\zeta)} (\{\!\{Q\{y \mapsto \zeta\Phi{\downarrow}\}\}\!\} \cup \mathcal{P}, \Phi)\,.$$

We write $P \sqsubseteq_{\mathbb{O}^\forall} Q$ when for all traces t of P that belong to $\mathbb{O}^\forall$, there exists a trace t' of Q such that $t \sim t'$ and t and t' are the two projections of a same twin trace of (P, Q).

Theorem E.3 *([CKR19])*
P and Q are equivalent by session iff $P \sqsubseteq_{\mathbb{O}^\forall} Q$ and $Q \sqsubseteq_{\mathbb{O}^\forall} P$.

In particular we will write $\mathbb{O}_0^\forall$ the set of traces of P or Q that first execute the initial input on c, then all the possible applications of rule (Par), the output of all messages of *Choice* in this order:

$$\mathsf{senc}(b_0, k), \dots, \mathsf{senc}(b_{n+2}, k),$$
$$\mathsf{senc}(b_0, k_1), \mathsf{senc}(b_1, k_1), \dots, \mathsf{senc}(b_0, k_{n+m}), \mathsf{senc}(b_1, k_{n+m})$$

eventually followed by an arbitrary suffix trace of $\mathbb{O}^\forall$. To conclude it then suffices to prove that $Q \sqsubseteq_{\mathbb{O}_0^\forall} P$ and $P \sqsubseteq_{\mathbb{O}_0^\forall} Q$.

The first inclusion holds independently of φ being unsatisfiable. Indeed let t is a maximal trace of Q belonging to $\mathbb{O}_0^\forall$ and let ξ the recipe fetched to the input $c(y)$ preceding Q' and Φ the frame at the time of this input. We also let the term $m = \mathsf{sdec}(\xi\Phi, k)$. If m is not a message, or if m is a message and $m{\downarrow} = b_i$ for $i \leqslant n+1$ then the trace t can trivially be matched in Q. Otherwise m is a message and $m{\downarrow} = b_{n+2}$. Then the trace can be matched in Q by executing the outputs of *Choice* in the following order:

$$\begin{aligned}&\mathsf{senc}(b_0, k), \dots, \mathsf{senc}(b_n, k), \mathsf{senc}(b_{n+2}, k), \mathsf{senc}(b_{n+1}, k),\\ &\mathsf{senc}(b_0, k_1), \mathsf{senc}(b_1, k_1), \dots, \mathsf{senc}(b_0, k_{n+m}), \mathsf{senc}(b_1, k_{n+m})\end{aligned}$$

Let us then prove that, if φ is unsatisfiable then $P \sqsubseteq_{\mathbb{O}_0^\forall} Q$. Following the same reasoning as in the other inclusion, writing ξ the recipe fetched to the input $c(y)$ preceding P', Φ the frame at the time of this input and $m = \mathsf{sdec}(\xi\Phi, k)$, the only non-trivial case is when it holds that m is a message, $m{\downarrow} = b_{n+2}$, and P executes the three final outputs of *Print*. We thus let $\xi_1, \dots, \xi_{n+m}$ the initial $n+m$ input recipes in this execution of *Print* and Φ the corresponding frame (it stays the same across all inputs since we consider a trace of $\mathbb{O}_0^\forall$), as well as $m_1, \dots, m_{n+m}$ with $m_i = \mathsf{sdec}(\xi_i\Phi, k_i)$. Note that in this case, all terms m_i's verify the predicate msg and $m_i{\downarrow} \in \mathbb{B}$.

We then make a case analysis on $\xi_0 \in \mathcal{T}(\mathcal{F}, \mathcal{F}_0)$ the term fetched to the first input of the trace.

- *case 1: ξ_0 is not a complete binary tree of height n or more.*
  Given a sequence of booleans $\vec{p}$, we say that $\vec{p}$ is a position of ξ_0 when $\xi_0 = \langle u_0, u_1\rangle$ for some terms u_0, u_1 and either
  – $\vec{p}$ is empty, or
  – $\vec{p} = b_i \cdot \vec{p}'$ and $\vec{p}'$ is a position of u_i.
  In particular there exists a sequence $\vec{p} = p_1, \dots, p_a$, $0 \leqslant a < n$, that is not a position of ξ_0. Without loss of generality we assume this position minimal, i.e. $a = 0$ or $p_1, \dots, p_{a-1}$ is a position of ξ_0. Then the trace t can be matched in Q by executing $\mathit{CheckTree}_a$ and guessing $\vec{p}$, i.e. we execute the outputs of *Choice* in the following order:

  $$\begin{aligned}&\mathsf{senc}(b_0, k), \dots, \mathsf{senc}(b_{a-1}, k), \mathsf{senc}(b_{n+2}, k),\\ &\mathsf{senc}(b_{a+1}, k), \dots, \mathsf{senc}(b_{n+1}, k), \mathsf{senc}(b_a, k),\\ &\mathsf{senc}(b_{\pi_1(0)}, k_1), \mathsf{senc}(b_{\pi_1(1)}, k_1), \dots, \mathsf{senc}(b_{\pi_{n+m}(0)}, k_{n+m}),\\ &\mathsf{senc}(b_{\pi_{n+m}(1)}, k_{n+m})\end{aligned}$$

  where π_i are permutations of $\{0,1\}$ such that $\pi_i \neq id$ *iff* $1 \leqslant i \leqslant a$ and $m_i{\downarrow} \neq p_i$.
- *case 2: ξ_0 is a complete binary tree of height n or more, and one of its nodes at depth n is not a leaf, or is a leaf that does not belong to $\mathbb{B}$.*
  The reasoning is analogous to the previous case, with $a = n$.
- *case 3: ξ_0 is a complete binary tree of height n with boolean leaves.*
  We let v the valuation of φ encoded by ξ_0, i.e. $v(x_i)$, $i \in [\![0, 2^n - 1]\!]$, is the i^{th} leaf of ξ_0 (ordered w.r.t. a leftmost depth-first search in the binary tree). Since φ is unsatisfiable by hypothesis, there exists a clause C_p that is falsifies by v. We write $\vec{p} = p_1, \dots, p_m$ the

binary decomposition of the integer $p \in [\![0, 2^m - 1]\!]$. Then the trace t can be matched in Q by executing *CheckSat* and guessing $\vec{p}$, i.e. we execute the outputs of *Choice* in the following order:

$$\begin{aligned}&\mathsf{senc}(b_0, k), \ldots, \mathsf{senc}(b_{n+2}, k),\\ &\mathsf{senc}(b_{\pi_1(0)}, k_1), \mathsf{senc}(b_{\pi_1(1)}, k_1), \ldots, \mathsf{senc}(b_{\pi_{n+m}(0)}, k_{n+m}),\\ &\mathsf{senc}(b_{\pi_{n+m}(1)}, k_{n+m})\end{aligned}$$

where π_i are permutations of $\{0, 1\}$ such that $\pi_i \neq id$ *iff* $n + 1 \leqslant i \leqslant n + m$ and $m_i{\downarrow} \neq p_i$.

▷ Proof of (*ii*) $\Longrightarrow$ (*iii*).

This follows from the soundness of equivalence by session w.r.t. trace equivalence, i.e., Proposition 5.2.

▷ Proof of (*iii*) $\Longrightarrow$ (*ii*).

Let $v : \{x_0, \ldots, x_{2^n-1}\} \to \mathbb{B}$ a valuation of φ and prove that v falsifies a clause of φ. We let ξ_0 the complete binary tree of height n whose i^{th} leaf (ordered w.r.t. a leftmost depth-first search in the binary tree) is $v(x_i)$. Consider then the trace of P that inputs ξ_0, executes all outputs of *Choice* and proceeds onto executing the last three outputs of *Print*. This trace cannot be matched by any trace of Q executing any *CheckTree*$_i$, $0 \leqslant i \leqslant n$, because ξ_0 is a complete binary tree with boolean leaves. Hence there exists a matching trace in Q executing *CheckSat*: let us write $\xi_1, \ldots, \xi_{n+m}$ the $n + m$ first inputs of *Print* and Φ the frame at the start of the execution of *Print* in Q. We also let $m_i = \mathsf{sdec}(\xi_i\Phi, k_i)$, which verifies $\mathsf{msg}(m_i)$ and $m_i{\downarrow} \in \mathbb{B}$. Then if p is the integer whose binary representation is $m_{n+1}{\downarrow}, \ldots, m_{n+m}{\downarrow}$, the p^{th} clause of φ is falsified by v.

1.2 Enforcing positivity

Trace and session equivalence In this section we study to which extent the lower bound above can be obtained in the positive fragment. There is actually only one else branch in the process, precisely in the process $R_i(t)$, $i < n$:

$$\begin{aligned}R_i(t) = \mathsf{if}\ \mathsf{fst}(t) = \mathsf{fst}(t)\ \mathsf{then}\ &\overline{c}\langle\alpha\rangle.\,\overline{c}\langle\beta\rangle.\,\overline{c}\langle\beta\rangle\\ \mathsf{else}\ &\overline{c}\langle\alpha\rangle.\,\overline{c}\langle\beta\rangle.\,\overline{c}\langle\gamma\rangle\end{aligned}$$

for reminder, the role of this process is to output three messages that are indistinguishable from three fresh nonces *iff* t is not a pair. In particular there are ways to get rid of the else branch:

- *slightly extending the theory*:
  we add a binary constructor symbol h, a unary destructor TestPair and the rewrite rule

$$\mathsf{TestPair}(h(\langle x, y\rangle, z)) \to \mathsf{ok}$$

  for some constant ok. Then we replace $R_i(t)$ by the process

$$R'_i(t) = \overline{c}\langle h(t, \alpha)\rangle.\,\overline{c}\langle\beta\rangle.\,\overline{c}\langle\gamma\rangle$$

- *getting rid off the constructor-destructor semantics*:
  the reduction of the previous section holds in the altered semantics we use in the constructor destructor semantics, in particular where a conditional $\mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q$ executes its negative branch when a destruction failure occurs in u or v. Without conditions on destruction failures we can replace $R_i(t)$ by

$$R'_i(t) = \overline{c}\langle \mathsf{senc}(t, \alpha)\rangle.\, \overline{c}\langle \mathsf{senc}(\langle \mathsf{fst}(t), \mathsf{snd}(t)\rangle, \alpha)\rangle.\, \overline{c}\langle \gamma\rangle$$

Theorem E.4

There exists a constructor-destructor theory for which TraceEquiv and SessEquiv are coNEXP-hard for bounded positive processes whose channels are constants.

