HAL
open science

Exploiting Symmetries When Proving Equivalence Properties for Security Protocols

Vincent Cheval, Steve Kremer, Itsaka Rakotonirina

To cite this version:

Vincent Cheval, Steve Kremer, Itsaka Rakotonirina. Exploiting Symmetries When Proving Equivalence Properties for Security Protocols. CCS'19 - 26th ACM Conference on Computer and Communications Security, Nov 2019, London, United Kingdom. hal-02269043

HAL Id: hal-02269043

https://hal.science/hal-02269043v1

Submitted on 22 Aug 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Exploiting Symmetries When Proving Equivalence Properties for Security Protocols

Vincent Cheval
INRIA Nancy Grand-Est
LORIA

Steve Kremer
INRIA Nancy Grand-Est
LORIA

Itsaka Rakotonirina
INRIA Nancy Grand-Est
LORIA

ABSTRACT

Verification of privacy-type properties for cryptographic protocols in an active adversarial environment, modelled as a behavioural equivalence in concurrent-process calculi, exhibits a high computational complexity. While undecidable in general, for some classes of common cryptographic primitives the problem is coNEXP-complete when the number of honest participants is bounded.

In this paper we develop optimisation techniques for verifying equivalences, exploiting symmetries between the two processes under study. We demonstrate that they provide a significant (several orders of magnitude) speed-up in practice, thus increasing the size of the protocols that can be analysed fully automatically.

CCS CONCEPTS

• **Security and privacy** → **Formal security models**; **Logic and verification**.

KEYWORDS

Security Protocols; Partial-Order Reductions; Symmetry

ACM Reference Format:
Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. 2019. Exploiting Symmetries When Proving Equivalence Properties for Security Protocols. In *2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19), November 11–15, 2019, London, United Kingdom.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3319535.3354260

1 INTRODUCTION

Security protocols are distributed programs transmitting data between several parties. The underlying messages may be sensitive—for economical, political, or privacy reasons—and communications are usually performed through an untrusted network such as the Internet. Therefore, such protocols need to guarantee strong security requirements in an *active* adversarial setting, i.e., when considering an adversary that has complete control over the communication network. Formal, symbolic methods, rooted in the seminal work of Dolev and Yao [26], have been successful in analysing complex protocols, including for instance the recent TLS 1.3 proposal [11, 25] and the upcoming 5G standard [8].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CCS '19, November 11–15, 2019, London, United Kingdom
© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6747-9/19/11…$15.00
https://doi.org/10.1145/3319535.3354260

While some security properties can be formalised as reachability statements, privacy related properties are generally defined as the indistinguishability of two situations where the value of a private attribute differs. This is why privacy-type properties such as anonymity, (strong flavors of) secrecy, unlinkability, or privacy in e-voting are often modelled as behavioural equivalences in concurrent process calculi, such as the applied pi-calculus [1]. The problem of verifying such equivalences is undecidable in the full, Turing-complete, calculus. Still, decidability results and fully automated analysers exist when the number of protocol sessions is bounded.

Unfortunately, recent results [16] show that the problem has a high computational worst-case complexity (coNEXP-complete). Yet, other results show that the problem is exponentially simpler (coNP-complete) for a class of practical scenarios (*determinate processes*) [21]. This gap is all the more striking in practice as, for determinate processes, the verification time can effectively be reduced by several orders of magnitude using *partial-order reductions* [6, 17]. This highlights the gap between the general, pessimistic complexity bound and what can be achieved by exploiting specificities of given instances. In practice, the processes that are analysed show a great amount of symmetries as they often consist of several copies (sessions) of the same protocol executed in parallel. Exploiting this helps factoring out large, redundant parts of equivalence proofs, and making theoretically hard verification feasible in practice.

Contributions

We present optimisations for the verification of trace equivalence in the applied pi-calculus. For that we exploit the symmetries of the two processes to be shown equivalent. More specifically, our contributions are as follows.

(1) We introduce *equivalence by session*, a new process equivalence that implies the classical trace equivalence. Intuitively, it is a refinement of trace equivalence designed for two processes sharing a similar structure, making verification easier.
(2) We show how partial-order reductions presented in [6] for determinate processes, can be used for proving equivalence by session for *any* processes.
(3) We give a group-theoretic characterisation of internal process redundancy, inspired by classical formalisations of symmetries in model checking [28], and use it to reduce further the complexity of deciding equivalence by session.
(4) We design a symbolic version of the above equivalence and optimisations, based on the constraint solving techniques of the DeepSec prover [17], a state-of-the-art tool for verifying equivalence properties in security protocols. This allowed us to implement our techniques in DeepSec and evaluate the gain in verification time induced by our optimisations.

Note that, while we designed equivalence by session as an efficient proof technique for trace equivalence it is also of independent interest: to some extent, equivalence by session models attackers that can distinguish different sessions of a same protocol. This may be considered realistic when servers allocate a distinct ephemeral port for each session; in other contexts, e.g. RFID communication this may however be too strong. When equivalence by session is used as a proof technique for trace equivalence, false attacks are possible, as it is a sound, but not complete, refinement. However, on the existing protocols we experimented on, each time equivalence by session was violated, trace equivalence was violated as well.

Our prototype is able to successfully analyse various security protocols that are currently out of scope—in terms of expressivity or exceeding a 12h timeout—of similar state-of-the-art analysers. We observe improvements of several orders of magnitude in terms of efficiency, compared to the original version of DeepSec. Among the case studies that we consider are

- the *Basic Acces Control (BAC)* protocol [29] implemented in European e-passports. In previous work, verification was limited to merely 2 sessions, while in this paper we scale up to 5 sessions.
- the *Helios* e-voting protocol [2]. Automated analyses of this protocol exist when no revote is allowed, or is limited to one revote from a honest voter [3, 16]. In this paper, we analyse several models covering revote scenarios for 7 emitted honest ballots.

The main technical proofs of our results can be found in appendix. See the companion technical report [18] for full proofs and more general versions of the results presented in this paper.

Related work

Partial-order reduction (por) techniques for the verification of cryptographic protocols were first introduced by Clark et al. [20]: 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. [30] proposed por techniques that are suitable for symbolic methods based on constraint solving. However, both the techniques of Clark et al. [20] and Mödersheim et al. [30] are only correct for trace properties.

Partial order reduction techniques for equivalence properties were only introduced more recently by Baelde et al. [5, 6]: implementing these techniques in the APTE tool resulted in spectacular speed-ups. Other state-of-the-art tools, AKISS [14] and DeepSec [16], 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 above. In recent work, Baelde et al. [7] 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, significantly improve efficiency.

There exist other tools for the verification of equivalence properties in the case of a bounded number of sessions. The SAT-Equiv tool [22] is extremely efficient, but its scope is more narrow: it does not support user-defined equational theories and is restricted to determinate processes. As shown in [16], the AKISS [14] and SPEC tool [32] were already less efficient (by orders of magnitude) than the DeepSec tool before our current work. We also mention the less recent S^3A tool [27] that verifies testing equivalence in the SPI calculus and integrates some symmetry (but no partial-order) reductions [19]. The tool however only supports a fixed equational theory and no else branches. We are not aware of a publicly available implementation.

Finally, our approach can also be compared to tools for an unbounded number of sessions. The ProVerif [13], Tamarin [9] and Maude-NPA [31] tools all show a process equivalence that is more fine-grained than trace-equivalence. The resulting equivalence is often referred to as *diff-equivalence* in that it requires that equivalent processes follow the same execution flow and only differ on the data. As a result these techniques may fail to prove equivalence of processes that are trace equivalent. Our approach goes in the same direction but 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 Section 3.1. Besides, 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.

2 MODEL

We first present our model for formalising privacy-type properties of security protocols. They are represented by trace equivalence of processes in the applied-pi calculus [1].

2.1 Messages and cryptography

To analyse protocols, we rely on symbolic models rooted in the seminal work of Dolev and Yao [26]. Cryptographic operations are modelled by a finite *signature*, i.e., a set of function symbols with their arity $\mathcal{F} = \{\mathsf{f}/n, \mathsf{g}/m, \ldots\}$. Atomic data such as nonces, random numbers, or cryptographic keys are represented by an infinite set of *names*

$$\mathcal{N} = \{a, b, k, \ldots\} = \mathcal{N}_{pub} \cup \mathcal{N}_{priv}$$

partitioned into *public* and *private* names. We also consider an infinite set of variables $\mathcal{X} = \{x, y, z, \ldots\}$. Protocol messages are then modelled as terms obtained by application of function symbols to names, variables or other terms. If $A \subseteq \mathcal{N} \cup \mathcal{X}$, $\mathcal{T}(\mathcal{F}, A)$ refers to the set of terms built from atoms in A.

Example 2.1. The following signature $\mathcal{F}$ models classical primitives: pair and projections, public-key generation, randomised asymmetric encryption and decryption, and digital signature.

$$\langle\cdot,\cdot\rangle/2 \qquad \mathsf{proj}_1/1 \qquad \mathsf{proj}_2/1 \qquad \mathsf{pk}/1$$
$$\mathsf{aenc}/3 \qquad \mathsf{adec}/2 \qquad \mathsf{sign}/2 \qquad \mathsf{verify}/2$$

For example, let $m \in \mathcal{N}_{pub}$ model some public value, and $k, r \in \mathcal{N}_{priv}$ a private key and a random value, respectively. The term $c = \mathsf{aenc}(m, \mathsf{pk}(k), r)$ represents a ciphertext obtained by encrypting m with the public key $\mathsf{pk}(k)$ and randomness r. Its decryption is represented by the term $\mathsf{adec}(c, k)$. △

An *equational theory* is a binary relation E on terms. It is extended to an equivalence relation $=_E$ that is the closure of E by

reflexivity, symmetry, transitivity, substitution and applications of function symbols. All the optimisations we present in this paper are sound for arbitrary equational theories although, obviously, the implementation in DeepSec naturally inherits the restrictions of the tool (limited to destructor subterm convergent rewriting systems).

Example 2.2. The following equations characterise the behaviour of the primitives introduced in Example 2.1:

$$\mathsf{proj}_1(\langle x, y\rangle) =_E x \qquad \mathsf{adec}(\mathsf{aenc}(x, \mathsf{pk}(y), z), y) =_E x$$
$$\mathsf{proj}_2(\langle x, y\rangle) =_E y \qquad \mathsf{verify}(\mathsf{sign}(x, y), \mathsf{pk}(y)) =_E x$$

That is, a message encrypted with a public key $\mathsf{pk}(k)$ can be recovered with the corresponding private key k, and a message signed with k can be recovered with $\mathsf{pk}(k)$. Using the notations of Example 2.1, we can derive from these equations that $\mathsf{adec}(c, k) =_E m$. One-wayness of h is modelled by an absence of equations. △

A *substitution* σ is a mapping from variables to terms, homomorphically extended to a function from terms to terms. We use the classical postfix notation $t\sigma$ instead of $\sigma(t)$, and the set notation $\sigma = \{x_1 \mapsto x_1\sigma, \ldots, x_n \mapsto x_n\sigma\}$. In particular we may use operators such as $\subseteq$ or $\cap$ with substitutions.

2.2 Protocols as processes

Syntax. Protocols are modelled as concurrent processes exchanging messages (i.e. terms). We define the syntax of *plain processes* by the following grammar:

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

where u, v are terms, $x \in \mathcal{X}$, and $c \in Ch$ where Ch denotes a set of *channels*. We assume a partition $Ch = Ch_{pub} \cup Ch_{priv}$ of channels into public and private channels: while public channels are under the control of the adversary, private channels allow confidential, internal communications. The 0 process is the terminal process which does nothing, the operator $P \mid Q$ executes P and Q concurrently, $\overline{c}\langle u\rangle$ sends a message u on channel c, and $c(x)$ receives a message (and binds it to the variable x).

We highlight two restrictions compared to the calculus of [1]: we only consider a bounded number of protocol sessions (i.e. there is no operator for unbounded parallel replication) and channels are modelled by a separate datatype (i.e. they are never used as parts of messages). The first restriction is necessary for decidability [14, 16, 32] but still allows to detect many flaws since attacks tend to require a rather small number of sessions. Our optimisations also rely on an invariant that private channels remain unknown to the adversary, hence the restriction to disallow channel names in messages.