Labelled bisimilarity Now we study the case of Bisimilarity by proving Theorem E.2. This shows that we can get rid of the else branch of $R_i(t)$; however internal communications on private channels are needed to make the overall reduction work (in particular because the encoding of non-deterministic choice by encrypted public communications does not work for Bisimilarity). We actively use again the syntax extensions introduced in Chapter 6, Section 4.1.

We prove Theorem E.2 by reduction from SuccinctSAT in this extended calculus. We consider $\Gamma : \mathbb{B}^{m+2} \to \mathbb{B}^{n+1}$ a circuit encoding a formula φ with 2^m clauses and 2^n variables $x_0, \ldots, x_{2^n-1}$. We define two processes P and Q that are labelled bisimilar *iff* φ is unsatisfiable. They have the following form

$$P = c(x).\, \big(!^{3n}\, \mathit{Extract} \mid (\mathit{CheckSat}(x) + \textstyle\sum_{i=0}^{n} \mathit{CheckTree}_i(x) + \mathit{Print})\big)$$
$$Q = c(x).\, \big(!^{3n}\, \mathit{Extract} \mid (\mathit{CheckSat}(x) + \textstyle\sum_{i=0}^{n} \mathit{CheckTree}_i(x))\big)$$

where $!^a\, P = P \mid \cdots \mid P$ (a parallel copies). The intuition is very similar to the reduction for trace equivalence and equivalence by session:

- *Print* serves as a baseline for comparison
- *Extract* performs tree extractions upon request
- $\mathit{CheckTree}_i(x)$ is equivalent to *Print iff* x is not a complete binary tree of height in $[\![i, n]\!]$ with boolean leaves
- assuming x is a complete binary tree of height n with boolean leaves, writing v the valuation it thus encodes, $\mathit{CheckSat}(x)$ is equivalent to *Print iff* v satisfies all clauses of φ.

In the construction we let a constant c (that will serve as a public channel), a fresh name d (that will serve as a private channel to communicate with *Extract*) and three fresh names α, β, γ for the final three outputs of the processes.

Definition of *Extract* The process simply receives a pair and a boolean on d and outputs back the corresponding component:

$$\mathit{Extract} = d(\langle\langle x, y\rangle, b\rangle).\, ([b = b_0]\, \overline{d}\langle x\rangle \mid [b = b_1]\, \overline{d}\langle y\rangle)$$

In the other processes, we will then use the following shortcut for interacting with *Extract*:

$$x_k \leftarrow \mathit{Extr}(x_0, a_0, \ldots, a_{k-1}).\, P$$
$$\triangleq \overline{d}\langle\langle x_0, a_0\rangle\rangle.\, d(x_1) \ldots \overline{d}\langle\langle x_{k-1}, a_{k-1}\rangle\rangle.\, d(x_k).P$$

Definition of *Print* The process simply performs the following dummy operations, mimicking the control flow of the processes defined below:

$$Print = c(\_).\, c(\_).\, \overline{c}\langle\alpha\rangle.\, \overline{c}\langle\beta\rangle.\, \overline{c}\langle\gamma\rangle$$

Definition of $CheckTree_i$ The process is defined as follows for all terms x_0:

$$\begin{aligned} CheckTree_i(x_0) = {} & \mathsf{Choose}(a_0).\, \overline{d}\langle\langle x_0, a_0\rangle\rangle.d(x_1). \\ & \vdots \\ & \mathsf{Choose}(a_{i-1}).\, \overline{d}\langle\langle x_{i-1}, a_{i-1}\rangle\rangle.d(x_i). \\ & R_i(x_i) \end{aligned}$$

where the process $R_i(t)$ is defined as follows

$$\begin{aligned} &\text{if } i < n\text{: } R_i(t) = c(x).\, c(y).\, \overline{c}\langle\mathsf{senc}(t, \alpha)\rangle.\, \overline{c}\langle\mathsf{senc}(\langle x, y\rangle, \alpha)\rangle.\, \overline{c}\langle\beta\rangle \\ &R_n(t) = c(\_).\, c(\_).\, \overline{c}\langle\mathsf{senc}(t, \alpha)\rangle.\, \overline{c}\langle\mathsf{senc}(b_0, \alpha)\rangle.\, \overline{c}\langle\mathsf{senc}(t, \alpha)\rangle \end{aligned}$$

The definition of $R_i(t)$ is different from the previous reduction (Section 1.2), adding an interaction with the attacker with two public inputs. This is the only argument in the proof that significantly differs from the previous reduction, making it possible to test that the term t is a pair without extending the theory.

Lemma E.5

Let $i \in [\![1, n-1]\!]$ (resp $i = n$), a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{F}_0)$. Then the processes *Print* and $R_i(t)$ are *not* labelled bisimilar *iff* there exists $t_0, t_1 \in \mathcal{T}(\mathcal{F}, \mathcal{F}_0)$ such that $t = \langle t_0, t_1\rangle$ (resp. $t \in \mathbb{B}$).

Proof. The proof for $i = n$ essentially follows from the fact that the two frames

$$\begin{aligned} \Phi_1 &= \{\mathsf{ax}_1 \mapsto \mathsf{senc}(t, \alpha), \mathsf{ax}_2 \mapsto \mathsf{senc}(b_0, \alpha), \mathsf{ax}_3 \mapsto \mathsf{senc}(b_1, \alpha)\} \\ \Phi_2 &= \{\mathsf{ax}_1 \mapsto \alpha, \mathsf{ax}_2 \mapsto \beta, \mathsf{ax}_3 \mapsto \gamma\} \end{aligned}$$

are statically equivalent *iff* t, b_0, b_1 are pairwise disjoint terms, i.e. *iff* $t \notin \mathbb{B}$. Similarly the frames

$$\begin{aligned} \Phi_1 &= \{\mathsf{ax}_1 \mapsto \mathsf{senc}(t, \alpha), \mathsf{ax}_2 \mapsto \mathsf{senc}(\langle t_0, t_1\rangle, \alpha), \mathsf{ax}_3 \mapsto \beta\} \\ \Phi_2 &= \{\mathsf{ax}_1 \mapsto \alpha, \mathsf{ax}_2 \mapsto \beta, \mathsf{ax}_3 \mapsto \gamma\} \end{aligned}$$

are statically equivalent *iff* $t \neq \langle x, y\rangle$. In particular it easily follows that, if $i < n$ and $t \in \mathcal{T}(\mathcal{F}, \mathcal{F}_0)$, then A and B are labelled bisimilar *iff* there exists two terms $t_0, t_1 \in \mathcal{T}(\mathcal{F}, \mathcal{F}_0)$ such that $\langle t_0, t_1\rangle = t$. □

Definition of *CheckSat* The process $CheckSat(x)$ operates in the same way as its counterpart in Section 1.2: it guesses a clause of φ, recovers its variables and negation bits by

evaluating Γ, computes its valuation ω by three tree extractions from x, and outputs three terms that are statically equivalent to fresh names *iff* $\omega \neq b_1$.

$$\begin{aligned} \mathit{CheckSat}(x) = \; & \mathsf{Choose}(a_1) \ldots \mathsf{Choose}(a_m). \\ & \omega_1, x_1, \ldots, x_n \leftarrow \Gamma(a_1, \ldots, a_m, b_0, b_0). \\ & \omega_2, y_1, \ldots, y_n \leftarrow \Gamma(a_1, \ldots, a_m, b_0, b_1). \\ & \omega_3, y_1, \ldots, y_n \leftarrow \Gamma(a_1, \ldots, a_m, b_1, b_0). \\ & \omega'_1 \leftarrow \mathit{Extr}(x, x_1, \ldots, x_n). \\ & \omega'_2 \leftarrow \mathit{Extr}(x, y_1, \ldots, y_n). \\ & \omega'_3 \leftarrow \mathit{Extr}(x, z_1, \ldots, z_n). \\ & \omega \leftarrow \Gamma_v(\omega_1, \omega'_1, \omega_2, \omega'_2, \omega_3, \omega'_3). \\ & R(\omega) \end{aligned}$$

where $\Gamma_v : \mathbb{B}^6 \to \mathbb{B}$ is a circuit computing the boolean formula $\Gamma_v(x, x', y, y', z, z') = (x = x' \vee y = y' \vee z = z')$ and for all terms t:

$$R(t) = \overline{c}\langle \mathsf{senc}(t, \alpha) \rangle \, \overline{c}\langle \mathsf{senc}(b_1, \alpha) \rangle \, \overline{c}\langle \beta \rangle$$

Following a similar argument as in the construction of $\mathit{CheckTree}_n$, we have the following property:

Lemma E.6

For all terms t, the processes *Print* and $R(t)$ are labelled bisimilar *iff* $t \neq b_1$.

Correctness of the construction Now we prove that the following two points are equivalent

(i) φ is unsatisfiable

(ii) P and Q are labelled bisimilar

▷ Proof of (i) $\Longrightarrow$ (ii)

To prove (ii) it suffices to construct a symmetric relation $\mathcal{R}$ on extended processes such that $P \mathcal{R} Q$ and, for all extended processes A, B, $A \mathcal{R} B$ implies

- the frames of A and B are statically equivalent
- for all transitions $A \xrightarrow{\alpha} A'$, there exists $B \xrightarrow{\tau \cdots \tau \alpha \tau \cdots \tau} B'$ such that $A' \mathcal{R} B'$ or A' and B' are labelled bisimilar.