Example 2.3. We describe a minimal toy protocol for electronic voting that will serve as a running example throughout the paper. We build upon the signature and equational theory introduced in Examples 2.1 and 2.2. The system operates as follows:

(1) Each voter generates a private signature key and sends the corresping public key to the voting authority using an authenticated channel;
(2) The voter encrypts a pair $\langle vote,\ public\ key\rangle$ (using the public key of the voting authority), signs it, and casts it;
(3) After verifying the signatures and decrypting the messages, the voting authority shuffles the votes using a mixnet, and outputs them in clear.

The first messages of the protocol (communication of the signature key, casting one vote, and verification of the signature) can be modelled as follows in the applied pi-calculus:

$$\begin{array}{l} \mathit{Voter}(c, vote, s, r) = \\ \quad \overline{\mathsf{auth}}\langle \mathsf{pk}(s_V)\rangle. \\ \quad \overline{c}\langle \mathsf{pk}(s_V)\rangle. \\ \quad \overline{c}\langle \mathsf{sign}(v, s_V)\rangle.\ 0 \quad \text{with } v = \mathsf{aenc}(\langle vote, \mathsf{pk}(s_V)\rangle, \mathsf{pk}(s_A), r) \end{array}$$

$$\begin{array}{l} \mathit{Authority} = \\ \quad \mathsf{auth}(pk_{sign1}).\ c(x_1). \\ \quad \mathsf{if}\ \mathsf{proj}_2(\mathsf{adec}(\mathsf{verify}(x_1, pk_{sign1}), s_A)) = pk_{sign1}\ \mathsf{then} \\ \quad \mathsf{auth}(pk_{sign2}).\ c(x_2). \\ \quad \mathsf{if}\ \mathsf{proj}_2(\mathsf{adec}(\mathsf{verify}(x_1, pk_{sign2}), s_A)) = pk_{sign2}\ \mathsf{then} \\ \quad (\cdots) \end{array}$$

where $\mathsf{auth} \in Ch_{priv}$, $s_V \in \mathcal{N}_{priv}$ is the generated signature key, and $s_A \in \mathcal{N}_{priv}$ the authority's private key. In the voter process, the authenticated transfer of the signature key is modelled by two consecutive outputs of $\mathsf{pk}(s_V)$: a first output on a private channel auth and a second output on a public channel c. The private channel ensures that the adversary cannot block or modify the content of the message; the public channel enables the adversary to eavesdrop the message. In the authority process, two signed votes are verified. The overall system with two voters is modelled by

$$S(v_1, v_2) = \mathit{Voter}(c_1, v_1, s_1, r_1) \mid \mathit{Voter}(c_2, v_2, s_2, r_2) \mid \mathit{Authority}$$

where $c_1, c_2 \in Ch_{pub}$, $s_1, s_2, r_1, r_2 \in \mathcal{N}_{priv}$. △

Semantics. The behaviour of protocols is defined by an operational semantics on processes. Its first ingredients are simplifying rules to normalise processes from non-observable, deterministic actions (Figure 1).

$$P \mid 0 \rightsquigarrow P \qquad 0 \mid P \rightsquigarrow P \qquad (P \mid Q) \mid R \rightsquigarrow P \mid (Q \mid R)$$

$$\left.\begin{array}{l} P \mid Q \rightsquigarrow P' \mid Q \\ Q \mid P \rightsquigarrow Q \mid P' \end{array}\right\} \text{if } P \rightsquigarrow P'$$

$$\mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q \rightsquigarrow \begin{cases} P & \text{if } u =_E v \\ Q & \text{otherwise} \end{cases}$$

Figure 1: Simplification rules for plain processes

These simplifying rules get rid of 0 processes, and evaluate conditionals at toplevel. We say that a process on which no more rule applies is in $\rightsquigarrow$-normal form. This rewriting relation being convergent, we will denote by $P\wr$ the unique $\rightsquigarrow$-normal form of P.

The operational semantics then operates on *extended processes*. An extended process is a pair $(\mathcal{P}, \Phi)$, where $\mathcal{P}$ is a multiset of plain processes (in $\rightsquigarrow$-normal form) and Φ is a substitution, called the *frame*. Intuitively, $\mathcal{P}$ is the multiset of processes that are ready to be executed in parallel, and Φ is used to record outputs on public channels. The domain of the substitution Φ is a subset of a set $\mathcal{A}\mathcal{X}$

of *axioms*, disjoint from $\mathcal{X}$: they record the raw observations of the attacker, that is, they are the axioms in intruder deduction proofs. The semantics (Figure 2) takes the form of a labelled transition relation $\xrightarrow{\alpha}$ between extended processes, where α is called an *action* and indicates what kind of transition is performed.

$$(\textsc{Out}) \quad (\{\!\{\overline{c}\langle u\rangle.P\}\!\} \cup \mathcal{P}, \Phi) \xrightarrow{\overline{c}\langle \mathsf{ax}\rangle} (\{\!\{P\}\!\} \cup \mathcal{P}, \Phi \cup \{\mathsf{ax} \mapsto u\}) \quad c \in Ch_{pub}, \mathsf{ax} \in \mathcal{AX} \setminus dom(\Phi)$$

$$(\textsc{In}) \quad (\{\!\{c(x).P\}\!\} \cup \mathcal{P}, \Phi) \xrightarrow{c(\xi)} (\{\!\{P[x \mapsto \xi\Phi]\}\!\} \cup \mathcal{P}, \Phi) \quad c \in Ch_{pub}, \xi \in \mathcal{T}(\mathcal{F}, \mathcal{N}_{pub} \cup dom(\Phi))$$

$$(\textsc{Comm}) \quad (\{\!\{\overline{c}\langle u\rangle.P, c(x).Q\}\!\} \cup \mathcal{P}, \Phi) \xrightarrow{\tau} (\{\!\{P, Q[x \mapsto u]\}\!\} \cup \mathcal{P}, \Phi) \quad c \in Ch_{priv}$$

$$(\textsc{Par}) \quad (\{\!\{P_1 \mid \ldots \mid P_n\}\!\} \cup \mathcal{P}, \Phi) \xrightarrow{\tau} (\{\!\{P_i\}\!\}_{i=1}^{n} \cup \mathcal{P}, \Phi)$$

Figure 2: Operational semantics of the applied pi-calculus

The output rule (Out) models that outputs on a public channel are added to the attacker knowledge, i.e., stored in Φ in a fresh axiom. The axioms thus provide handles for the attacker to refer to these outputs. The input rule (In) reads a term ξ, called a *recipe* provided by the attacker, on a public channel. This term ξ can be effectively constructed by the attacker as it is built over public names and elements of $dom(\Phi)$, i.e. previous outputs. The resulting term is then bound to the input variable x. Rule (Comm) models internal communication on a private channel and rule (Par) adds processes in parallel to the multiset of active processes. These last two actions are internal actions (label τ), unobservable by the attacker.

Traces. A *trace* of an extended process A is a sequence of reduction steps starting from the extended process A, written

$$t : A \xrightarrow{\alpha_1} A_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} A_n\,.$$

When the intermediate processes are not relevant we rather write

$$t : A \xRightarrow{\alpha_1 \cdots \alpha_n} A_n\,.$$

We define $tr(t)$ to be the word of actions $\alpha_1 \cdots \alpha_n$ (including τ's), and $\Phi(t)$ to be the frame of A_n.

The set of the traces of A is written $\mathbb{T}(A)$, and the notation is extended to plain processes by writing $\mathbb{T}(P)$ for $\mathbb{T}(\{\!\{P\}\!\}, \emptyset)$. Indistinguishability against active adversaries will be modelled as relations between such sets of traces.

Example 2.4. Consider again the voting system partially represented in Example 2.3. Let us write R the non-modelled part of the process *Authority*. Then the system S has the following trace modelling the first voter casting her vote:

$$\begin{aligned}(\{\!\{S\}\!\}, \emptyset) &\xrightarrow{\tau} (\{\!\{Voter(c_1, v_1, s_1, r_1), Voter(c_2, v_2, s_2, r_2), Authority\}\!\}, \emptyset)\\ &\xRightarrow{\tau\, \overline{c_1}\langle \mathsf{ax}_1\rangle \overline{c_1}\langle \mathsf{ax}_2\rangle} (\{\!\{0, Voter(c_2, v_2, s_2, r_2), A'\}\!\}, \Phi)\\ &\xrightarrow{c(\mathsf{ax}_2)} (\{\!\{0, Voter(c_2, v_2, s_2, r_2), A''\}\!\}, \Phi)\end{aligned}$$

where

$$\begin{aligned}A' &= c(x_1).\\ &\quad \text{if } \mathsf{proj}_2(\mathsf{adec}(\mathsf{verify}(x_1, \mathsf{pk}(s_1)), s_A)) = \mathsf{pk}(s_1) \text{ then } A''\\ A'' &= \mathsf{auth}(pk_{sign2}).\ c(x_2).\\ &\quad \text{if } \mathsf{proj}_2(\mathsf{adec}(\mathsf{verify}(x_1, pk_{sign2}), s_A)) = pk_{sign2} \text{ then } R\\ \Phi &= \{\mathsf{ax}_1 \mapsto \mathsf{pk}(s_1),\\ &\quad \mathsf{ax}_2 \mapsto \mathsf{sign}(\mathsf{aenc}(\langle v_1, \mathsf{pk}(s_1)\rangle, \mathsf{pk}(s_A), r_1), \mathsf{pk}(s_1))\}\end{aligned}$$

△

2.3 Security properties

Many security properties can be expressed in terms of indistinguishability (from the attacker's viewpoint). The preservation of anonymity during a protocol execution can for example be modelled as the indistinguishability of two instances of the protocol with different participants. Strong flavors of secrecy can also be expressed: after interacting with the protocol, the attacker is still unable to distinguish between a secret used during the protocol and a fresh random nonce. Our case studies also include such modellings of unlinkability or vote privacy.

Static equivalence. The ability to distinguish or not between two situations lies on the attacker's observations, i.e. the frame. Indistinguishability of two frames is captured by the notion of *static equivalence*. Intuitively, we say that two frames are statically equivalent if the attacker cannot craft an equality test that holds in one frame and not in the other.

Definition 2.1. Two frames Φ_1 and Φ_2 are statically equivalent, written $\Phi_1 \sim \Phi_2$ when $dom(\Phi_1) = dom(\Phi_2)$ and, for any recipes $\xi_1, \xi_2 \in \mathcal{T}(\mathcal{F}, \mathcal{N}_{pub} \cup dom(\Phi_1))$,

$$\xi_1\Phi_1 =_E \xi_2\Phi_1 \quad \Leftrightarrow \quad \xi_1\Phi_2 =_E \xi_2\Phi_2$$

We lift static equivalence to traces and write $t_0 \sim t_1$ when $\Phi(t_0) \sim \Phi(t_1)$ and $\mathsf{tr}_0 = \mathsf{tr}_1$, where tr_i is obtained by removing τ's from $tr(t_i)$. Removing τ actions reflects that these actions are unobservable by the attacker.

Trace equivalence. While static equivalence models the (passive) indistinguishability of two sequences of observations, *trace equivalence* captures the indistinguishability of two processes P and Q in the presence of an active attacker. Intuitively, we require that any sequence of visible actions executable on P is also executable on Q and yields indistinguishable outputs, i.e., statically equivalent frames.

Definition 2.2. Let P, Q be plain processes in $\rightsquigarrow$-normal form. P is *trace included* in Q, written $P \sqsubseteq_{tr} Q$, when

$$\forall t \in \mathbb{T}(P),\ \exists t' \in \mathbb{T}(Q),\ t \sim t'\,.$$

We say that P and Q are *trace equivalent*, written $P \approx_{tr} Q$, when $P \sqsubseteq_{tr} Q$ and $Q \sqsubseteq_{tr} P$.