We define $\mathcal{R}$ as the smallest symmetric relation verifying the following properties:

1 $(\{\!\{P\}\!\}, \varnothing) \; \mathcal{R} \; (\{\!\{Q\}\!\}, \varnothing)$

2 $(\{\!\{!^{3n} \mathit{Extract} \mid P'(t)\}\!\}, \varnothing) \; \mathcal{R} \; (\{\!\{!^{3n} \mathit{Extract} \mid Q'(t)\}\!\}, \varnothing)$ for all terms $t \in \mathcal{T}(\mathcal{F}, \mathcal{F}_0)$ and

$$\begin{aligned} P'(t) &= \mathit{CheckSat}(t) + \textstyle\sum_{i=0}^{n} \mathit{CheckTree}_i(t) + \mathit{Print} \\ Q'(t) &= \mathit{CheckSat}(t) + \textstyle\sum_{i=0}^{n} \mathit{CheckTree}_i(t) \end{aligned}$$

3 $(\{\!\{!^{3n} \mathit{Extract}, P'(t)\}\!\}, \varnothing) \; \mathcal{R} \; (\{\!\{!^{3n} \mathit{Extract}, Q'(t)\}\!\}, \varnothing)$ for all terms $t \in \mathcal{T}(\mathcal{F}, \mathcal{F}_0)$

Let us prove that $\mathcal{R}$ verifies the expected properties. The inclusion of $\mathcal{R}$ into static equivalence is immediate. Then let two extended processes A adn B such that $A \mathcal{R} B$ and let us perform a case analysis on the hypothesis $A \mathcal{R} B$.

- *case 1:* $A = (\{\!\{P\}\!\}, \varnothing)$ and $B = (\{\!\{Q\}\!\}, \varnothing)$.
  Let a transition $A \xrightarrow{\alpha} A'$. Then $\alpha = \xi(\xi')$ where $\xi, \xi \in \mathcal{T}(\mathcal{F}, \mathcal{F}_0)$ and $\xi\!\downarrow\! = c$, and $A' = (\{\!\{!^{3n}\, \mathit{Extract} \mid P'(\xi')\}\!\}, \varnothing)$. Then it suffices to consider the transition $B \xrightarrow{\alpha} (\{\!\{!^{3n}\, \mathit{Extract} \mid Q'(\xi')\}\!\}, \varnothing)$.
- *case 1 + symmetry:* $A = (\{\!\{Q\}\!\}, \varnothing)$ and $B = (\{\!\{P\}\!\}, \varnothing)$.
  Symmetric to case 1.
- *case 2:* there exists $t \in \mathcal{T}(\mathcal{F}, \mathcal{F}_0)$ such that $A = (\{\!\{!^{3n}\, \mathit{Extract} \mid P'(t)\}\!\}, \varnothing)$ and $B = (\{\!\{!^{3n}\, \mathit{Extract} \mid Q'(t)\}\!\}, \varnothing)$.
  The conclusion is immediate by the clause 3.
- *case 2 + symmetry:* there exists $t \in \mathcal{T}(\mathcal{F}, \mathcal{F}_0)$ such that $A = (\{\!\{!^{3n}\, \mathit{Extract} \mid Q'(t)\}\!\}, \varnothing)$ and $B = (\{\!\{!^{3n}\, \mathit{Extract} \mid P'(t)\}\!\}, \varnothing)$.
  Symmetric to case 2.
- *case 3:* there exists $t \in \mathcal{T}(\mathcal{F}, \mathcal{F}_0)$ such that $A = (\{\!\{!^{3n}\, \mathit{Extract}, P'(t)\}\!\}, \varnothing)$ and $B = (\{\!\{!^{3n}\, \mathit{Extract}, Q'(t)\}\!\}, \varnothing)$.
  Consider a transition $A \xrightarrow{\alpha} A'$. Then $\alpha = \tau$ and we are in one of the following cases:
  - *case a:* $A' = (\{\!\{!^{3n}\, \mathit{Extract}, \mathit{CheckSat}(t)\}\!\}, \varnothing)$
    Then there exists a transition $B \xrightarrow{\tau} A'$, hence the conclusion by reflexivity of labelled bisimilarity.
  - *case b:* $A' = (\{\!\{!^{3n}\, \mathit{Extract}, \mathit{CheckTree}_i(t)\}\!\}, \varnothing)$ for some i
    Then there exists a transition $B \xrightarrow{\tau} A'$, hence the conclusion by reflexivity of labelled bisimilarity.
  - *case c:* $A' = (\{\!\{!^{3n}\, \mathit{Extract}, \mathit{Print}\}\!\}, \varnothing)$
    We now make a case analysis on the term t.
    - *case c.1:* t is not a complete binary tree of height n or more.
      Given a sequence of booleans $\vec{p}$, we say that $\vec{p}$ is a position of ξ_0 when $\xi_0 = \langle u_0, u_1 \rangle$ for some terms u_0, u_1 and either
      - $\vec{p}$ is empty, or
      - $\vec{p} = b_i \cdot \vec{p}'$ and $\vec{p}'$ is a position of u_i.

      In particular there exists a sequence $\vec{p} = p_1, \ldots, p_a$, $0 \leqslant a < n$, that is not a position of t. Without loss of generality we assume this position minimal, i.e. $a = 0$ or $p_1, \ldots, p_{a-1}$ is a position of t. In particular by choosing $\vec{p}$ in the non-deterministic choices of $\mathit{CheckTree}_a(t)$, there exists a trace

      $$\begin{aligned} B &\xrightarrow{\tau} (\{\!\{!^{3n}\, \mathit{Extract}, \mathit{CheckTree}_i(t)\}\!\}, \varnothing) \\ &\xRightarrow{\tau \cdots \tau} (\{\!\{!^{3n-a}\, \mathit{Extract}, R_i(t')\}\!\}, \varnothing) = B' \end{aligned}$$

      where t' is the node of t rooted at position $\vec{p}$. By construction t' is not a pair. In particular $(\{\!\{R_i(t')\}\!\}, \varnothing)$ and $(\{\!\{\mathit{Print}\}\!\}, \varnothing)$ are labelled bisimilar by Lemma E.5. Since the private channel d does not appear in neither *Print* nor $R_i(t')$, A' and B' are therefore labelled bisimilar, hence the conclusion.
    - *case c.2:* t is a complete binary tree of height n or more, and one of the nodes at depth n is not a leaf, or is a leaf that is not a boolean.
      This is the same reasoning as in the previous case, except that $a = n$.
    - *case c.3:* t is a complete binary tree of height n with boolean leaves.
      We let v the valuation of φ such that for all $i \in [\![0, 2^n - 1]\!]$, $v(x_i)$ is the i^{th} leaf of t

(ordered w.r.t. to a leftmost depth-first search in the tree t). Since φ is unsatisfiable by hypothesis, there exists $p \in [\![0, 2^m - 1]\!]$ such that v falsifies the p^{th} clause of φ. We let $\vec{p}$ the binary representation of p. In particular by choosing $\vec{p}$ in the non-deterministic choices of $\mathit{CheckSat}(t)$, there exists a trace

$$\begin{aligned} B &\xrightarrow{\tau} (\{\!\{!^{3n}\, \mathit{Extract}, \mathit{CheckSat}(t)\}\!\}, \varnothing) \\ &\xRightarrow{\tau \cdots \tau} (\{\!\{R(b_0)\}\!\}, \varnothing) = B' \end{aligned}$$

and A' and B' are labelled bisimilar by Lemma E.6.

- *case 3 + symmetry:* there exists $t \in \mathcal{T}(\mathcal{F}, \mathcal{F}_0)$ such that $A = (\{\!\{!^{3n}\, \mathit{Extract}, Q'(t)\}\!\}, \varnothing)$ and $B = (\{\!\{!^{3n}\, \mathit{Extract}, P'(t)\}\!\}, \varnothing)$.
  Then there exists a transition $B \xrightarrow{\tau} A'$, hence the conclusion by reflexivity of bisimilarity.

▷ Proof of $(ii) \implies (i)$

Let v a valuation of φ and let us prove that v falsifies a clause of φ. We also let t the complete binary tree of depth n whose i^{th} leaf (ordered w.r.t. a leftmost depth-first search in t) is $v(x_i)$. Since P and Q are labelled bisimilar by hypothesis, they are also trace equivalent. In particular consider the following trace t of P:

$$\begin{aligned} P &\xRightarrow{c(t)\tau\tau} (\{\!\{!^{3n}\, \mathit{Extract}, \mathit{Print}\}\!\}, \varnothing) \\ &\xRightarrow{c(c)c(c)\overline{c}\langle \mathsf{ax}_1\rangle\overline{c}\langle \mathsf{ax}_2\rangle\overline{c}\langle \mathsf{ax}_3\rangle} (\{\!\{!^{3n}\, \mathit{Extract}\}\!\}, \Phi) \end{aligned}$$

where $\Phi = \{\mathsf{ax}_1 \mapsto \alpha, \mathsf{ax}_2 \mapsto \beta, \mathsf{ax}_3 \mapsto \gamma\}$. We deduce that there exists a trace t' of Q such that $t \sim t'$. Since t is a complete binary tree of depth n with boolean leaves, we deduce that t' has the following form:

$$\begin{aligned} Q &\xRightarrow{c(t)\tau\tau} (\{\!\{!^{3n}\, \mathit{Extract}, \mathit{CheckSat}(t)\}\!\}, \varnothing) \\ &\xRightarrow{\tau \cdots \tau} (\{\!\{R(\omega)\}\!\}, \varnothing) \\ &\xRightarrow{c(c)c(c)\overline{c}\langle \mathsf{ax}_1\rangle\overline{c}\langle \mathsf{ax}_2\rangle\overline{c}\langle \mathsf{ax}_3\rangle} (\{\!\{0\}\!\}, \Phi') \end{aligned}$$

where, if $\vec{a} = a_1, \ldots, a_m$ are the non-deterministic choices performed in this execution of $\mathit{CheckSat}(t)$ and $p \in [\![0, 2^m - 1]\!]$ is the integer whose binary representation is $\vec{a}$, $\omega \in \mathbb{B}$ is the valuation of the p^{th} clause of φ by v. In particular the fact that Φ and Φ' are statically equivalent implies that $\omega = b_0$, hence the conclusion.