Example 2.5. A statement of vote privacy of the voting system modelled in Example 2.3 would be

$$S(vote_1, vote_2) \approx_{tr} S(vote_2, vote_1)$$

where $vote_1, vote_2 \in \mathcal{N}_{pub}$ model two potentiel different votes. For a shorter, more concrete example, we let $s \in \mathcal{N}_{priv}$ and define

$$V(R) = c(x).\ \text{if } \mathsf{proj}_2(\mathsf{verify}(x, \mathsf{pk}(s))) = \mathsf{pk}(s) \text{ then } R.$$

This is similar to the verification performed by the authority in the running example, without encryption. The secrecy of s, i.e. the inability of the attacker to forge signatures, can be stated by

$$V(\overline{c}\langle \mathsf{oops}\rangle) \approx_{tr} V(0)\,.$$

As expected, this statement does not hold anymore if s is revealed,

$$V(\overline{c}\langle \mathsf{oops}\rangle) \mid \overline{c}\langle s\rangle \not\approx_{tr} V(0) \mid \overline{c}\langle s\rangle$$

where $c \in Ch_{pub}$. A possible witness of non-equivalence is the following trace in $A = (\{\!\{V(\overline{c}\langle \mathsf{oops}\rangle) \mid \overline{c}\langle s\rangle\}\!\}, \emptyset)$, where $n \in \mathcal{N}_{pub}$:

$$\begin{aligned} t_{left} : A &\xRightarrow{\overline{c}\langle \mathsf{ax}_1\rangle} (\{\!\{V(\overline{c}\langle \mathsf{oops}\rangle)\}\!\}, \{\mathsf{ax}_1 \mapsto s\}) \\ &\xrightarrow{c(\mathsf{sign}(\langle n, \mathsf{pk}(\mathsf{ax}_1)\rangle, \mathsf{ax}_1))} (\{\!\{\overline{c}\langle \mathsf{oops}\rangle\}\!\}, \{\mathsf{ax}_1 \mapsto s\}) \\ &\xrightarrow{\overline{c}\langle \mathsf{ax}_2\rangle} (\{\!\{0\}\!\}, \{\mathsf{ax}_1 \mapsto s, \mathsf{ax}_2 \mapsto \mathsf{oops}\}) \qquad \triangle \end{aligned}$$

3 OPTIMISING VERIFICATION

The problem of verifying trace equivalence in the presented model is coNEXP-complete for equational theories represented as subterm convergent destructor rewrite systems [16]. Despite this high theoretical complexity, automated analysers can take advantage of the specificities of practical instances. One notable example is the class of *determinate* processes that encompasses many practical scenarios and has received quite some attention [6, 14, 17, 21]. It allows for partial-order reductions [6], speeding up the verification time by several orders of magnitude. Our approach, similar in spirit but applicable in a more general setting, consists in guiding the decision procedure with the structural similarities of the two processes that we aim to show equivalent.

3.1 Equivalence by session

We introduce a new equivalence relation, *equivalence by session*: the main idea is that, when proving the equivalence of P and Q, every action of a given parallel subprocess of P should be matched by the actions of a same subprocess in Q. This is indeed often the case in protocol analysis where a given session (the execution of an instance of a protocol role) on one side is matched by a session on the other side. By requiring to match sessions rather than individual actions, this yields a more fine-grained equivalence and effectively reduces the combinatorial explosion. Moreover, thanks to the optimisations that exploit the structural properties of equivalence by session (presented in the following sections), we obtain significant speed-ups during the verification of case studies that are neither determinate nor in scope of the (even more fined-grained) diff-equivalence of ProVerif and Tamarin.

Twin-processes. To formalise session matchings we use a notion of *twin-process*, that are pairs of matched processes that have the same action at toplevel, called their *skeleton*.

Definition 3.1. A twin-process is a pair of plain processes in $\rightsquigarrow$-normal form (P, Q) such that $\mathsf{skel}(P) = \mathsf{skel}(Q)$, where

$$\begin{array}{lll} \text{if } c \in Ch_{pub}: & \mathsf{skel}(c(x).Q) = \{\!\{\mathsf{in}_c\}\!\} & \mathsf{skel}(\overline{c}\langle x\rangle.Q) = \{\!\{\mathsf{out}_c\}\!\} \\ \text{if } d \in Ch_{priv}: & \mathsf{skel}(d(x).Q) = \{\!\{\mathsf{in}\}\!\} & \mathsf{skel}(\overline{d}\langle x\rangle.Q) = \{\!\{\mathsf{out}\}\!\} \end{array}$$

$$\mathsf{skel}(P_1 \mid \cdots \mid P_n) = \mathsf{skel}(P_1) \cup \ldots \cup \mathsf{skel}(P_n)$$

An *extended twin-process* $A^2 = (\mathcal{P}^2, \Phi_0, \Phi_1)$ is then a triple where $\mathcal{P}^2$ is a multiset of twin-processes and Φ_0, Φ_1 are frames. This thus models two extended processes with identical skeletons, matched together. We retrieve the original extended processes by projection,

$$\mathsf{fst}(A^2) = (\{\!\{P_0 \mid (P_0, P_1) \in \mathcal{P}^2\}\!\}, \Phi_0)$$
$$\mathsf{snd}(A^2) = (\{\!\{P_1 \mid (P_0, P_1) \in \mathcal{P}^2\}\!\}, \Phi_1)$$

The semantics of twin-processes is defined in Figure 3 and mostly requires that the two projections follow the same reduction steps in the single-process semantics. The rule (Par) is however replaced by a rule that allows to match each parallel subprocess from the left with a parallel process from the right. We underline that, by definition of twin-processes, a transition $A^2 \xrightarrow{\alpha}_s (\mathcal{P}^2, \Phi)$ is possible only if for all $(P, Q) \in \mathcal{P}^2$, it holds that $\mathsf{skel}(P) = \mathsf{skel}(Q)$.

Similarly to extended processes, we use $\mathbb{T}(A^2)$ to denote the set of reduction steps from an extended twin-process A^2. Besides if

$$t^2 : A^2 \xrightarrow{\alpha_1}_s A_1^2 \cdots \xrightarrow{\alpha_n}_s A_n^2 \quad \in \mathbb{T}(A_1^2)\,,$$

we also lift the projection functions by writing

$$\mathsf{fst}(t^2) : \mathsf{fst}(A^2) \xrightarrow{\alpha_1}_s \mathsf{fst}(A_1^2) \cdots \xrightarrow{\alpha_n}_s \mathsf{fst}(A_{n+1}^2)$$

and similarly for $\mathsf{snd}(t^2)$. Note that $\mathsf{fst}(t^2) \in \mathbb{T}(\mathsf{fst}(A^2))$.

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 formalised by considering traces of the twin-process (P, Q). Formally speaking, given two plain processes P and Q in $\rightsquigarrow$-normal form having the same skeleton, we write $P \sqsubseteq_s Q$ when

$$\forall t \in \mathbb{T}(P),\ \exists t^2 \in \mathbb{T}(P, Q),\ t = \mathsf{fst}(t^2) \sim \mathsf{snd}(t^2)\,.$$

We say that P and Q are *equivalent by session*, referred as $P \approx_s Q$, when $P \sqsubseteq_s Q$ and $Q \sqsubseteq_s P$.

While equivalence by session has been designed 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 which originate 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 when considering 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,

$$P(c) \mid P(c) \approx_s Q(c) \mid Q(c)$$

could be encoded by relying on dynamically-generated private channels that are revealed to the attacker. This can be expressed in the original syntax of the applied pi-calclulus [1] as:

$$P_{fresh} \mid P_{fresh} \approx_{tr} Q_{fresh} \mid Q_{fresh} \quad \text{where } P_{fresh} = \mathsf{new}\, e.\, \overline{c}\langle e\rangle.\, P(e)$$
$$\text{and } Q_{fresh} = \mathsf{new}\, e.\, \overline{c}\langle e\rangle.\, Q(e)$$

Such encodings however break determinacy and are therefore incompatible with the partial-order reductions of [6]. Our dedicated

$$\frac{(\{\!\{P_i\}\!\}, \Phi_i) \xrightarrow{\alpha} (\{\!\{P'_i\}\!\}, \Phi'_i) \text{ by rule (In) or (Out) for all } i \in \{0,1\}}{(\{\!\{(P_0, P_1)\}\!\} \cup \mathcal{P}^2, \Phi_0, \Phi_1) \xrightarrow{\alpha}_s (\{\!\{(P'_0, P'_1)\}\!\} \cup \mathcal{P}^2, \Phi'_0, \Phi'_1)} \quad \text{(IO)}$$

$$\frac{(\{\!\{P_i, Q_i\}\!\}, \Phi_i) \xrightarrow{\tau} (\{\!\{P'_i, Q'_i\}\!\}, \Phi_i) \text{ by rule (Comm) for all } i \in \{0,1\}}{(\{\!\{(P_0, P_1), (Q_0, Q_1)\}\!\} \cup \mathcal{P}^2, \Phi_0, \Phi_1) \xrightarrow{\tau}_s (\{\!\{(P'_0, P'_1), (Q'_0, Q'_1)\}\!\} \cup \mathcal{P}^2, \Phi_0, \Phi_1)} \quad \text{(Comm)}$$

$$\frac{\pi \text{ permutation of } [\![1, n]\!]}{(\{\!\{(P_1 \mid \cdots \mid P_n,\ Q_1 \mid \cdots \mid Q_n)\}\!\} \cup \mathcal{P}^2, \Phi_0, \Phi_1) \xrightarrow{\tau}_s (\{\!\{(P_i, Q_{\pi(i)})\}\!\}_{i=1}^n \cup \mathcal{P}^2, \Phi_0, \Phi_1)} \quad \text{(Match)}$$

Figure 3: Semantics on twin-processes

equivalence offers similar-in-spirit optimisations that are applicable on all processes . In this paper we mostly focus on the use of equivalence by session as a heuristic for trace equivalence.

3.2 Comparison to other equivalences

Relation to trace equivalence. We first show that equivalence by session is a sound refinement of trace equivalence.

Proposition 3.2. *If $P \approx_s Q$ then $P \approx_{tr} Q$.*

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 $n \in \mathcal{N}_{pub}$,

$$P = \overline{c}\langle n\rangle.\, \overline{c}\langle n\rangle \qquad\qquad Q = \overline{c}\langle n\rangle \mid \overline{c}\langle n\rangle$$

We call *false attacks* traces witnessing a violation of equivalence by session, but that can still be matched trace-equivalence-wise. 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: as previously explained, 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 the two processes

$$P = \overline{s}\langle n\rangle.\, \overline{a}\langle n\rangle \mid s(x).\, \overline{b}\langle n\rangle \qquad Q = \overline{s}\langle n\rangle.\, \overline{b}\langle n\rangle \mid s(x).\, \overline{a}\langle n\rangle$$

with $a, b \in Ch_{pub}$ and $s \in Ch_{priv}$. These two processes first synchronise on a private channel s by the means of 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). The consequence is that any trace executing an output on either a or b is a false attack.

Let us finally mention that false attacks cannot happen for determinate processes, i.e. the class of processes for which the partial-order reductions of [6] were designed. A plain process P is determinate if it does not contain private channels and,

$$\forall P \overset{\mathsf{tr}}{\Rightarrow} (\{\!\{P_1, \ldots, P_n\}\!\}, \Phi),\ \forall i \neq j,\ \mathsf{skel}(P_i) \neq \mathsf{skel}(P_j)\,.$$

Proposition 3.3. *If P, Q are determinate plain processes such that $P \approx_{tr} Q$ then $P \approx_s Q$.*

The core argument is that determinacy ensures the uniqueness of potential session matchings; that is, there is always at most one permutation that can be chosen when applying the rule (Match) to a pair of determinate processes. We detailed the proof of this proposition in Appendix B: thanks to the structural requirements imposed by skeletons, we even prove that trace equivalence ($\approx_{tr}$) and inclusion by session ($\sqsubseteq_s$) coincide for determinate processes.