2 Lower bounds in the pure fragment

We prove in this section the complexity lower bounds that we obtained for the pure pi calculus, relying again on tools presented in Appendix D.2 (winning strategies and syntax extensions).

> ***Theorem 7.5*** *(equivalences in the pure pi calculus)*
> In the pure pi-calculus, Bisimilarity (resp. TraceEquiv and SessEquiv) is PSPACE complete (resp. Π_2 complete) for bounded processes, and for bounded positive processes.

2.1 Trace equivalence

To show that trace equivalence is Π_2-hard we proceed by a reduction from QBF_2 (see definition in Chapter 1, Section 4). Let thus φ be a boolean formula whose variables are partitioned into $\{\vec{x}\} \cup \{\vec{y}\}$. Our goal is to construct two processes A and B such that:

$$A \approx_t B \quad \text{if and only if} \quad \forall \vec{x}.\exists \vec{y}.\varphi(\vec{x},\vec{y}) = 1 \tag{E.1}$$

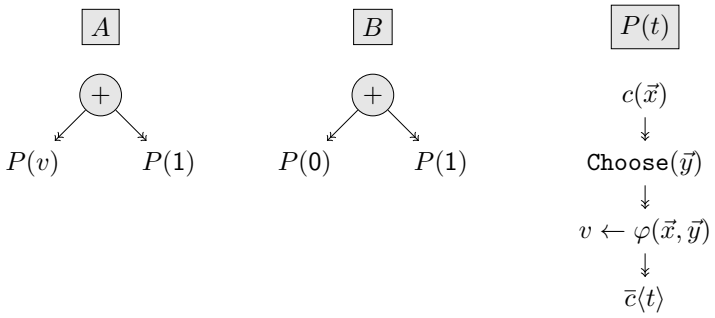


Figure E.1 Schematic definition of A and B

Consider three distinct names $c, \mathsf{0}, \mathsf{1} \in \mathcal{F}_0$ (the last two modelling booleans for the syntax extension of circuit evaluation, see Chapter 6, Section 4.1). Processes A and B are depicted in Figure E.1. Intuitively, the process $P(t)$ (where t is a term which may depend on the variables bound by P) gets a valuation of $\vec{x}$ from the attacker, internally chooses a valuation of $\vec{y}$, computes the value of $\varphi(\vec{x},\vec{y})$ using rule ValuateEq, and outputs t. From that it is quite easy to see that A and B have the same set of traces *iff* for all valuation of $\vec{x}$, there exists a valuation of $\vec{y}$ such that $\varphi(\vec{x},\vec{y}) = \mathsf{0}$. More formally, we define $P(t)$, A and B as follows.

$$\begin{aligned} P(t) &\overset{def}{=} c(\vec{x}).\ \mathtt{Choose}(\vec{y}).\ v \leftarrow \varphi(\vec{x},\vec{y}).\ \overline{c}\langle t\rangle \\ A &\overset{def}{=} P(v) + P(\mathsf{1}) \\ B &\overset{def}{=} P(\mathsf{0}) + P(\mathsf{1}) \end{aligned}$$

Theorem E.7

In the pure pi-calculus, TraceEquiv is Π_2-hard for bounded positive processes.

Proof. We prove (E.1) by double implication.

$\Rightarrow$ Suppose that $A \not\approx_t B$. By a quick case analysis, we obtain $B \overset{\varepsilon}{\to} (\{\!\{P(0)\}\!\}, \varnothing) \overset{\mathsf{tr}}{\Rightarrow} (\varnothing, \{\mathsf{ax}_1 \mapsto \mathsf{0}\})$ where $\mathsf{tr} = c(\vec{t}).\overline{c}\langle \mathsf{ax}_1\rangle$ for some messages $\vec{t}$, such that for all reduction $A \overset{\mathsf{tr}}{\Rightarrow} (\mathcal{C}, \Phi)$ the frames $\{\mathsf{ax}_1 \mapsto \mathsf{0}\}$ and Φ are not statically equivalent. In particular, for all $\vec{y} \subseteq \mathbb{B}$, by choosing $\Phi = \{\mathsf{ax}_1 \mapsto \varphi(\vec{t},\vec{y})\}$ reachable from A, we obtain $\varphi(\vec{t},\vec{y}) \neq \mathsf{0}$, hence the result.

$\Leftarrow$ Conversely, suppose that exists exists $\vec{x} \subseteq \mathbb{B}$ such that $\varphi(\vec{x},\vec{y}) = \mathsf{1}$ for all $\vec{y} \subseteq \mathbb{B}$. Then the trace $B \overset{\varepsilon}{\Rightarrow} (\varnothing, \{\mathsf{ax}_1 \mapsto \mathsf{0}\})$ cannot be matched in A and therefore $A \not\approx_t B$. □

2.2 Labelled bisimilarity

We now prove that labelled bisimilarity is PSPACE-hard for the positive pure pi calculus by reduction from QBF. This is more involved as QBF allows arbitrary quantifier alternation. Let φ be a boolean formula whose variables are partitioned into $\{x_1, \ldots, x_n\} \cup \{y_1, \ldots, y_n\}$ for some $n \in \mathbb{N}$. We construct (in polynomial time in the size of φ and n) two processes A and B such that:

$$A \not\approx_l B \text{ if and only if } \exists x_1 \forall y_1 \ldots \exists x_n \forall y_n.\ \varphi(x_1, \ldots, x_n, y_1, \ldots, y_n) = 1 \tag{E.2}$$

Both QBF and labelled bisimilarity may be seen as bisimulation games: an attacker plays the $\exists$-quantifiers (selects a transition in a process) whereas a defender responds with the $\forall$-quantifiers (tries to find a similarly-labelled sequence of transitions in the other process). The role of A and B is to implement this intuitive connection: the attacker moves will be simulated by public inputs $c(x_i)$ and the defender responses by instructions $\texttt{Choose}(z_i).c(y_i)$. The structure of A and B is then designed to constrain the moves of the two players so that the winning condition of the attacker is exactly $\exists x_1 \forall y_1 \ldots \exists x_n \forall y_n.\ \varphi(\vec{x}, \vec{y}) = 1$.

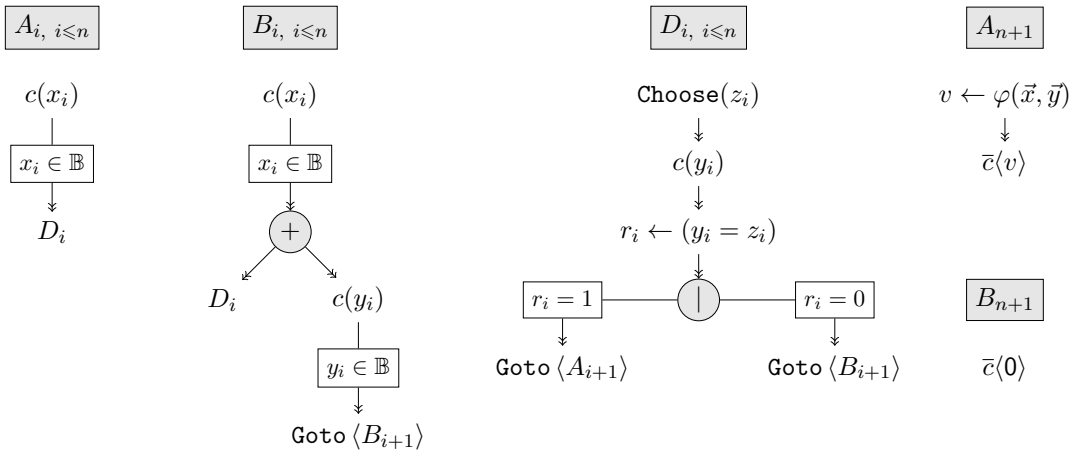


Figure E.2 Schematic definition of A_i and B_i

A and B are defined inductively by processes A_i, B_i and D_i, depicted in Figure E.2, structured in a way that, in a bisimulation game:

1. the attacker chooses the instance of x_i;
2. the defender chooses the instance of z_i and can force the attacker to instanciate y_i with the same value (the attacker not doing so allows for a trivial victory of the defender).