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 [12] conditional branchings were also required to result into the same control-flow. This condition has however been relaxed within [15]: the resulting diff-equivalence can be defined in our formalism as equivalence by session in which the rule (Match) only performs the identity matching. That is, if we write $\mathbb{T}_d(P, Q)$ for the subset of traces of $\mathbb{T}(P, Q)$ where the rule (Match) is replaced by

$$(\{\!\{(P_1 \mid \cdots \mid P_n, Q_1 \mid \cdots \mid Q_n)\}\!\} \cup \mathcal{P}^2, \Phi_0, \Phi_1) \xrightarrow{\tau}_s (\{\!\{(P_i, Q_i)\}\!\}_{i=1}^n \cup \mathcal{P}^2, \Phi_0, \Phi_1)$$

then we define $P \sqsubseteq_d Q$ as the statement

$$\forall t \in \mathbb{T}(P),\ \exists t^2 \in \mathbb{T}_d(P, Q),\ t = \mathsf{fst}(t^2) \sim \mathsf{snd}(t^2)\,.$$

We say that P and Q are diff-equivalent, written $P \approx_d Q$, when $P \sqsubseteq_d Q$ and $Q \sqsubseteq_d P$. By definition $\mathbb{T}_d(P, Q) \subseteq \mathbb{T}(P, Q)$ and diff-equivalence therefore refines equivalence by session. The converse

does not hold in general as witnessed by

$$P = \overline{c}\langle a\rangle \mid \overline{c}\langle b\rangle \qquad Q = \overline{c}\langle b\rangle \mid \overline{c}\langle a\rangle \qquad a, b \in \mathcal{N}_{pub} \text{ distinct}$$

This example is rather extreme as a simple 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 the equivalences we prove on the BAC protocol in Section 7. The underlying reason is that the matchings have to be selected dynamically, that is, different session matchings are needed to match different traces. To sum up:

Proposition 3.4. $\approx_d \subsetneq \approx_s \subsetneq \approx_{tr}$

3.3 Trace refinements

In this section we present an abstract notion of *optimisation*, based on trace refinements. This comes with several properties on how to compose and refine them, providing a unified way of presenting different concrete optimisations for the decision of equivalence by session in the following sections.

Definition 3.5. An *optimisation* is a pair $\mathbb{O} = (\mathbb{O}^\forall, \mathbb{O}^\exists)$ where $\mathbb{O}^\forall$ is 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 studied. That is, we define the equivalence $\approx_{\mathbb{O}} = \sqsubseteq_{\mathbb{O}} \cap \sqsupseteq_{\mathbb{O}}$ where $P \sqsubseteq_{\mathbb{O}} Q$ means

$$\forall t \in \mathbb{T}(P) \cap \mathbb{O}^\forall,\ \exists t^2 \in \mathbb{T}(P, Q) \cap \mathbb{O}^\exists,\ t = \mathsf{fst}(t^2) \sim \mathsf{snd}(t^2)\,.$$

In particular $\approx_{\mathbb{O}_{\mathsf{all}}}$ is the equivalence by session, where $\mathbb{O}_{\mathsf{all}} = (\mathbb{O}^\forall_{\mathsf{all}}, \mathbb{O}^\exists_{\mathsf{all}})$ contains all traces. However, of course, such 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 $\mathbb{O}_\beta = (\mathbb{O}^\forall_\beta, \mathbb{O}^\exists_\beta)$, we say that $\mathbb{O}_\alpha$ is a *correct refinement* of $\mathbb{O}_\beta$ when

$$\mathbb{O}^\forall_\alpha \subseteq \mathbb{O}^\forall_\beta \quad \text{and} \quad \mathbb{O}^\exists_\alpha \subseteq \mathbb{O}^\exists_\beta \quad \text{and} \quad \approx_{\mathbb{O}_\alpha} = \approx_{\mathbb{O}_\beta}\,.$$

Correct refinements contribute to reducing the complexity of deciding equivalence.

Properties. The remainder of this section provides elementary properties useful when constructing, and composing optimisations. First we show that they can be constructed stepwise.

Proposition 3.6 (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 refinements of an optimisation in a modular way, showing separately that universal and existential optimisations are correct.

Proposition 3.7 (combination). *If $(\mathbb{O}^\forall_{opt}, \mathbb{O}^\exists)$ and $(\mathbb{O}^\forall, \mathbb{O}^\exists_{opt})$ are 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 $\approx_{\circ\circ}$ 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 want to implement these optimisations partially, using for example sufficient conditions. The following proposition states that such partial implementations still result into correct refinements.

Proposition 3.8 (partial implementability). *Let us consider $\mathbb{O}^\forall_{opt} \subseteq \mathbb{O}^\forall_{part} \subseteq \mathbb{O}^\forall$ 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 $(\mathbb{O}^\forall, \mathbb{O}^\exists)$.*

This is a straightforward corollary of Proposition 3.7. In the rest of the paper we assume the reader familiar with group theory (group actions, stabilisers), in particular the group of permutations (written in cycle notation). Most of our optimisations are indeed expressed using this terminology.

4 PARTIAL-ORDER REDUCTIONS

In this section we present partial-order reductions for equivalence by session. They are inspired by similar techniques developed for proving trace equivalence of determinate processes [6], although they differ in their technical development to preserve correctness in our more general setting. In particular the optimisations we present account for non determinacy and private channels which will be useful when analysing e-voting protocols.

4.1 Labels and independence

Labels. Partial-order reductions 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:

- Plain processes P are labelled $[\,P\,]^\ell$, with ℓ a word of integers reflecting the position of P within the whole process.
- 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 word of integers ℓ (for inputs and outputs) or a pair of such, written $\ell_1 \mid \ell_2$ (for internal communications).

Labels can be bootstrapped arbitrarily, say, by the empty word ε, and are propagated as follows in the operational semantics. The (Par) rule extends labels:

$$(\{\!\{[\,P_1 \mid \cdots \mid P_n\,]^\ell\}\!\} \cup \mathcal{P}, \Phi) \xrightarrow{[\,\tau\,]^\ell}_s (\{\!\{[\,P_i\,]^{\ell\cdot i}\}\!\}_{i=1}^n \cup \mathcal{P}, \Phi)$$

The rules (In) and (Out) preserve them:

$$(\{\!\{[\,P\,]^\ell\}\!\} \cup \mathcal{P}, \Phi) \xrightarrow{[\,\alpha\,]^\ell}_s (\{\!\{[\,P'\,]^\ell\}\!\} \cup \mathcal{P}, \Phi')$$

and so does (Comm), however producing a double label:

$$(\{\!\{[\,P\,]^\ell, [\,Q\,]^{\ell'}\}\!\} \cup \mathcal{P}, \Phi) \xrightarrow{[\,\tau\,]^{\ell|\ell'}} (\{\!\{[\,P'\,]^\ell, [\,Q'\,]^{\ell'}\}\!\} \cup \mathcal{P}, \Phi)\,.$$

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 materialise control-flow dependencies. Two actions $\alpha = [\,a\,]^L$ and $\alpha' = [\,a'\,]^{L'}$ are *sequentially dependent* if one of the (one or two) words of L, and one of L', are comparable w.r.t. the prefix ordering. Regarding input-output dependencies, we say that α and α' are *data dependent* when $\{a, a'\} = \{\overline{c}\langle \mathsf{ax}\rangle, c(\xi)\}$ and ax appears in ξ. The two notions combine into:

Definition 4.1 (independence). Two actions α and α' are said *independent*, written $\alpha \parallel \alpha'$, when they are sequentially independent and data independent.

There is some redundancy in the trace space in that, intuitively, swapping adjacent, independent actions in a trace has no substantial effect. The rest of this section formalises the intuition that equivalence by session can be studied up to arbitrary permutation of their independent actions (proofs 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

$$\pi.\mathsf{tr} = \alpha_{\pi(1)} \cdots \alpha_{\pi(n)}$$

for the new sequence of actions obtained after permuting the actions of tr with π. This is an action of the group of permutations of $[\![1, n]\!]$ on action words of size n.

We say that π *permutes independent actions* of tr if either $\pi = \mathsf{id}$, or $\pi = \pi_0 \circ (i\ \ i{+}1)$ with $\alpha_i \parallel \alpha_{i+1}$ and π_0 permutes independent actions of $(i\ \ i{+}1).\mathsf{tr}$. An important result is that the actions of such permutations can be meaningfully lifted to entire traces rather than only action words:

Proposition 4.2. *If $t : A \overset{\mathsf{tr}}{\Rightarrow} B$ and π permutes independent actions of* tr*, then there exists a trace $A \overset{\pi.\mathsf{tr}}{\Longrightarrow} B$. This trace is unique if we take labels into account, and will be denoted $\pi.t$.*

This leads to the core property justifying the correctness of our partial-order reductions for equivalence by session. The remaining of Section 4 constructs optimisations leveraging this result.

Proposition 4.3. *Let $\mathbb{O}^{\forall}_1 \subseteq \mathbb{O}^{\forall}_2$ be universal optimisations. We assume that for all $t \in \mathbb{O}^{\forall}_2$, there is π permuting independent actions of t such that $\pi.t \in \mathbb{O}^{\forall}_1$. Then $\mathbb{O}^{\forall}_1$ is a correct refinement of $\mathbb{O}^{\forall}_2$.*

4.2 Compression optimisations

We first present a compression of traces into blocks of actions of a same type (inputs, outputs and parallel, or internal communications). It is presented as a sequence of three successive universal optimisations. Whether a trace t is discarded or not depends on its action word $tr(t)$. The correctness of these refinements essentially follows from Proposition 4.3.

Optim. 1: performing outputs and parallel in priority. We say that an action is *negative* when the underlying transition is derived from the rules (Par) or (Out). This vocabulary is taken from [6], where traces are compressed in a similar way, inspired by focused proof theory. We observe that negative actions are data-independent of all previous actions in the trace: in practice, when executing actions from different parallel subprocesses, it is therefore sound to give priority to negative ones by Proposition 4.3.

To formalise this universal optimisation $\mathbb{O}^{\forall}_{\mathsf{c},1}$ we impose that, if a trace has a non-negative action α^+ before a negative action α^- then there must be an intermediate action β (possibly $\beta = \alpha^+$) on which α^- depends. This indeed makes it incorrect to push α^- before α^+ by adjacent swaps. Formally, for all traces

$$t : A \overset{\alpha_1 \cdots \alpha_n}{\Longrightarrow} B$$

we have that $t \in \mathbb{O}^{\forall}_{\mathsf{c},1}$ if, and only if

$$\forall i < j,\ \text{if} \begin{cases} \alpha_i \text{ non-negative} \\ \alpha_j \text{ negative} \end{cases} \quad \text{then } \exists i \leqslant k < j, \neg(\alpha_k \parallel \alpha_j)$$

Proposition 4.4. *$\mathbb{O}^{\forall}_{\mathsf{c},1}$ is a correct refinement of $\mathbb{O}^{\forall}_{\mathsf{all}}$.*

Optim. 2: performing negative actions deterministically. Data independence between negative actions trivially holds. By Proposition 4.3 again, we can therefore fix some arbitrary priority over the execution of consecutive, concurrent negative actions. This universal optimisation $\mathbb{O}^{\forall}_{\mathsf{c},2}$ is defined as follows. For all traces

$$t : A \overset{\alpha_1 \cdots \alpha_n}{\Longrightarrow} B$$

we have that $t \in \mathbb{O}^{\forall}_{\mathsf{c},2}$ if, and only if for all $i < n$ such that α_i and α_{i+1} are sequentially-independent negative actions, it holds that $\alpha_i \preccurlyeq \alpha_{i+1}$ (where $\preccurlyeq$ is an arbitrary total ordering on actions).