We fix a family of private channels $(c_P)_P \subseteq \mathcal{N}$ indexed by processes P which will be used to simulate instructions $\texttt{Goto}\,\langle P \rangle$. We use a shortcut $\overline{d}\langle \overrightarrow{t}^{\,p} \rangle$ for an indexed sequence of terms $(t_i)_i$ to denote the sequence of outputs:

$$\overline{d}\langle \overrightarrow{t}^{\,p} \rangle \stackrel{def}{=} \overline{d}\langle t_1 \rangle \ldots \overline{d}\langle t_p \rangle$$

and a similar notation for sequences of inputs. Then the Goto feature is implemented as

follows, allowing for passing and receiving program states through parallel processes:

$$\texttt{Goto}\,\langle A_i\rangle \stackrel{def}{=} \overline{c_{A_i}}\langle \vec{x}^i, \vec{y}^i\rangle \qquad \texttt{Goto}\,\langle B_i\rangle \stackrel{def}{=} \overline{c_{B_i}}\langle \vec{x}^i, \vec{y}^i\rangle$$
$$\texttt{GetEnv}\,\langle A_i\rangle .P \stackrel{def}{=} c_{A_i}(\vec{x}^i, \vec{y}^i) \qquad \texttt{GetEnv}\,\langle B_i\rangle .P \stackrel{def}{=} c_{B_i}(\vec{x}^i, \vec{y}^i)$$

Formally the processes A_i, B_i and D_i are defined below. We stress out that A and B are closed (as required) but that A_i, B_i and D_i are not in general. Fixing a public channel $c \in \mathcal{F}_0$, we write:

$$\begin{aligned}
\forall i \leqslant n,\ A_i &\stackrel{def}{=} c(x_i).\ x_i \leftarrow x_i.\ D_i \\
\forall i \leqslant n,\ B_i &\stackrel{def}{=} c(x_i).\ x_i \leftarrow x_i.\ (D_i + (c(y_i).\ y_i \leftarrow y_i.\ \texttt{Goto}\,\langle B_{i+1}\rangle)) \\
A_{n+1} &\stackrel{def}{=} v \leftarrow \varphi(\vec{x}, \vec{y}).\ \overline{c}\langle v\rangle \\
B_{n+1} &\stackrel{def}{=} \overline{c}\langle 0\rangle \\
D_i &\stackrel{def}{=} \texttt{Choose}(z_i).\ c(y_i).\ r_i \leftarrow (y_i = z_i). \\
&\qquad ((\text{if } r_i = 1 \text{ then } \texttt{Goto}\,\langle A_{i+1}\rangle) \\
&\qquad\quad \mid (\text{if } r_i = 0 \text{ then } c(y_i).\ y_i \leftarrow y_i.\ \texttt{Goto}\,\langle B_{i+1}\rangle))
\end{aligned}$$

As in the reduction for trace equivalence (Section 2.1), the $\texttt{Choose}(\alpha)$ simulates nondeterministic choice among $\mathbb{B}$; the construction $\alpha \leftarrow \alpha$, which may seem useless, encodes the test $\alpha \in \mathbb{B}$. Finally, we define A and B by putting the auxiliary processes in parallel and connecting the `Goto`'s to the `getEnv`'s:

$$A \stackrel{def}{=} A_1 \mid C \qquad B \stackrel{def}{=} B_1 \mid C \qquad C \stackrel{def}{=} \prod_{i=2}^{n+1} (\texttt{GetEnv}\,\langle A_i\rangle .A_i) \ \mid \ \prod_{i=2}^{n+1} (\texttt{GetEnv}\,\langle B_i\rangle .B_i)$$

A and B can be computed in time $\mathcal{O}(n^2 + |\varphi|)$ in a straightforward way. The formal proof of the reduction is detailed below.

Theorem E.8

In the pure pi-calculus, Bisimilarity is is PSPACE-hard for bounded positive processes.

Proof. We recall that, again, our proof uses the advanced winning strategy framework presented in Appendix D, Section 2.1. We prove (E.2) by double implication.

⇒ By contraposition, suppose that $\forall x_1 \exists y_1 \ldots \forall x_n \exists y_n.\ \varphi(x_1, \ldots, x_n, y_1, \ldots, y_n) = 1$. For convinience we use a notation for subprocess extraction: is ℓ is a position of a process C, then the subprocess of C at position ℓ (which may not be closed) is denoted by $C_{|\ell}$. Then if:

$$C_i = \prod_{j=i+1}^{n+1} (\texttt{GetEnv}\,\langle A_j\rangle .A_j) \mid \prod_{j=i+1}^{n+1} (\texttt{GetEnv}\,\langle B_j\rangle .B_j)$$

we define $\mathcal{R}$ the smallest reflexive symmetric relation on closed extended processes such that:

1. $(A_i \mid C_i)(\vec{x}^{i-1}, \vec{y}^{i-1})\mathcal{R}(B_i \mid C_i)(\vec{x}^{i-1}, \vec{y}^{i-1})$
   if $\forall x_i \exists y_i \ldots \forall x_n \exists y_n, \varphi(\vec{x}, \vec{y})$.

2. $(A_{i|\ell} \mid C_i)(\overrightarrow{x}^i, \overrightarrow{y}^{i-1})\mathcal{R}(B_{i|\ell} \mid C_i)(\overrightarrow{x}^{i-1}, \overrightarrow{y}^{i-1})$
   if $\ell \in \{0, 0.0\}$ and $\exists y_i \ldots \forall x_n \exists y_n, \varphi(\vec{x}, \vec{y})$.
3. $(B_{i|0.0.1.\ell} \mid C_i)(\overrightarrow{x}^i, \overrightarrow{y}^i)\mathcal{R}(D_{i|0.\ell} \mid C_i)(\overrightarrow{x}^i, \overrightarrow{y}^i)$
   if $\ell \in \{\varepsilon, 0\}$ and $\forall x_{i+1} \exists y_{i+1} \ldots \forall x_n \exists y_n, \varphi(\vec{x}, \vec{y})$.

Then one can verify that $\mathcal{R}$ is a bisimulation up to $\rightsquigarrow$, and $A\mathcal{R}B$ by hypothesis, hence the $A \approx_l B$.

$\Leftarrow$ If we suppose that $\exists x_1 \forall y_1 \ldots \exists x_n \forall y_n.\ \varphi(x_1, \ldots, x_n, y_1, \ldots, y_n) = 1$, then one can define a labelled attack $\mathcal{S}$ such that $B\mathcal{S}A$. We omit the concrete construction as it is analogous to that of $\mathcal{R}$ above; all in all this gives the conclusion $A \not\approx_l B$. □

3 co-NEXP hardness for simple patterned processes

In this section we prove the coNEXP-hardness of the problem studied in [CCD15a], namely trace equivalence of patterned, simple, acyclic, type-compliant processes with a theory limited to symmetric encryption and pairs (Theorem 7.8). Due to Proposition 7.19, all results for this fragment also apply for labelled bisimilarity and equivalence by session, and diff equivalence.

3.1 Formalising the hypotheses

First of all we formalise the hypotheses of *type compliance* and *acyclicity* that were not detailed in the body of the paper.

Type compliance A *type system* consists of a set of *atomic types* $\mathcal{T}_0$ and a typing function δ mapping any term (that may contain variables) to types τ defined by the following grammar

$$\tau, \tau' ::= \qquad \tau \quad (\in \mathcal{T}_0) \qquad \langle \tau, \tau' \rangle \qquad \mathsf{senc}(\tau, \tau')$$

There should be infinitely many constants, names and variables of any type. We say that a type system is *structure preserving* if it additionally verifies the following property for all constructors f (i.e. f is either symmetric encryption or pairing) and all terms u, v:

$$\delta(f(u, v)) = f(\delta(u), \delta(v))\,.$$

In particular a structure-preserving type system is defined by the image of δ on the set of atomic data, i.e. $\mathcal{F}_0 \cup \mathcal{N} \cup \mathcal{X}$.

If P is a process we define $St(P)$ the set of subterms of patterns or outputs appearing in P^2, where P^2 is the process obtained after replacing all replicated subprocess $!\,R$ of P by $R \mid R$. This duplication is used to materialise syntactically that $!\,R$ implies several parallel copies of R. To avoid name and variable capture when studying the unifiability of terms of $St(P)$, we assume that all new names and input variables of P^2 have been alpha-renamed in a type-preserving manner. We then define the set of *encrypted subterms* of P by:

$$ESt(P) = \{u \in St(P) \mid u \text{ is of the form } \mathsf{senc}(v, w)\}\,.$$

We say that a process P is *type-compliant* w.r.t. a structure preserving type system if for all unifiable terms $t, t' \in ESt(P)$, we have $\delta(t) = \delta(t')$.

Acyclicity A process P is said *acyclic* when its *dependency graph* is acyclic, where this dependency graph G is defined in the rest of this section. Two definitions of G are given in [CCD15a]: a preliminary one sufficient to obtain decidability and a refined variant which allows more protocols to verify the acyclicity hypothesis. Here we only need the simple version, hence the lighter presentation compared to [CCD15a].

First of all we assume a structure preserving type system $(\mathcal{T}_0, \delta)$ and define two important syntactic classes of types. Intuitively *public types* (resp. *private types*[1]) correspond to values that are always (resp. never) deducible by the adversary. Naturally a type may be neither public nor private. Formally we say a type τ is

- *public* if there exist no names n occuring in P such that $\delta(n)$ is a syntactic subterm of the type τ.
- *private* if it is atomic, $\tau \neq \delta(a)$ for all variables and constants a appearing in P, and $\tau \notin \mathsf{pp}(u)$ for any term u occurring in P.

Regarding the last item, the set $\mathsf{pp}(u)$ of types in *plaintext position* in a term u is defined inductively as follows:

$$\begin{aligned}
&\mathsf{pp}(u) = \{\delta(u)\} && \text{if } u \in \mathcal{F}_0 \cup \mathcal{N} \cup \mathcal{X} \\
&\mathsf{pp}(u) = \{\delta(u)\} \cup \mathsf{pp}(u_0) \cup \mathsf{pp}(u_1) && \text{if } u = \langle u_0, u_1 \rangle \\
&\mathsf{pp}(u) = \{\delta(u)\} \cup \mathsf{pp}(u_0) && \text{if } u = \mathsf{senc}(u_0, u_1)
\end{aligned}$$

This covers all cases since, in the patterned fragment, no destructors appear explicitly in the process.

We then define a set $\varrho_{\mathsf{io}}(\tau)$ that characterises the types of the terms that can be deduced by the adversary from a term of type τ. More precisely $\varrho_{\mathsf{io}}(\tau)$ contains elements of the form

$$\tau' \# S \qquad\qquad \tau' \text{ type and } S \text{ set of non-private types}$$

meaning that a term of type τ' may be deducible provided prior deduction of terms of type in S. Formally $\varrho_{\mathsf{io}}(\tau) = \varrho^{\varnothing}_{\mathsf{io}}(\tau)$ where

$$\begin{aligned}
&\varrho^S_{\mathsf{io}}(\tau) = \{\tau \# S\} && \text{if } \tau \in \mathcal{T}_0 \\
&\varrho^S_{\mathsf{io}}(\langle \tau_0, \tau_1 \rangle) = \varrho^S_{\mathsf{io}}(\tau_0) \cup \varrho^S_{\mathsf{io}}(\tau_1) && \\
&\varrho^S_{\mathsf{io}}(\mathsf{senc}(\tau_0, \tau_1)) = \{\mathsf{senc}(\tau_0, \tau_1) \# S\} && \text{if } \tau_1 \text{ private} \\
&\varrho^S_{\mathsf{io}}(\mathsf{senc}(\tau_0, \tau_1)) = \{\mathsf{senc}(\tau_0, \tau_1) \# S\} \cup \varrho^{S \cup \{\tau_1\}}_{\mathsf{io}}(\tau_0) && \text{otherwise}
\end{aligned}$$

Using this, we eventually define the dependency graph G. The node of this graph are the input and output instructions of the process P; for the sake of reference we tag each of them with a label ℓ with the following syntax

$$\ell : c(u).P \qquad\qquad \ell : \overline{c}\langle u \rangle.P$$

and the nodes of G should rather be seen as the labels themselves. The edges of G are then defined as follows; there is an edge $\ell' \to \ell$ in the graph whenever

[1] private types were called *honest types* in [CCD15a].

- *sequential edges:*
  $\ell : \alpha.Q$ is a subprocess of P and $\ell' : \beta$ is the first input or output instruction appearing in Q (if any)
- *pattern edges:*
  $\ell : \overline{c}\langle u\rangle$ and $\ell' : d(v)$ are actions occurring in P and there exists $\tau\#S \in \varrho_{\mathsf{io}}(\delta(u))$, τ non-public type, and $\tau'\#S' \in \varrho_{\mathsf{io}}(\delta(v))$ such that $\tau = \tau'$ or $\tau \in S'$.
- *deduction edges:*
  $\ell : \overline{c}\langle u\rangle$ and $\ell' : \overline{d}\langle v\rangle$ are actions occurring in P and there exists $\tau\#S \in \varrho_{\mathsf{io}}(\delta(u))$, τ non-public type, and $\tau'\#S' \in \varrho_{\mathsf{io}}(\delta(v))$ such that $\tau \in S'$.

An edge $\ell' \to \ell$ model that it may be necessary to execute the action labelled ℓ before the one labelled ℓ' to be able to perform some attacker actions in a trace. Sequential edges model the dependencies between non-concurrent actions. Pattern edges model that it may be necessary to execute a given output u for the attacker to be able to produce a term complying to a given pattern v: for example $u = \mathsf{senc}(m, k)$ and $v = \mathsf{senc}(x, k)$ for some name k. Deduction edges model that an output u may be necessary to deduce a subterm of an output v: for example $u = k$ and $v = \mathsf{senc}(m, k)$ for some name k.

3.2 Proof of coNEXP hardness

We now prove Theorem 7.8, by reduction from SuccinctSAT as annouced in the proof sketch in the body of the paper. We therefore let a circuit $\Gamma : \mathbb{B}^{m+2} \to \mathbb{B}^{n+1}$ encoding a formula φ whose variables are $x_0, \ldots, x_{2^n-1}$, and the set of booleans $\mathbb{B}$ is modelled by two distinct constants $\mathsf{0}, \mathsf{1}$ for false and true, respectively. We then construct two patterned, simple, type-compliant, acyclic processes with atomic keys P_0 and P_1 that are trace equivalent iff φ is unsatisfiable.

Construction of the processes For simplicity we use a single channel c in the definition of P_0 and P_1 which can easily be converted to simple processes by using a different channel for each parallel subprocess. More precisely $P_i = P(b_i)$ where $P(t)$ has the following form

$$P(t) = \mathit{Extract} \mid \mathit{Eval} \mid \mathit{Init} \mid \mathit{CheckSat} \mid \mathit{Final}(t)$$

Intuitively the processes operate as follows:

- Similarly to the reduction presented in Appendix 1.1, the processes *Extract* and *Eval* are utilitaries that perform tree extractions and circuit evaluations, respectively, upon request from other parallel processes.
- *Init* is a non-replicated process that receives a input x from the attacker that is expected to be a valuation of φ, modelled by a complete binary tree of height n whose leaves are booleans. As in Appendix 1.1, the nodes of this tree are modelled by pairs. After receiving x the process reveals its encryption under a dedicated secret key, forcing the attacker to commit once and for all on this valuation.
- *CheckSat* is a replicated process whose instances can be used to verify, one clause at a time, that the valuation chosen by the attacker in *Init* satisfies φ. Each time a clause is successfully verified this way, say the i^{th} clause of φ, the encryption of the binary representation of i is revealed to the adversary.

- In particular, after verifying all clauses of φ the attacker obtains the encryption of all integers of $[\![0, 2^m - 1]\!]$. Hence the process $Final(t)$ checks that the attacker knows them all (using a dichotomy-based procedure to avoid the process containing explicitly an exponential number of verification steps) and eventually outputs t.
  The definition will also use the following atomic data:

const. :	$0, 1$	booleans
	nil	empty list
names :	$r^{i,j}_{Extr}, a^{i,j}_{Extr}, 1 \leqslant i \leqslant 3, 1 \leqslant j \leqslant n$	encr. keys for *Extract*
	$k^0_{Final}, \ldots, k^m_{Final}$	encr. keys for *Final*
	k	encr. key for *Init*

Definition of *Extract* Following the same intuition as in Appendix 1.1, we could define *Extract* as a process

$$!^{ch}\, c(\mathsf{senc}(\langle\langle x, y\rangle, 0\rangle, k)).\overline{c}\langle\mathsf{senc}(x, k')\rangle$$
$$\mid\, !^{ch}\, c(\mathsf{senc}(\langle\langle x, y\rangle, 1\rangle, k)).\overline{c}\langle\mathsf{senc}(y, k')\rangle$$

for some names k, k'. However there are several reasons why the situation requires a more complex construction. First, this process does not verify the acyclicity property: a process would need to perform several round-trip with *Extract* to extract a leaf from a binary tree, node by node, which highlights a circular dependency (more precisely a *pattern dependency* from the outputs to the inputs). Type-compliance would not be satisfiable neither. This problem can be solved by stratifying the construction: *Extract* is split into n replicated processes, each performing pair extractions for trees of a given height. The second problem is that, since all answers are encrypted with the same key, we need a form of marker to identify the different requests. A solution is to add a nonce to each request which is forwarded together with the answer. Altogether we have

$$Extract = Extract_1 \mid Extract_2 \mid Extract_3$$

where $Extract_i = \prod_{j=1}^{n} E^j_i$ where for all $j \in [\![1, n]\!]$:

$$E^j_i = !^{ch}\, c(\mathsf{senc}(\langle\langle x, y\rangle, z, 0\rangle, r^{i,j}_{Extr})).\overline{c}\langle\mathsf{senc}(\langle x, z\rangle, a^{i,j}_{Extr})\rangle \mid$$
$$!^{ch}\, c(\mathsf{senc}(\langle\langle x, y\rangle, z, 1\rangle, r^{i,j}_{Extr})).\overline{c}\langle\mathsf{senc}(\langle y, z\rangle, a^{i,j}_{Extr})\rangle$$

In the other processes, if $t, t_1, \ldots, t_n$ are terms and $i \in [\![1, 3]\!]$, branch extractions are written $u_n \leftarrow Extr_i(t, t_1, \ldots, t_n).\, P$ instead of:

$$\begin{array}{l}
\mathsf{new}\ r.\, \overline{c}\langle\mathsf{senc}(\langle t, r, t_1\rangle, r^{i,1}_{Extr})\rangle.\, c(\mathsf{senc}(\langle u_1, r\rangle, a^{i,1}_{Extr})).\\
\overline{c}\langle\mathsf{senc}(\langle u_1, r, t_2\rangle, r^{i,2}_{Extr})\rangle.\, c(\mathsf{senc}(\langle u_2, r\rangle, a^{i,2}_{Extr})).\\
\qquad\vdots\\
\overline{c}\langle\mathsf{senc}(\langle u_{n-1}, r, t_n\rangle, r^{i,n}_{Extr})\rangle.\, c(\mathsf{senc}(\langle u_n, r\rangle, a^{i,n}_{Extr})).\, P
\end{array}$$

Definition of *Eval* The definition of *Eval* follows the same intuition as in Appendix 1.1, but using nonces to mark messages as in the previous paragraph. As before the gates of a circuit are tuples (e_1, e_2, f, e_3, e_4) where e_1, e_2 are the input edges, e_3, e_4 are the output edges, and $f : \mathbb{B}^2 \to \mathbb{B}$ is the gate boolean function Given $G(C)$ is the set of gates of a circuit C and for each edge e we associate a fresh name k_e. Such a circuit C is then translated as follows into a process:

$$\begin{array}{ll} [\![C]\!] = \prod_{g \in G(C)} [\![g]\!] & \\ [\![(e_1, e_2, f, e_3, e_4)]\!] = \prod_{b,b' \in \mathbb{B}} !^{ch} & c(\mathsf{senc}(\langle b, z\rangle, k_{e_1})). \\ & c(\mathsf{senc}(\langle b', z\rangle, k_{e_2})). \\ & \overline{c}\langle \mathsf{senc}(\langle f(b, b'), z\rangle, k_{e_3})\rangle. \\ & \overline{c}\langle \mathsf{senc}(\langle f(b, b'), z\rangle, k_{e_4})\rangle \end{array}$$