Proposition 4.5. *$\mathbb{O}^{\forall}_{\mathsf{c},1} \cap \mathbb{O}^{\forall}_{\mathsf{c},2}$ is a correct refinement of $\mathbb{O}^{\forall}_{\mathsf{c},1}$.*

Optim. 3: performing chains of inputs in a row. Finally, we observe that inputs are data-independent of all actions appearing later in the trace. In particular, by permuting independent actions, it is always possible to group together consecutive inputs on a same label. It is also possible to move them to the end of traces when they are sequentially independent of all following actions. We formalise this universal optimisation $\mathbb{O}^{\forall}_{\mathsf{c},3}$ as follows. For all traces

$$t : A \overset{\alpha_1 \cdots \alpha_n}{\Longrightarrow} B$$

we have that $t \in \mathbb{O}^{\forall}_{\mathsf{c},3}$ if, and only if

- for all $i, j \in [\![1, n]\!]$, if $\alpha_i = [\,c(\xi)\,]^{\ell}$ and $\alpha_j = [\,d(\zeta)\,]^{\ell}$ with $j \geqslant i{+}2$, then there exists $i < k < j$ such that α_k and α_j are sequentially dependent;
- and for all $i \in [\![1, n-1]\!]$, if α_i and α_{i+1} are input actions on different labels then all α_j, $j \geqslant i$, are input actions.

Combining everything we obtain the *compression optimisation*

$$\mathbb{O}^{\forall}_{\mathsf{c}} = \mathbb{O}^{\forall}_{\mathsf{c},1} \cap \mathbb{O}^{\forall}_{\mathsf{c},2} \cap \mathbb{O}^{\forall}_{\mathsf{c},3}$$

whose correctness is stated below.

Proposition 4.6. *$\mathbb{O}^{\forall}_{\mathsf{c}}$ is a correct refinement of $\mathbb{O}^{\forall}_{\mathsf{c},1} \cap \mathbb{O}^{\forall}_{\mathsf{c},2}$.*

4.3 Reduction optimisations

Blocks. This optimisation relies on the observation that all compressed sequences of actions, i.e. sequences $tr(t)$ with $t \in \mathbb{O}^{\forall}_{\mathsf{c}}$, can be uniquely decomposed into *blocks*

$$tr(t) = b_0^- b_1^+ b_1^- \cdots b_n^+ b_n^- B$$

where each b_i^s is a sequence of actions such that

- no b_i^s is empty except maybe b_0^-;

- all b_i^+ are either a *positive block* (i.e., a sequence of input actions on a same label $\ell(b_i^+)$) or a single internal communication;
- all b_i^- contain only negative actions;
- B is a (possibly-empty) sequence of positive blocks.

We call $b_i = b_i^+ b_i^-$ $(i > 0)$ a *block* of $tr(t)$. Just as actions, adjacent blocks b_i, b_j may be independent (written $b_i \parallel b_j$) in the sense that all actions appearing in b_i are independent of all actions appearing in b_j. The goal of this optimisation is to fix, as much as possible, the order of execution of blocks by exploiting their independence.

Authorisation. Concretely, this optimisation $\mathbb{O}^{\vee}_{\mathsf{c+r}} \subseteq \mathbb{O}^{\vee}_{\mathsf{c}}$ defines a condition on sequences of blocks that only accepts sequences that are lexicographically minimal among all those that can be obtained by independent permutations. We formalise this through a predicate Minimal that characterises, step by step, such traces. Formally, with the notations above,

$$t \in \mathbb{O}^{\vee}_{\mathsf{c+r}} \quad \textit{iff} \quad \bigwedge_{i=1}^{n} \mathsf{Minimal}(b_1 \cdots b_{i-1}, b_i)$$

where the predicate Minimal is inductively defined by

$$\mathsf{Minimal}(b_1 \cdots b_p, b_i) = \begin{cases} \top & \text{if } p = 0 \text{ or } \neg(b_p \parallel b_i) \\ \ell(b_p^+) \prec_{lex} \ell(b_i^+) \wedge \mathsf{Minimal}(b_1 \cdots b_{p-1}, b_i) & \text{otherwise} \end{cases}$$

Proposition 4.7. *$\mathbb{O}^{\vee}_{\mathsf{c+r}}$ is a correct refinement of $\mathbb{O}^{\vee}_{\mathsf{c}}$.*

5 REDUCTIONS BY SYMMETRY

In this section, we show how to exploit process symmetries for equivalence by session, referring again to the framework of Section 3.3. Such symmetries often appear in practice when we verify multiple sessions of a same protocol as it results into parallel copies of identical processes, up to renaming of fresh names. We first provide a group-theoretical characterisation of internal process redundancy, and then design two optimisations.

5.1 Group actions and process redundancy

Let $P = P_1 \mid \cdots \mid P_n$ be a plain process and $\pi \in S_n$, where S_n denotes the symmetric group, i.e., the group of all permutations of $[\![1, n]\!]$. Then we denote by $\vec{P}$ and $\pi.\vec{P}$ the tuples of plain processes

$$\vec{P} = \langle P_1, \ldots, P_n \rangle \qquad \pi.\vec{P} = \langle P_{\pi(1)}, \ldots, P_{\pi(n)} \rangle .$$

Suppose that $\equiv$ is an equivalence relation on tuples of processes that is stable by application of permutations, i.e. $\vec{P} \equiv \vec{Q}$ implies $\pi.\vec{P} \equiv \pi.\vec{Q}$. We can capture process redundancy w.r.t. this equivalence relation using group-theoretic terminology. This is inspired from model-checking where the symmetries of systems have been represented by the group of their automorphisms [28]. We can define the set of *symmetries of P for* $\equiv$ by means of the group stabiliser

$$\mathsf{Stab}_{\equiv}(\vec{P}) = \{\pi \in S_n \mid \pi.\vec{P} \equiv \vec{P}\}$$

Example 5.1. $\mathsf{Stab}_{\equiv}(\langle P, \ldots, P \rangle) = S_n$ models the extreme case where all parallel subprocesses are identical. On the contrary, the case where $\mathsf{Stab}_{\equiv}(\langle P_1, \ldots, P_n \rangle) = \{\mathsf{id}\}$ models that there is no redundancy at all between parallel processes. Intermediate examples model partial symmetries: the larger the stabilizer, the more redundancy we have. For example, if $P \not\equiv Q$, $\mathsf{Stab}_{\equiv}(\langle P, P, Q, Q, Q \rangle)$ is the subgroup of S_n generated by (1 2), (3 4) and (3 5) (using cycle notation for permutations). △

Proposition 5.1. *$\mathsf{Stab}_{\equiv}(\vec{P})$ is a subgroup of S_n.*

Proof. Consider the application $(\pi, \vec{P}) \mapsto \pi.\vec{P}$. It is a group action of S_n on the set of tuples of plain processes quotiented by the equivalence relation $\equiv$. $\mathsf{Stab}_{\equiv}(P)$ is a stabiliser of this action, hence the conclusion. □

5.2 Structural equivalence

We exhibit a notion of redundancy based on identifying processes that have an identical structure. We define *structural equivalence* $\equiv$ on plain processes as the smallest relation such that

$$P \mid Q \equiv Q \mid P \qquad (P \mid Q) \mid R \equiv P \mid (Q \mid R)$$

and that is closed under context application (that is, composition of equivalent processes with either a same process in parallel, or an input, output, or conditional instruction at toplevel). To capture equivalence modulo the equational theory, we write $P\sigma \equiv_E Q\sigma'$ when $P \equiv Q$ and σ, σ' are substitutions such that $x\sigma =_E x\sigma'$ for all variables x.

We lift $\equiv$ to tuples of plain processes by requiring that the tuples are pointwise equivalent, i.e., $\langle P_1, \ldots, P_n \rangle \equiv_E \langle Q_1, \ldots, Q_n \rangle$ if, for all i, $P_i \equiv_E Q_i$. We also add a restricted form of alpha equivalence of private names. We write $\vec{P} \equiv^A_\alpha \vec{Q}\rho$ when $\vec{P} \equiv_E \vec{Q}$ and ρ is a bijective renaming of private names such that $\rho_{|names(A) \cap \mathcal{N}_{priv}} = \mathsf{id}$, i.e., only private names outside of A may be renamed.

5.3 Universal symmetry optimisation

The first optimisation is a universal optimisation: there is no need to consider all executions of available inputs when the underlying processes are structurally equivalent. In this case we can simply choose one of these process as a representant for all the others. Note that for negative actions this is already achieved by the optimisation 2 of the compression (Section 4.2). Consider a labelled instance of the rule (In) at the root of the process, of the form

$$(\{\!\{ [\, P_i \,]^i \}\!\}_{i=1}^n, \emptyset) \xrightarrow{[\, c(\xi) \,]^\ell}_s A \qquad (\star)$$

To characterise redundancy in $(\star)$ we define

$$S = \mathsf{Stab}_{\equiv^0_\alpha}(\langle P_1, \ldots, P_n \rangle)$$

and the binary relation $\sim$ on integers of $[\![1, n]\!]$

$$i \sim j \quad \textit{iff} \quad \exists \pi \in S,\ i = \pi(j) .$$

We have that $\sim$ is an equivalence relation as S is a subgroup of S_n. Indeed, S contains the identity (hence the reflexivity); S is closed by inverse (hence the symmetry); and S is closed by composition (hence the transitivity).

Then we say that an instance of (In) is *well-formed* when ℓ is minimal in its equivalence class for $\sim$. We write $\mathbb{O}^{\vee}_{\mathsf{c+r+s}}$ the set of traces of $\mathbb{O}^{\vee}_{\mathsf{c+r}}$ whose instances of $(\star)$ are all well-formed. The correctness of this optimisation is straightforward and stated below.

Proposition 5.2. *$\mathbb{O}^{\vee}_{\mathsf{c+r+s}}$ is a correct refinement of $\mathbb{O}^{\vee}_{\mathsf{c+r}}$.*

5.4 Existential symmetry optimisation

The goal of this optimisation is to exploit symmetries when applying the matching rule: when several processes are structurally equivalent 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. Consider an instance of the rule (Match)

$$\begin{aligned}(\mathcal{P}^2 \cup \{\!\{(P_1 \mid \cdots \mid P_n,\ Q_1 \mid \cdots \mid Q_n)\}\!\}, \Phi_0, \Phi_1) \\ \xrightarrow{\tau}_s (\mathcal{P}^2 \cup \{\!\{(P_i, Q_{\pi(i)})\}\!\}_{i=1}^n, \Phi_0, \Phi_1)\end{aligned} \qquad (\star)$$

We let $A = (\mathsf{snd}(\mathcal{P}^2), \Phi_0)$, and define

$$S = \mathsf{Stab}_{\equiv_\alpha^A}(\langle Q_1, \ldots, Q_n \rangle)\,.$$

Moreover, we define the binary relation $\sim$ on permutations

$$\pi \sim \pi' \quad \textit{iff} \quad \exists u \in S,\ \pi' = \pi \circ u\,.$$

As in the previous subsection, $\sim$ is an equivalence relation as S is a subgroup of S_n. We say that an instance of $(\star)$ is *well-formed* when π is minimal (w.r.t. the lexicographic ordering) in its equivalence class for $\sim$. We denote by $\mathbb{O}_s^\exists$ the set of traces on extended bi-processes whose instances of $(\star)$ are all well-formed.

The correctness of this optimisation essentially states that it does not introduce additional false attacks. The actual proof can be found in Appendix D.

Proposition 5.3. *$\mathbb{O}_s^\exists$ is a correct refinement of $\mathbb{O}_{\mathsf{all}}^\exists$.*

6 SYMBOLIC SETTING

Even though we do not consider unbounded replication, the semantics of our process calculus defines an infinite transition system due to the unbounded number of possible inputs that can be provided by the adversary. To perform exhaustive verification of such infinite systems, it is common to resort to *symbolic techniques* abstracting inputs by symbolic variables and constraints. We briefly describe in this section how our optimisations are integrated in the symbolic procedure underlying the DeepSec tool.

6.1 DeepSec's baseline procedure

Symbolic setting. In the DeepSec tool [17] and its underlying theory [16], the deduction capabilities of the attacker are represented by so-called *deduction facts* $X \vdash^? u$, intuitively meaning that the attacker is able to deduce the term u by the means of a recipe represented by the variable X. Additionally, conditional branching, *e.g.* if $u = v$ then $\ldots$ else $\ldots$, is represented by *equations* $u =^? v$ and *disequations* $u \neq^? v$.