We then let $\mathit{Eval} = [\![\Gamma_1]\!] \mid [\![\Gamma_2]\!] \mid [\![\Gamma_3]\!] \mid [\![\Gamma_v]\!]$ where $\Gamma_1, \Gamma_2, \Gamma_3$ are three copies of Γ with fresh edges and Γ_v is a circuit computing the boolean function

$$\left\{ \begin{array}{ccc} \mathbb{B}^6 & \to & \mathbb{B} \\ (x_1, b_1, x_2, b_2, x_3, b_3) & \mapsto & (x_1 = b_1 \vee x_2 = b_2 \vee x_3 = b_3) \end{array} \right.$$

For the sake of succinctness, when an other processes interacts with *Eval* to, say, compute the a circuit C whose input edges (resp. output edges) are $e_1, \ldots, e_p$ (resp. $f_1, \ldots, f_q$), we write

$$x_1, \ldots, x_q \leftarrow C(t_1, \ldots, t_p).\, P$$

instead of:

$$\begin{array}{l} \mathsf{new}\ r.\, \overline{c}\langle \mathsf{senc}(\langle t_1, r\rangle, e_1)\rangle \ldots \overline{c}\langle \mathsf{senc}(\langle t_p, r\rangle, e_p)\rangle. \\ c(\mathsf{senc}(\langle x_1, r\rangle, f_1)) \ldots c(\mathsf{senc}(\langle x_q, r\rangle, f_q)).\, P \end{array}$$

Definition of *Init* This process simply forces the attacker to commit on a value x that will allow to violate equivalence *iff* it is a complete binary tree of height n with boolean leaves modelling a valuation satisfying φ. Formally

$$\mathit{Init} = c(x).\overline{c}\langle \mathsf{senc}(x, k)\rangle$$

This is the only ciphertext encrypted by k produced by the process and the attacker thus cannot forge any others. All processes that use the commited value x will therefore first receive an input encrypted by k, which can thus only be this one.

Definition of *CheckSat* This replicates a process that first retrieves the value x committed in *Init*, then receives an integer i chosen by the attacker, computes the valuation of the i^{th} clause w.r.t. x by interacting with *Eval* and *Extract*, and finally output the i encrypted with the key of *Final* if the clause is satisfied by x. Integers are represented in binary, using a

linked list $\langle b_1, \ldots, b_m, \mathsf{nil}\rangle$.

$$\begin{aligned} \mathit{CheckSat} = \ !^{ch}\ & c(\mathsf{senc}(x,k)).\\ & c(\langle b_1,\ldots,b_m,\mathsf{nil}\rangle).\\ & \omega_1, x_1, \ldots, x_n \leftarrow \Gamma_1(b_1,\ldots,b_m,\mathsf{0},\mathsf{0}).\\ & \omega_2, y_1, \ldots, y_n \leftarrow \Gamma_2(b_1,\ldots,b_m,\mathsf{0},\mathsf{1}).\\ & \omega_3, z_1, \ldots, z_n \leftarrow \Gamma_3(b_1,\ldots,b_m,\mathsf{1},\mathsf{0}).\\ & \omega'_1 \leftarrow \mathit{Extr}_1(x, x_1, \ldots, x_n).\\ & \omega'_2 \leftarrow \mathit{Extr}_2(x, y_1, \ldots, y_n).\\ & \omega'_3 \leftarrow \mathit{Extr}_3(x, z_1, \ldots, z_n).\\ & \mathsf{1} \leftarrow \Gamma_v(\omega_1, \omega'_1, \omega_2, \omega'_2, \omega_3, \omega'_3).\\ & \overline{c}\langle\mathsf{senc}(\langle b_1,\ldots,b_m,\mathsf{nil}\rangle, k^m_{\mathit{Final}})\rangle \end{aligned}$$

Definition of *Final* The process $\mathit{Final}(t)$ gathers integers sent by *CheckSat* and outputs t if when the whole set $[\![0, 2^m - 1]\!]$ has been received, which is only possible if it has been verified that all clauses of φ were satisfied by the valuation committed in *Init*. For that $\mathit{Final}(t)$ gathers pairs of integers that differ only by their least significant bits, and then reveals the encryption of these integers with this bit truncated. It then suffices to iterate this operation until the encryptions of $\mathsf{0}$ and $\mathsf{1}$ are eventually revealed. Formally

$$\mathit{Final}(t) = \mathit{Final}_1 \mid \cdots \mid \mathit{Final}_m \mid F(t)$$

where $F(t) = c(\mathsf{senc}(\mathsf{nil}, k^0_{\mathit{Final}})).\overline{c}\langle t\rangle$ and for all $i \in [\![1, m]\!]$

$$\begin{aligned} \mathit{Final}_i = \ !^{ch}\ & c(\mathsf{senc}(\langle \mathsf{0}, x_2, \ldots, x_i, \mathsf{nil}\rangle, k^i_{\mathit{Final}})).\\ & c(\mathsf{senc}(\langle \mathsf{1}, x_2, \ldots, x_i, \mathsf{nil}\rangle, k^i_{\mathit{Final}})).\\ & \overline{c}\langle\mathsf{senc}(\langle x_2,\ldots,x_i,\mathsf{nil}\rangle, k^{i-1}_{\mathit{Final}})\rangle \end{aligned}$$

Proof of type compliance and acyclicity Now we show that $P(b)$, $b \in \mathbb{B}$, indeed satisfies the hypotheses of the theorem. That is, we define a structure-preserving type system $(\mathcal{T}_0, \delta)$ and show that $P(b)$ is type-compliant and acyclic w.r.t. it.

On the size of the type system In terms of complexity, the type system *is part of the input of the problem*: as it can be observed in the proof of Theorem 7.7, the complexity of the decision procedure depends on its size. In particular for our reduction it should be ensured that $(\mathcal{T}_0, \delta)$ is of polynomial size.

Unfortunately we need a type for boolean trees of height n, which is a type of size 2^n. However when inspecting the details of the decision procedure in [CCD15a] we observe that:

- The coNEXP complexity is obtained by applying a coNP decidability result in the bounded fragment (Theorem 7.20) to an exponential number of sessions of the process.
- This exponential number N of sessions is to be computed from the type system and the process. More precisely there exist two polynomials A, B such that

  $$N = A(||\mathsf{io}_P||)^{B(|P|)}$$

  where $|P|$ is number of instructions of the process P and $||\mathsf{io}_P||$ is the maximal size (i.e. the maximal number of constructor symbols) of an input or an output in a trace where input variables x are only instanciated by terms t such that $\delta(x) = \delta(t)$.

In particular the complexity analysis is robust to $||\mathsf{io}_P||$ being exponential in the size of the parameters. Since $||\mathsf{io}_P||$ is polynomial in the size of the type system and P (in a classical tree representation of terms), this shows that the procedure would be coNEXP even if the types are represented in *DAG form* (i.e. the representation size of a type is only the number of its different subtypes, which is linear in n for the type of binary trees of height n).

To sum up, our coNEXP-completeness result holds for processes represented in any form (tree or DAG) but only with the type system represented in DAG form.

Construction of the type system The set of atomic types $\mathcal{T}_0$ contains $\tau_{\mathbb{B}}$, a type τ_k for each name k used as an encryption key in the process, τ_{nil} and τ_{new} . The typing function δ is then defined as follows on the atomic data of the process:

- $\delta(a) = \tau_{\mathbb{B}}$ if a is one of the constants $\mathsf{0}, \mathsf{1}$ or a variable that expects a boolean in the description of the process (e.g. b_i, x_i, y_i, ω_i in *CheckSat*). Note that $\tau_{\mathbb{B}}$ is public.
- $\delta(k) = \tau_k$ if k is a name used as an encryption key. Note that the types τ_k are private.
- $\delta(\mathsf{nil}) = \tau_{\mathsf{nil}}$
- $\delta(r) = \tau_{\mathsf{new}}$ if r is declared in the process using a new binder.
- we write τ^p_{BT} the type of binary trees of height p, defined by

$$\tau^0_{\mathsf{BT}} = \tau_{\mathbb{B}} \qquad\qquad \tau^{p+1}_{\mathsf{BT}} = \langle \tau^p_{\mathsf{BT}}, \tau^p_{\mathsf{BT}} \rangle$$

  Then $\delta(x) = \tau^n_{\mathsf{BT}}$ where x is the initial input variable of *Init* and *CheckSat*. We also have $\delta(x) = \delta(y) = \tau^{n-j}_{\mathsf{BT}}$ where x and y are the input variables at the start of E^j_i.

It is then straightforward to verify that $P(\mathsf{0})$ and $P(\mathsf{1})$ are type-compliant w.r.t. this type system. Regarding acyclicity a picture of the dependency graph of $P(t)$ can be found in Figure E.3. Each circle is a node of the process, an arrow is an edge of the graph and dotted arrows materialise a chain of linked nodes of non-constant length. We omitted some nodes in the picture, e.g. the subgraphs of the circuits $[\![\Gamma_i]\!]$ since they are isomorphic to the circuits themselves.

Correctness of the reduction We now prove that the reduction is correct. More precisely we prove that it is even correct for reachability by showing that the following three points are equivalent:

(i) $P(\mathsf{0})$ and $P(\mathsf{1})$ are not trace equivalent

(ii) There exists a trace $P(\mathsf{0}) \overset{\mathsf{tr}}{\Rightarrow} (\mathcal{P}, \Phi)$ such that $\mathsf{senc}(\mathsf{nil}, k^0_{Final})$ is deducible from Φ

(iii) φ is satisfiable

▷ Proof of $\neg(ii) \implies \neg(i)$

Under the assumption $\neg(ii)$, the process $P(t)$ is trace equivalent to the process obtained by replacing the last output of $F(t)$ by the null process in the definition of $P(t)$. This process does not depends of t anymore, hence $P(\mathsf{0})$ and $P(\mathsf{1})$ are also trace equivalent.