To represent infinitely many extended processes, [16] relies on *symbolic processes* $(\mathcal{P}, \Phi, C)$ where $\mathcal{P}$ and Φ are, as in our setting, a multiset of processes and a frame respectively. The difference is that the processes and frame may contain free variables: they model the variables bound by inputs and are subject to constraints in C. These constraints are a conjunction of deduction facts, equations and disequations. For example, if we consider the process

$$P = c(x).\,\mathsf{if}\ \mathsf{proj}_1(x) = t\ \mathsf{then}\ \overline{c}\langle \mathsf{h}(x) \rangle$$

then after executing symbolically the input and the positive branch of the test, we reach the symbolic process

$$(\{\!\{0\}\!\},\ \{\mathsf{ax} \mapsto \mathsf{h}(x)\},\ X \vdash^? x \wedge \mathsf{proj}_1(x) =^? t)$$

A concrete extended process is thus represented by any ground instantiation of the free variables of the symbolic process that satisfies the constraints in C. Such instantiations are called *solutions*, and therefore form an abstraction of concrete traces treated as symbolic objects and constraint solving.

Example 6.1. Let us continue Example 2.5 where we introduced an equivalence statement to model the secrecy of a signature key. After the symbolic execution of the head input of $V(R)$, finding an attack trace reduces to finding a solution of the constraint:

$$C = X \vdash^? x \wedge \mathsf{proj}_2(\mathsf{verify}(x, \mathsf{pk}(s))) =^? \mathsf{pk}(s)$$

Intuitively the internal constraint solver will gradually deduce that solutions to this constraint need to map x to a term of the form $\mathsf{sign}(y, z)$, then y to a term of the form $\langle y_1, y_2 \rangle$, where y_2 and z point to recipes for constructing $\mathsf{pk}(s)$ and s, respectively. △

Partition tree. To decide trace equivalence between two processes P and Q, the procedure underlying DeepSec builds a refined tree of symbolic executions of P and Q, called a *partition tree*. This finite, symbolic tree intuitively embodies all scenarios of (potential violations of) equivalence, and the final decision criterion is a simple syntactic check on this tree.

More technically, nodes of the partition tree contain sets of symbolic processes derived from P or Q; that is, a branch is a symbolic abstraction of a subset of $\mathbb{T}(P) \cup \mathbb{T}(Q)$. It is constructed in a way that each node contains all—and only—equivalent processes reachable from P or Q with given trace actions tr. When generating this partition tree, trace equivalence holds if and only if each node contains at least one symbolic process derived from P and one from Q.

6.2 Symbolic matching

Subprocess matchings. In order to make the integration into DeepSec 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 (as defined in Section 4.1) that we call *session matchings*. A precise definition is given in Appendix A, as well as 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. The set of all these bijections is then updated at each new symbolic transition step in the partition tree, among others to satisfy the requirement that matched subprocesses should have the same skeleton (recall Definition 3.1).

Example 6.2. Consider two initial processes

$$P = c(x).P_0 \mid c(x).P_1 \mid \overline{c}\langle u \rangle.P_2 \qquad Q = c(x).Q_0 \mid \overline{c}\langle u' \rangle.Q_1 \mid c(x).Q_2\,.$$

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

$$(\{[\,P\,]^0\}, \emptyset, \emptyset) \qquad\qquad (\{[\,Q\,]^0\}, \emptyset, \emptyset)\,.$$

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

$$(\{\!\![\, c(x).P_0 \,]^{0.1} ;\ [\, c(x).P_1 \,]^{0.2} \ ;\ [\, \overline{c}\langle u\rangle.P_2 \,]^{0.3}]\!\!\},\emptyset,\emptyset)$$
$$(\{\!\![\, c(x).Q_0 \,]^{0.1};\ [\, \overline{c}\langle u'\rangle.Q_1 \,]^{0.2};\ [\, c(x).Q_2 \,]^{0.3}]\!\!\},\emptyset,\emptyset)$$

There are then only two possible bijections of labels that respect the skeleton requirement of twin processes:

$$\begin{array}{lcl} 0.1 \mapsto 0.1 & & 0.1 \mapsto 0.3 \\ 0.2 \mapsto 0.3 & \text{and} & 0.2 \mapsto 0.1 \\ 0.3 \mapsto 0.2 & & 0.3 \mapsto 0.2 \end{array} \qquad \triangle$$

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 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 $C_1, \ldots, C_n$ is 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 decision criterion for equivalence is then strenghtened. 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 $\exists$.

6.3 Integration

From a high-level of abstraction, the integration of the universal optimisations described in sections Sections 4 and 5 prune some branches of the partition tree—those that abstract traces that do not belong to $\mathbb{O}^\forall_{\mathsf{c+r+s}}$. For instance in Section 4.2, we showed that to prove equivalence by session, we can always perform output and parallel 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 *reduction* $\mathbb{O}^\forall_{\mathsf{c+r}}$ described in Section 4.3. 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 notion of independence (Definition 4.1) 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 3.8, such partial implementations do not break correctness.

7 EXPERIMENTS

Equivalence by session in practice. Based on the high-level description of the previous section, we extended the implementation of the DeepSec tool 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 3.2.
(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 3.2, 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 $P \sqsubseteq_{tr} Q$ 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 $P \not\approx_{tr} Q$. Otherwise, the false attack may guide us to discover a real attack. Our analysis of session equivalence indeed consider traces with a specific shape (see Sections 4 and 5): thus, we implemented a simple heuristic that, whenever a false attack is discovered, also checks whether different permutations of its actions 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 to future work the design of an efficient and complete decision procedure for trace equivalence that builds on a preliminary analysis of equivalence by session.

Experimental setting. We report experiments (Figure 4) comparing the scope and efficiency of the following two approaches for proving trace equivalence:

- The original version of DeepSec as a baseline for comparison;
- The analysis leveraging our contributions (preliminary analysis of equivalence by session, test of false attack if it fails, and the heuristic attempting to reconstruct a true attack from false ones).

We describe the benchmarks below in more details. The column **# roles** is an indicator of the intricacy of the system, namely the number of parallel processes that the model file exhibits.

All benchmarks were carried out on 20 Intel Xeon 3.10GHz cores, with 50 Gb of memory. We ran the toy example described in this paper on a single core to illustrate simply the algorithmic improvements compared to DeepSec. As DeepSec supports parallelisation, we distributed the computation of the other, bigger proofs over 20 cores. The implementation and the specification files are available at https://deepsec-prover.github.io/.

Running example: toy voting. We modelled the toy e-voting protocol that was partially specified in Example 2.3 and ran an analysis of vote privacy. The parts of this protocol that are not

Protocol	scenario	# roles	DeepSec baseline	DeepSec eq. by session
Toy voting (parallel mix)	2 honest 1 dishonest	7	✓ 1m41s	✗ <1s
Toy voting	2 honest + 1 dishonest	7	✓ 15s	✓ <1s
	2 honest + 2 dishonest	9	✓ 30m	✓ <1s
	2 honest + 3 dishonest	11	⏱	✓ 22s
BAC	2 identical	4	↯ <1s	↯ <1s
	2 identical + 1 fresh	6	⏱	↯ 2s
	3 identical + 1 fresh	8	⏱	↯ 3s
	2 identical + 2 fresh	8	⏱	✓ 1m20s
	4 identical + 1 fresh	10	⏱	↯ 4s
	3 identical + 2 fresh	10	⏱	↯ 9m22s
	2 identical + 3 fresh	10	⏱	✓ 11h06m
Helios vote swap	no revote	6	✓ <1s	✓ <1s
	$2 \times A$ $1 \times B$	11	✓ 2h41m	✓ 1m2s
	$3 \times A$ $1 \times B$	12	⏱	✓ 2m40s
	$3 \times A$ $2 \times B$	13	⏱	✓ 7m40s
	$4 \times A$ $2 \times B$	14	⏱	✓ 16m36s
	$7 \times A$ $3 \times B$	18	⏱	✓ 3h53m
Helios BPRIV	2 honest + 1 dishonest 7 ballots (19 scenarios)	9 (each)	⏱	✓ 3m26s (total)
Scytl	vote privacy	5	✓ 3m8s	✓ 1s
AKA	anonymity	8	✓ 30s	✓ 4s

✓ trace equivalence verified ↯ trace equivalence violated ⏱ timeout (12 hours)
✗ false attack (disproves session equivalence but unable to conclude for trace equivalence)

Figure 4: Experimental evaluation

specified in the body of the paper, i.e. mostly the mixnet, can be represented in several ways that may trigger or not a false attack. Mixnets are usually modelled as processes receiving the values to mix, and then outputing them in an arbitrary order induced by the inherent non-determinism of concurrency. However, the mixing of two elements can be performed using two models (where c is a private channel):

$$\mathsf{MixSeq} = c(x).\, c(y).\, (\overline{c}\langle x\rangle \mid \overline{c}\langle y\rangle)$$
$$\mathsf{MixPar} = (c(x).\, \overline{c}\langle x\rangle) \mid (c(y) \mid \overline{c}\langle y\rangle)$$

In the second case, subprocess-matching constraints arise earlier in the trace, triggering a false attack. However, the natural modelling of MixSeq allows to complete a security proof. We observed the same behaviour on other experimentation on voting protocols with mixnets.

On the other hand, the baseline version of DeepSec does not suffer from this shortcoming as it decides trace equivalence precisely. However, due to the combinatorics of its proofs the tool fails to terminate the analysis within 12h when adding two dishonest voters to the models.

BAC. We also study the Basic Access Control (BAC) protocol implemented in European electronic passport [29]. The security property we study is unlinkability formalised as an equivalence statement. This property is out of the scope of the fine-grained diff-equivalence of ProVerif and Tamarin, and computationally out of reach of tools for a bounded number of sessions.

Unlinkability is formalised here as the indistinguishability of two situations where n systems—each consisting of a passport and a reader—are put in parallel: on one side all n systems are distinct (fresh), while on the other side a same system may appear several times. Our analysis indicates that, depending on the precise setting, the security property may be violated in the model or not. This is due to the error codes raised when a passport communicates with a wrong reader: depending on how many identical systems the process contains, the same number of such errors may not be observable. Although not present in the result table, we also implemented inclusion by session (i.e. $\sqsubseteq_s$) as it is sometimes used to define other flavours of unlinkability.

Helios. As a bigger e-voting case study, we consider the Helios e-voting protocol [2]. We analyse vote privacy of a version that uses zero-knowledge proofs to ensure the voter knows the plaintext of her vote, thus avoiding copy-attacks [24]. Vote privacy is formalised as in our running example, using a vote-swapping model.

A reduction result of Arapinis et al. [3] ensures that it is sufficient to consider two honest voters and one dishonest voter (that is implicit in the model, embedded in the intruder capabilities) to obtain a proof of the system for an unbounded number of sessions. Such scenarios could already be handled by automated analysers, e.g. DeepSec [16] and ProVerif [3]. However, when revoting is allowed, one needs to consider all scenarios when the tally accepts 7 ballots. In particular, it is not sufficient to consider only revotes by the adversary, but also arbitrary revotes of the two honest voters. In Figure 4 we listed several scenarios, indexed by how many times the honest voters A and B are sending revotes.

This kind of analysis is out of the scope of many automated analysers. For example, Figure 4 shows that DeepSec (baseline) fails to prove after 12h of computation any scenarios where more than one honest revote is emitted. In [3] the ProVerif proofs are limited to dishonest revotes. We compiled several intermediary scenarios to give an overview of the verification-time growth using our prototype, but all are subsumed by the last scenario were we allow A to revote 7 times and B 3 times. Indeed, using a simple symmetry argument on A and B this covers all scenarios where honest voters cast a total of 7 ballots or less. Note however that, strictly speaking, the reduction result of [3] does not bound the number of *emitted* honest revotes (that may not be effectively received by the ballot box) that have to be considered during an analysis of vote privacy; extensions of this reduction should be considered in the future.

We also experimented an other model of voting privacy inspired by the game-based definition BPRIV [10]. In this definition the (re)votes are dicted to honest voters by the adversary, which permits to effectively model revotes of arbitrary values. As reported in Figure 4 DeepSec was able to handle the 19 queries modelling all revote scenarios for 7 emitted ballots, in a total of a few minutes.

Other case studies. As side experiments, we also tried our prototype on other model files of similar tools that we could find in the literature. We performed for example an analysis of vote privacy of an e-voting protocol by Scytl deployed in the Swiss canton of Neuchâtel, based on the ProVerif file presented in [23]. We also studied anonymity in a model of the AKA protocol deployed in 3G telephony networks [4] (without XOR), presented in the previous version of DeepSec [16].

8 CONCLUSION AND FUTURE WORK

In this paper we introduce a new process equivalence, the equivalence by session. We show that it is a sound proof technique for trace equivalence which allows for several optimisations when performing automated verification. This includes powerful partial order reductions, that were previously restricted to the class of determinate processes, and allows to exploit symmetries that naturally arise when verifying multiple sessions of a same protocol. In addition to the theoretical basis we have implemented these techniques in the DeepSec tool and evaluated their effectiveness in practice. The optimisations indeed allowed for efficient verification of non-determinate processes that were previously out of scope of existing techniques.

We also discussed how to handle false attacks, that are a natural consequence of the fact that equivalence by session is a strict refinement of trace equivalence. We implemented a test to verify automatically, when equivalence by session is disproved, whether the underlying attack is genuine with respect to trace equivalence. When this is not the case, as part of future work it would be interesting to refine the part of the proof that failed, while exploiting that some parts of the system have already been shown equivalent.

Acknowledgements. The authors wish to thank the anonymous reviewers for their helpful comments. 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), as well as from the French ANR under the projects SEQUOIA (ANR-14-CE28-0030-01) and TECAP (ANR-17-CE39-0004-01). Itsaka Rakotonirina benefits from a Google PhD Fellowship.

REFERENCES

[1] Martín Abadi, Bruno Blanchet, and Cédric Fournet. 2018. The Applied Pi Calculus: Mobile Values, New Names, and Secure Communication. *J. ACM* (2018).
[2] B. Adida. 2008. Helios: web-based open-audit voting. In *USENIX Security Symposium.*
[3] Myrto Arapinis, Véronique Cortier, and Steve Kremer. 2016. When are three voters enough for privacy properties?. In *European Symposium on Research in Computer Security (ESORICS).*
[4] M. Arapinis, L. Mancini, E. Ritter, M. Ryan, N. Golde, K. Redon, and R. Borgaonkar. 2012. New privacy issues in mobile telephony: fix and verification. In *ACM Conference on Computer and Communications Security (CCS).*
[5] David Baelde, Stéphanie Delaune, and Lucca Hirschi. 2014. A Reduced Semantics for Deciding Trace Equivalence Using Constraint Systems. In *International Conference on Principles of Security and Trust (POST).*
[6] David Baelde, Stéphanie Delaune, and Lucca Hirschi. 2015. Partial Order Reduction for Security Protocols. *International Conference on Concurrency Theory (CONCUR)* (2015).
[7] David Baelde, Stéphanie Delaune, and Lucca Hirschi. 2018. POR for Security Protocol Equivalences - Beyond Action-Determinism. In *European Symposium on Research in Computer Security (ESORICS).*
[8] David A. Basin, Jannik Dreier, Lucca Hirschi, Sasa Radomirovic, Ralf Sasse, and Vincent Stettler. 2018. A Formal Analysis of 5G Authentication. In *ACM Conference on Computer and Communications Security (CCS).*
[9] David A. Basin, Jannik Dreier, and Ralf Sasse. 2015. Automated Symbolic Proofs of Observational Equivalence. In *ACM Conference on Computer and Communications Security (CCS).*
[10] David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. 2015. A comprehensive analysis of game-based ballot privacy definitions. In *IEEE Symposium on Security and Privacy (S&P).*
[11] Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. 2017. Verified Models and Reference Implementations for the TLS 1.3 Standard Candidate. In *IEEE Symposium on Security and Privacy (S&P).*
[12] Bruno Blanchet, Martín Abadi, and Cédric Fournet. 2005. Automated Verification of Selected Equivalences for Security Protocols. In *20th IEEE Symposium on Logic in Computer Science (LICS).*
[13] Bruno Blanchet, Martín Abadi, and Cédric Fournet. 2008. Automated verification of selected equivalences for security protocols. *J. Log. Algebr. Program.* (2008).
[14] Rohit Chadha, Vincent Cheval, Ştefan Ciobâcă, and Steve Kremer. 2016. Automated verification of equivalence properties of cryptographic protocols. *ACM Transactions on Computational Logic* (2016).
[15] Vincent Cheval and Bruno Blanchet. 2013. Proving More Observational Equivalences with ProVerif. In *Proceedings of the 2nd International Conference on Principles of Security and Trust (POST'13).*
[16] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. 2018. DEEPSEC: Deciding Equivalence Properties in Security Protocols – Theory and Practice. In *IEEE Symposium on Security and Privacy (S&P).*
[17] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. 2018. The DEEPSEC prover. In *International Conference on Computer Aided Verification (CAV).*
[18] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. 2019. Exploiting symmetries when proving equivalence properties for security protocols (technical report). available at https://hal.archives-ouvertes.fr/hal-02267866.
[19] Ivan Cibrario, Luca Durante, Riccardo Sisto, and Adriano Valenzano. 2004. Exploiting symmetries for testing equivalence in the spi calculus. In *International Symposium on Automated Technology for Verification and Analysis (ATVA).*
[20] Edmund M. Clarke, Somesh Jha, and Wilfredo R. Marrero. 2003. Efficient verification of security protocols using partial-order reductions. *STTT* (2003).
[21] Véronique Cortier and Stéphanie Delaune. 2009. A method for proving observational equivalence. In *IEEE Computer Security Foundations Symposium (CSF).*

[22] Véronique Cortier, Stéphanie Delaune, and Antoine Dallon. 2017. SAT-Equiv: an efficient tool for equivalence properties. In *IEEE Computer Security Foundations Symposium (CSF)*.

[23] Véronique Cortier, David Galindo, and Mathieu Turuani. 2018. A formal analysis of the Neuchâtel e-voting protocol. In *IEEE European Symposium on Security and Privacy (EuroS&P)*.

[24] Véronique Cortier and Ben Smyth. 2013. Attacking and fixing Helios: An analysis of ballot secrecy. *Journal of Computer Security* (2013).

[25] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. 2017. A Comprehensive Symbolic Analysis of TLS 1.3. In *ACM Conference on Computer and Communications Security (CCS)*.

[26] D. Dolev and A.C. Yao. 1981. On the Security of Public Key Protocols. In *Symposium on Foundations of Computer Science (FOCS)*.

[27] Luca Durante, Riccardo Sisto, and Adriano Valenzano. 2003. Automatic testing equivalence verification of spi calculus specifications. *ACM Transactions on Software Engineering and Methodology (TOSEM)* (2003).

[28] E. Allen Emerson and A. Prasad Sistla. 1996. Symmetry and model checking. *Formal methods in system design* (1996).

[29] PKI Task Force. 2004. *PKI for machine readable travel documents offering ICC read-only access.* Technical Report. International Civil Aviation Organization.

[30] Sebastian Mödersheim, Luca Viganò, and David A. Basin. 2010. Constraint differentiation: Search-space reduction for the constraint-based analysis of security protocols. *Journal of Computer Security* (2010).

[31] Sonia Santiago, Santiago Escobar, Catherine Meadows, and José Meseguer. 2014. A Formal Definition of Protocol Indistinguishability and Its Verification Using Maude-NPA. In *International Workshop on Security and Trust Management (STM)*.

[32] Alwen Tiu, Nam Nguyen, and Ross Horne. 2016. SPEC: An Equivalence Checker for Security Protocols. In *Asian Symposium on Programming Languages and Systems (APLAS)*.

A EXPLICIT SESSION MATCHINGS

In this section we present an alternative characterisation of equivalence by session. The process matchings operated by twin processes (in particular in the rule (Match) of the semantics) are represented by an explicit permutation with properties mirroring the structure of twin processes.

Twin-process based characterisations makes it easier to define symmetry-based optimisations and limit the manipulation of permutations to the minimum, thus simplifying many proofs. On the contrary, the formalism presented here makes a closer link with the definition of trace equivalence: 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 4.1 to make reference to subprocess positions. In the rest of the paragraph, we refer to two plain processes in $\rightsquigarrow$-normal form P, Q such that $\mathsf{skel}(P) = \mathsf{skel}(Q)$ and two labelled traces $t \in \mathbb{T}(P), t' \in \mathbb{T}(Q)$ such that $tr(t) = tr(t')$:

$$t : A_0 \xrightarrow{[\,\alpha_1\,]^{\ell_1}} \cdots \xrightarrow{[\,\alpha_n\,]^{\ell_n}} A_n \quad t' : B_0 \xrightarrow{[\,\alpha_1\,]^{\ell'_1}} \cdots \xrightarrow{[\,\alpha_n\,]^{\ell'_n}} B_n$$

We write L and L' the sets of labels appearing in t and t', respectively.

Definition A.1. A *session matching* for t and t' is a bijection $\pi : L \to L'$ verifying the following properties

(1) $\pi(\varepsilon) = \varepsilon$
(2) $\forall i \in [\![1, n]\!],\ \pi(\ell_i) = \ell'_i$
(3) $\forall \ell \cdot p \in dom(\pi), \exists q, \pi(\ell \cdot p) = \pi(\ell) \cdot q$
(4) for all $i \in [\![0, n]\!]$, if $\pi(\ell) = \ell'$ and A_i and B_i respectively contain a process $[\,P\,]^{\ell}$ and a process $[\,Q\,]^{\ell'}$, then $\mathsf{skel}(P) = \mathsf{skel}(Q)$.

Proposition A.2. *The following two points are equivalent:*

(1) There exists a session matching for t and t'.
(2) $\exists t^2 \in \mathbb{T}(P, Q),\ \mathsf{fst}(t^2) = t$ and $\mathsf{snd}(t^2) = t'$.

Proof of 1⇒2. The trace t^2 can be easily constructed by induction on the length of t:

- Items 1 and 4 of Definition A.1 ensure that the twin-processes in t^2 are composed of pairs of processes with the same skeleton as expected,
- Item 2 ensures that pairs of transitions of P and Q can be mapped into transitions of twin-processes, and
- The permutations 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; that is, if P^2 is such a process, we may refer to the labellings of $\mathsf{fst}(P^2)$ and $\mathsf{snd}(P^2)$. Thus, each instance of rule (Match) in t^2

$$(\{\!\![\,([\,P_1 \mid \cdots \mid P_n\,]^{\ell}, [\,Q_1 \mid \cdots \mid Q_n\,)]^{\ell'}\,]\!\!\} \cup \mathcal{P}^2, \Phi_0, \Phi_1) \xrightarrow{\tau}_s (\{\!\![\,([\,P_i\,]^{\ell \cdot i}, [\,Q_{\sigma(i)}\,]^{\ell' \cdot i})\,]\!\!\}_{i=1}^{n} \cup \mathcal{P}^2, \Phi_0, \Phi_1)$$

can be associated with a permutation σ and two labels ℓ, ℓ'. We list all such elements $\sigma_1, \ell_1, \ell'_1, \ldots, \sigma_p, \ell_p, \ell'_p$ when considering all instances of rule (Match) in t^2. In particular the ℓ_i's are pairwise distinct and, if L is the set of labels appearing in t, we have

$$L = \{\varepsilon\} \cup \bigcup_{i=1}^{p} \{\ell_i \cdot j \mid j \in dom(\sigma_i)\}\,.$$

An analoguous statement can be done for L' the set of labels appearing in t'. Therefore the following equations well define a bijection $\pi : L \to L'$:

$$\pi(\varepsilon) = \varepsilon \qquad \forall p \in dom(\sigma_i),\ \pi(\ell_i \cdot p) = \ell'_i \cdot \sigma_i(p)\,.$$

A quick induction on the length of t^2 shows that π is a session matching for t and t'. □

Link with equivalence. As a direct corollary, we give an alternative characterisation of equivalence by session.