▷ Proof of $(ii) \implies (i)$

Let $t : P(\mathsf{0}) \overset{\mathsf{tr}}{\Rightarrow} (\mathcal{P}, \Phi)$ a trace such that there exists a recipe ξ such that $\mathsf{msg}(\xi\Phi)$ and $\xi\Phi{\downarrow} = \mathsf{senc}(\mathsf{nil}, k^0_{Final})$. Without loss of generality we assume this trace of minimal size; in particular no instruction of $F(t)$ has been executed in this trace. Then it suffices to consider

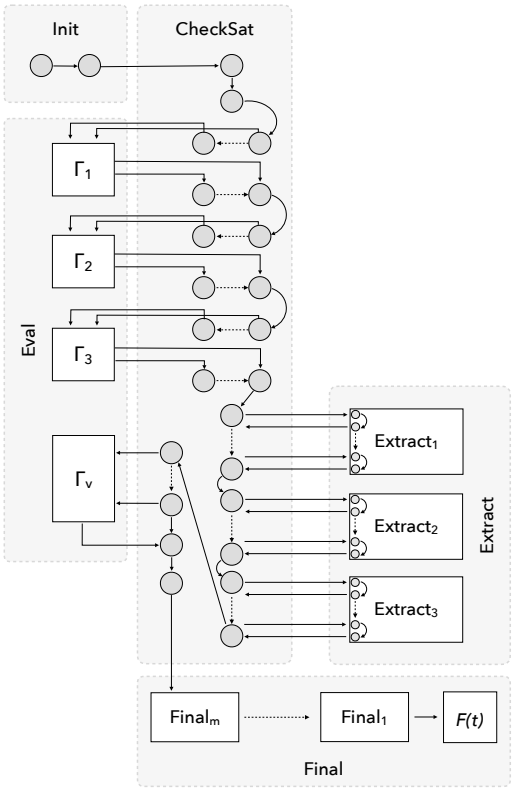


Figure E.3 Dependency graph of $P(t)$.

the trace t' extending t by executing the input and output of $F(t)$:

$$P(0) \overset{\mathsf{tr}}{\Rightarrow} (\mathcal{P}, \Phi) \overset{c(\xi)\overline{c}\langle \mathsf{ax}\rangle}{\Longrightarrow} (\mathcal{P}', \Phi \cup \{\mathsf{ax} \mapsto 0\})$$

The unique trace in $P(1)$ with the same action word is

$$P(1) \overset{\mathsf{tr}}{\Rightarrow} (\mathcal{P}, \Phi) \overset{c(\xi)\overline{c}\langle \mathsf{ax}\rangle}{\Longrightarrow} (\mathcal{P}', \Phi \cup \{\mathsf{ax} \mapsto 1\})$$

whose final frame is not statically equivalent to that of t'.

▷ Proof of $(ii) \implies (iii)$

A quick induction on i shows that for any trace t of $P(t)$ whose final frame is Φ' (where t is an arbitrary term), if the term $\mathsf{senc}(\langle x_1, \ldots, x_i\rangle, k^i_{Final})$ is deducible from Φ' then all terms

$$\mathsf{senc}(\langle b_1, \ldots, b_{m-i}, x_1, \ldots, x_i\rangle, k^m_{Final}) \qquad\qquad b_1, \ldots, b_{m-i} \in \mathbb{B}$$

are also deducible from Φ'. In particular by (*ii*), all terms $\mathsf{senc}(\langle b_1, \ldots, b_m\rangle, k^m_{Final})$, $b_1, \ldots, b_m \in \mathbb{B}$ are deducible from Φ. Since k^m_{Final} is of private type and the only output encrypted with k^m_{Final} is in *CheckSat*, all such messages need have been output by an instance of *CheckSat* during the trace. Therefore we deduce

- x is a binary tree of height at least n (otherwise the process *CheckSat* could never be executed until the end). It contains all positions $p_1, \ldots, p_n$ such that the variable x_i appears in a clause of φ, where $p_1, \ldots, p_n$ is the binary representation of i, and these positions are boolean leaves. Hence x encodes a valuation v.
- For all $i \in [\![0, 2^m - 1]\!]$, the i^{th} clause of φ is valued to $\mathsf{1}$ by v.

All in all v satisfies φ.

▷ Proof of (*iii*) $\Longrightarrow$ (*ii*)

Let v a valuation that satisfies φ and t the complete binary tree of height n whose i^{th} leaf is the valuation $v(x_i)$ of the i^{th} variable of φ. Then we consider the trace consisting of the following actions:

- Execute *Init* with input recipe $\xi = t$ for x. The output $\mathsf{senc}(x, k)$ is referred through the axiom ax.
- Execute the 2^m instances of *CheckSat* each obtained for initial input recipes $\xi_1 = \mathsf{ax}$ and $\xi_2 = \langle p_1, \ldots, p_m, \mathsf{nil}\rangle$ for some booleans $p_1, \ldots, p_m$. In particular, after interacting with *Extract* and *Eval*, the final result of Γ_v is the valuation of the i^{th} clause of φ w.r.t. v, where i is the integer whose binary representation is $p_1, \ldots, p_m$. Since v satisfies all clause of φ by hypothesis, this enables the execution of the final output $\mathsf{senc}(\langle p_1, \ldots, p_m, \mathsf{nil}\rangle, k^m_{Final})$ of this instance of *CheckSat*. We refer to this output as $\mathsf{ax}_{p_1,\ldots,p_m}$.
- For each $i \in [\![1, m]\!]$ in decreasing order, execute all instances of $Final_i$ obtained by inputting $\mathsf{ax}_{0,p_{m-i+2},\ldots,p_m}$, $\mathsf{ax}_{1,p_{m-i+2},\ldots,p_m}$, and outputting back an output referred through the axiom $\mathsf{ax}_{p_{m-i+2},\ldots,p_m}$. Eventually the axiom $\mathsf{ax}_{\varepsilon}$ points to the term $\mathsf{senc}(\mathsf{nil}, k^0_{Final})$.

In particular this gives the expected conclusion.

Efficient verification of observational equivalences of cryptographic processes: theory and practice

Abstract. This thesis studies the analysis of *cryptographic protocols*. They are sequences of instructions permitting to interact with a recipient remotely while protecting the sensitive content of the communication from a potential malicious third party. Classical cases where the confidentiality and the integrity of the communication are critical are, among others, online payments and medical-service booking, or electronic voting.

We study notions of security defined technically by *observational equivalences* (which includes among others *confidentiality*, *anonymity* or *non-traceability*). We designed a program, DeepSec, which, from the description of a protocol for a fixed number of participants, verifies in a fully-automated way whether the protocol offers a security guarantee of this type. We demonstrate the ability of this tool to analyse complex attack scenarios through several examples.

After that, we present an optimisation technique exploiting *symmetries* in security proofs. Typically, participants with similar roles in the protocol often have similar instructions to follow, inducing redundant work during the analysis. On several examples, using this technique allowed to reduce the analysis time of DeepSec by several orders of magnitude.

Finally, we also studied the theoretical aspect the problem in order to determine to which extent DeepSec's algorithm could be improved (can we make the tool faster?) or made more expressive (can we get rid of some of the limitations of the tool?). For that we carried out a complete analysis of the *computational complexity* of DeepSec's algorithm, and integrated it to a detailed survey of the state of the art regarding complexity results in similar contexts. This meticulous survey allowed us to expose subtle variations in how the problem is formalised across the literature—sometimes impacting its complexity. We also include new results and improve some of the surveyed ones, resulting in a clearer understanding of the problem.

Vérification efficace d'équivalences observationnelles de processus cryptographiques : théorie et pratique

Résumé. Cette thèse porte sur l'analyse des *protocoles cryptographiques*. Ce sont des suites d'instructions permettant d'interagir à distance avec un interlocuteur tout en protégeant le contenu sensible de la communication d'une potentielle tierce partie malveillante. Des cas classiques où la confidentialité et l'intégrité de la communication sont critiques sont, parmi d'autres, les paiements et services de santé en ligne, ou le vote électronique.

Nous étudions les notions de sécurité définies techniquement par des *équivalences observationelles* (ce qui inclut entre autre la *confidentialité*, l'*anonymat* ou la *non-traçabilité*). Nous avons conçu un programme, DeepSec, qui, à partir de la description d'un protocole pour un nombre fixé de participants, vérifie de manière entièrement automatisée si le protocole offre une garantie de sécurité donnée de ce type. Nous démontrons ensuite la capacité de cet outil à analyser des scenarios d'attaques complexes à travers plusieurs exemples.

Dans un deuxième temps, nous présentons une technique d'optimisation reposant sur l'exploitation de *symétries* dans les preuves de sécurité. Typiquement, les participants ayant des roles similaires dans le protocole ont souvent les mêmes instructions à suivre, ce qui induit des tâches redondantes lors de l'analyse. Sur plusieurs exemples, l'utilisation de cette technique d'optimisation a permis de réduire le temps d'analyse de DeepSec de plusieurs ordres de magnitude.

Enfin, nous avons également étudié l'aspect théorique du problème afin de déterminer dans quelle mesure l'algorithme de DeepSec pouvait être amélioré (peut-on rendre l'outil plus rapide?) ou rendu plus expressif (peut-on rendre l'outil capable d'analyser plus de protocoles, *i.e.*, nous débarasser de certaines de ses limitations?). Nous avons pour cela fait une analyse de *complexité calculatoire* complète de l'algorithme de DeepSec, et l'avons intégré à une revue détaillée de l'état de l'art des résultats de complexité dans des contextes similaires. Cette revue minutieuse nous a permis de mettre au jour des variations subtiles dans la formalisation du problème à travers la littérature — parfois ayant un impact sur sa complexité. Nous y incluons de nouveaux résultats et améliorons certains ce ceux passés en revue, offrant une compréhension plus claire du problème.