Proposition A.3. *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 $tr(t) = tr(t')$ (labels removed) and $\Phi(t) \sim \Phi(t')$

B FALSE ATTACKS AND DETERMINACY

In this section we give a detailed proof of Proposition 3.3, stating that equivalence by session suffers from no false attacks for determinate processes. The intuition of this rather technical proof is determinacy ensures the uniqueness of traces with a given action word; in particular, matched traces need follow the same structure, and a proof of equivalence by session can be derived from a proof of trace equivalence.

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

$$t : A_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} A_n \qquad t' : B_0 \xrightarrow{\beta_1} \cdots \xrightarrow{\beta_p} B_p$$

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 B.1. 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 B.2. *If P, Q are plain processes in $\rightsquigarrow$-normal form such that* $\mathsf{skel}(P) = \mathsf{skel}(Q)$*:*

- $P \sqsubseteq_{tr} Q$ *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 4.2). □

Definition B.3. We say that $(\{\!\{P_1,\ldots,P_n\}\!\},\Phi)$ 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 B.4. *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) $P \approx_{tr} Q$

(2) $\forall t \in \mathbb{T}_\tau(P),\ \exists t' \in \mathbb{T}_\tau(Q),\ \begin{cases} tr(t) = tr(t') \\ \Phi(t) \sim \Phi(t') \\ \mathsf{skel}(t) = \mathsf{skel}(t') \end{cases}$

Proof of 2⇒1. Given two determinate extended processes A, B, we write $\varphi(A,B)$ the property stating that

$$\forall t \in \mathbb{T}_\tau(A),\ \exists t' \in \mathbb{T}_\tau(B),\ \begin{cases} tr(t) = tr(t') \\ \Phi(t) \sim \Phi(t') \\ \mathsf{skel}(t) = \mathsf{skel}(t') \end{cases}.$$

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, $\varphi(A,B) \Rightarrow A \sqsubseteq_{tr} B$ and $\varphi(A,B) \Rightarrow \varphi(B,A)$.

The first implication is immediate. As for the second implication, 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 $tr(s) = tr(s')$, $\Phi(t) \sim \Phi(t')$ 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 we know that neither of A_1 nor B_1 contain a parallel operator due to the $\rightsquigarrow$-normalisation, 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 $A_0 \approx_{tr} B_0$ 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 $A_1 \approx_{tr} B_1$ and $\mathsf{Inv}(A_1, B_1)$ hold in order to apply the induction hypothesis and conclude.

→ *proof that* $A_1 \approx_{tr} B_1$.

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 $A_0 \approx_{tr} B_0$ 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 $A_1 \sqsubseteq_{tr} B_1$, and a symmetric argument can be used for the converse inclusion $B_1 \sqsubseteq_{tr} A_1$.

→ *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 $A_1 \approx_{tr} B_1$ and $\mathsf{Inv}(A_1, B_1)$ hold in order to apply the induction hypothesis and conclude.

→ *proof that* $A_1 \approx_{tr} B_1$.

The argument is the same as its analogue in *case 1*, using the determinacy of B_0 instead of its τ-determinism.

→ *proof that* $\mathsf{Inv}(A_1, B_1)$.

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 us fix a skeleton $s \in \mathsf{skel}(A_1) \setminus \mathsf{skel}(B_1)$. 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 $A_0 \approx_{tr} B_0$: 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 $A_0 \approx_{tr} B_0$, 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. Proposition B.4 is the core result of the proof. We now connect it with the equivalence by session by using the characterisation of Appendix A.

Proposition B.5. *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 $tr(t) = tr(t')$ 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

$$t : A_0 \xrightarrow{[\,\alpha_1\,]^{\ell_1}} \cdots \xrightarrow{[\,\alpha_n\,]^{\ell_n}} A_n \quad t' : B_0 \xrightarrow{[\,\alpha_1\,]^{\ell'_1}} \cdots \xrightarrow{[\,\alpha_n\,]^{\ell'_n}} B_n$$

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

$$A_{n-1} = (\{\!\![\,P\,]^{\ell_n}\}\!\!\} \cup \mathcal{P}, \Phi) \qquad B_{n-1} = (\{\!\![\,Q\,]^{\ell'_n}\}\!\!\} \cup \mathcal{Q}, \Psi)$$

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

$$A_n = (\{\!\![\,P'\,]^{\ell_n}\}\!\!\} \cup \mathcal{P}, \Phi') \qquad B_n = (\{\!\![\,Q'\,]^{\ell'_n}\}\!\!\} \cup \mathcal{Q}, \Psi')$$

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 4 of Definition A.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 4 of Definition A.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 Propositions A.3 to B.5 justify the following corollary (that actually appears to be stronger than the expected Proposition 3.3).

Corollary B.6. *If P and Q are determinate plain processes in $\rightsquigarrow$-normal form, $P \approx_{tr} Q$ iff $P \sqsubseteq_s Q$.*

C CORRECTNESS OF PARTIAL-ORDER REDUCTIONS

In this section we give the proofs of the main technical results of our partial-order reductions, namely that traces can be considered up to permutation of independent actions. First we prove Proposition 4.2 for traces of two actions.

Proposition C.1. *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 $u^2 : A^2 \overset{\beta\alpha}{\Longrightarrow}_s B^2$ such that $\mathsf{fst}(u^2) = u$, there exists $t^2 : A^2 \overset{\alpha\beta}{\Longrightarrow}_s B^2$ 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

$$A = (\mathcal{P} \cup \mathcal{Q} \cup \mathcal{R}, \Phi) \xrightarrow{\alpha} (\mathcal{P}' \cup \mathcal{Q} \cup \mathcal{R}, \Phi') \xrightarrow{\beta}_s (\mathcal{P}' \cup \mathcal{Q}' \cup \mathcal{R}, \Phi'')$$

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 $\mathsf{ax} \neq \mathsf{ax}'$, $\{\mathsf{ax}, \mathsf{ax}'\} \cap dom(\Phi) = \emptyset$. Then we choose

$$u : (\mathcal{P} \cup \mathcal{Q} \cup \mathcal{R}, \Phi) \xrightarrow{\beta} (\mathcal{P} \cup \mathcal{Q}' \cup \mathcal{R}, \Phi \cup \{\mathsf{ax}' \mapsto m'\})$$
$$\xrightarrow{\alpha} (\mathcal{P}' \cup \mathcal{Q}' \cup \mathcal{R}, \Phi'')\,.$$

□

Then Proposition 4.2 can be obtained by induction on the hypothesis of π permuting independent actions of tr, using Proposition C.1. We actually prove the stronger result:

Proposition C.2. *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 $u^2 : A^2 \overset{\pi.\mathsf{tr}}{\Longrightarrow}_s B^2$ such that $\mathsf{fst}(u^2) = \pi.t$, there exists $t^2 : A^2 \overset{\mathsf{tr}}{\Rightarrow}_s B^2$ 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 $tr(u)$ (if labels are included). We then construct $\pi.t$ by induction on the hypothesis that π permutes independent actions of $tr(t)$. Let us write

$$t : A = A_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} A_n = B\,.$$

If $\pi = \mathrm{id}$ 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 $\mathrm{tr}' = \alpha_p \cdots \alpha_{i-1}\alpha_{i+1}\alpha_i\alpha_{i+2} \cdots \alpha_n$. By Proposition C.1, 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 $u^2 : A^2 \overset{\mathrm{tr}'}{\Longrightarrow}_s B^2$ verifying $\mathsf{fst}(u^2) = u$, there exists $t^2 : A^2 \overset{\mathrm{tr}}{\Longrightarrow}_s B^2$ such that $\mathsf{fst}(t^2) = t$. Then since π_0 permutes independent actions of $\mathrm{tr}' = tr(u)$, it sufficies to choose $\pi.t = \pi_0.u$ by induction hypothesis. □

And finally we have the Proposition 4.3 that follows from this result.

Proof of Proposition 4.3. Let $\approx_i = \sqsubseteq_i \cap \sqsupseteq_i$ the 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 $t \in \mathbb{T}(P) \cap \mathbb{O}_2^{\forall}$. Therefore by hypothesis, there exists π permuting independent actions of t such that $\pi.t \in \mathbb{O}_1^{\forall}$. Since $P \sqsubseteq_1 Q$, there is $u^2 \in \mathbb{T}(P, Q)$ such that

$$\pi.t = \mathsf{fst}(u^2) \sim \mathsf{snd}(u^2)\,.$$

Therefore by Proposition C.2, there exists $t^2 \in \mathbb{T}(P, Q)$ such that $t = \mathsf{fst}(t^2) \sim \mathsf{snd}(t^2)$. □

D CORRECTNESS OF THE REDUCTIONS BY SYMMETRY

In this section we prove of the technical optimisation relying on symmetries of matchings (Proposition 5.3). For that we introduce a notion of equivalence of traces; this is intuitively the invariant preserved by permutation of structurally-equivalent subprocesses.

Definition D.1. We write

$$(\{\!\{(P_i, Q_i)\}\!\}_{i=1}^n, \Phi_0, \Phi_1) \equiv_\alpha (\{\!\{(P_i, Q_i'\rho)\}\!\}_{i=1}^n, \Phi_0, \Phi_1'\rho)$$

when ρ is a bijective renaming of private names, $Q_i \equiv_E Q_i'$ for all i, and $\Phi_1 \equiv_E \Phi_1'$. We extend this to traces by writing $A_0^2 \xrightarrow{\alpha_1}_s \cdots \xrightarrow{\alpha_n}_s A_n^2 \equiv_\alpha B_0^2 \xrightarrow{\alpha_1}_s \cdots \xrightarrow{\alpha_n}_s B_n^2$ when $A_0^2 = B_0^2$ and $A_i^2 \equiv_\alpha B_i^2$ for all $i > 0$.

Lemma D.2. *$\equiv_\alpha$ is an equivalence relation.*

Proof. Reflexivity and Symmetry are immediate, but transitivity requires the observation that for any terms t, t' and bijective renaming of private names ρ, $t =_E t'$ entails $t\rho =_E t'\rho$. □

Proposition D.3. *The relation $\equiv_\alpha$ has the following properties*

(*i*) $\forall t^2 \in \mathbb{T}(A^2), \exists s^2 \in \mathbb{O}_s^{\exists}, s^2 \equiv_\alpha t^2$

(*ii*) *if* $t^2 \equiv_\alpha s^2$, *then* $\mathsf{fst}(t^2) \sim \mathsf{snd}(t^2)$ *iff* $\mathsf{fst}(s^2) \sim \mathsf{snd}(s^2)$

(*iii*) *if* $t^2 \equiv_\alpha s^2$, *then* $\mathsf{fst}(t^2) = \mathsf{fst}(s^2)$

Proof. The property (*i*) can be proved by induction on the length of the trace. The main argument is that if $A^2 \xrightarrow{\alpha}_s B^2$ is ill-formed, then there exists a well-formed transition $A^2 \xrightarrow{\alpha}_s C^2$ such that $B^2 \equiv_\alpha C^2$. The property (*ii*) follows from the fact that $\Phi =_E \Phi'$ implies $\Phi \sim \Phi'$, and $\Phi \sim \Phi\rho$ for any bijective renaming of private names ρ. The property (*iii*) is immediate. □

Then Proposition 5.3 is a simple corollary of this proposition:

Proof of Proposition 5.3. Let us write $\approx$ the notion of equivalence induced by the optimisation $\mathbb{O}_s^{\exists}$. The inclusion $\approx \subseteq \sqsubseteq_s$ is immediate. Let us then assume that $P \sqsubseteq_s Q$ and prove that $P \sqsubseteq Q$. Let $t \in \mathbb{T}(P)$. By hypothesis, there is $t^2 \in \mathbb{T}(P, Q)$ such that

$$t = \mathsf{fst}(t^2) \sim \mathsf{snd}(t^2)\,.$$

By Proposition D.3 (Item (*i*)), there exists $s^2 \in \mathbb{O}_s^{\exists}$ such that $t^2 \equiv_\alpha s^2$. Therefore we have, by Items (*i*) and (*ii*),

$$t = \mathsf{fst}(s^2) \sim \mathsf{snd}(s^2)\,.$$ □