HAL
open science

Exploiting symmetries when proving equivalence properties for security protocols (Technical report)

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 (Technical report). 2019. hal-02267866v1

HAL Id: hal-02267866

https://hal.science/hal-02267866v1

Preprint submitted on 19 Aug 2019 (v1), last revised 17 Apr 2020 (v3)

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 (*Technical report*)

Vincent Cheval, Steve Kremer, Itsaka Rakotonirina
INRIA Nancy Grand-Est & LORIA

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 [DY81], have been successful in analysing complex protocols, including for instance the recent TLS 1.3 proposal [BBK17, CHH$^+$17] and the upcoming 5G standard [BDH$^+$18b].

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 [ABF18]. 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 [CKR18a] 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*) [CD09]. 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* [BDH15, CKR18b]. 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 information 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 the partial-order reductions presented in [BDH15] 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 [ES96], 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 [CKR18b], 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 [For04] 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 [Adi08]. Automated analyses of this protocol exist when no revote is allowed, or is limited to one revote from a honest voter [ACK16, CKR18a]. In this paper, we analyse several models covering revote scenarios for 7 emitted honest ballots.

This document is the technical report of the conference paper [CKR19]. It contains full technical proofs and generalised results, as well as a different running example to provide a complementary presentation.

Related work

The partial-order reductions (por) for the verification of cryptographic protocols were first introduced by Clark et al. [CJM03]: while well developed in verification of reactive systems they do not easily carry over to security protocols, mainly due to the symbolic treatment of attacker knowledge. Mödersheim et al. [MVB10] proposed por techniques that are suitable for symbolic methods based on constraint solving. However, both the techniques of [CJM03] and [MVB10] are only correct for trace properties.

Partial order reduction techniques for equivalence properties were only introduced more recently by Baelde et al. [BDH14, BDH15]: implementing these techniques in the APTE tool resulted in spectacular speed-ups. Other state-of-the-art tools, AKISS [CCCK16a] and DeepSec [CKR18a], 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. [BDH18a] propose por techniques that also apply to non-determinate processes (but do not support private channels) and implement these techniques in the DeepSec tool. Unfortunately, these techniques introduce a computational overhead, that limits the efficiency gain. As our experiments will show, our techniques, although including some approximations, 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 [CDD17] 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 [CKR18a], the AKISS [CCCK16a] and SPEC tool [TNH16] were already less efficient (by orders of magnitude) than the DeepSec tool before our current work.

Finally, our approach can also be compared to tools for an unbounded number of sessions. The ProVerif [BAF08], Tamarin [BDS15] and Maude-NPA [SEMM14] 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, represented by trace equivalence of processes in the applied-pi calculus [ABF18].

2.1 Messages and cryptography

To analyse protocols, we rely on symbolic models rooted in the seminal work of Dolev and Yao [DY81]. Cryptographic operations are modelled by a finite *signature*, i.e., a set of function symbols with their arity $\mathcal{F} = \{\mathrm{f}/n, \mathrm{g}/m, \ldots\}$. 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 models the classical primitives of pairs, randomised symmetric encryption, and their inverse operations:

$$\mathcal{F} = \{\ \langle\cdot,\cdot\rangle/2,\ \mathsf{proj}_1/1,\ \mathsf{proj}_2/1,\ \mathsf{senc}/3,\ \mathsf{sdec}/2\ \}$$

For example, let $m \in \mathcal{N}_{pub}$, and $k, r \in \mathcal{N}_{priv}$ modelling a private key and a random nonce, respectively. The term

$$c = \mathsf{senc}(m, r, k)$$

models a ciphertext obtained by encrypting m with the key k and randomness r, and $\mathsf{sdec}(c, k)$ models its decryption.△

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

$$\mathsf{proj}_i(\langle x_1, x_2\rangle) =_E x_i \qquad \mathsf{sdec}(\mathsf{senc}(x, y, z), z) =_E x$$

That is, a message encrypted with a key k can be recovered by decrypting using the same key k. Using the notations of Example 2.1, we can for instance derive from these equations that $\mathsf{sdec}(c, k) =_E m$.

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:

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

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 [ABF18]: 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 [CCCK16b, TNH16, CKR18a] 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 2.2. We describe a toy protocol that will serve as a running example throughout the paper. This is a simplification of the BAC protocol implemented in the European e-passports. The system builds upon the signature and equational theory introduced in Example 2.1. In a preliminary phase, a reader obtains the private key k of a passport, and then they communicate as follows (in Alice-Bob notation):

$$\begin{array}{rcll} \textit{Reader} \to \textit{Passport} & : & \textsc{get_challenge} & \\ \textit{Passport} \to \textit{Reader} & : & n & \\ \textit{Reader} \to \textit{Passport} & : & \mathsf{senc}(n, r, k) & \text{bound as } x \\ \textit{Passport} \to \textit{Reader} & : & \textsc{ok} & \text{if } \mathsf{sdec}(x, k) = n \\ & & \textsc{error} & \text{otherwise} \end{array}$$

where $n, r \in \mathcal{N}_{priv}$ and $\textsc{get_challenge}, \textsc{ok}, \textsc{error} \in \mathcal{N}_{pub}$. In particular, the passport triggers an error when it receives a communication originated from a reader that has not the right key k, i.e. a reader that has been paired with an other passport during the preliminary phase. In the applied pi-calculus, they are modelled by the following processes

$$\begin{aligned} P(k, n) &= c(x_0).\ \text{if } x_0 = \textsc{get_challenge} \text{ then} \\ &\quad\ \overline{c}\langle n\rangle.\ c(x). \\ &\quad\ \text{if } \mathsf{sdec}(x, k) = n \text{ then } \overline{c}\langle\textsc{ok}\rangle.\ 0 \\ &\quad\ \text{else } \overline{c}\langle\textsc{error}\rangle.\ 0 \\ R(k, r) &= \overline{c}\langle\textsc{get_challenge}\rangle. \\ &\quad\ c(x_n).\ \overline{c}\langle\mathsf{senc}(x_n, r, k)\rangle.\ 0 \end{aligned}$$

where $c \in Ch_{pub}$. △

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).

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

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

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

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 $P\!\downarrow$ the unique $\rightsquigarrow$-normal form of P.

The operational semantics then operates on *extended processes* $(\mathcal{P}, \Phi)$, 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 $\mathcal{AX}$ 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.

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

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

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 $\mathbb{T}(\{\!\{P\}\!\}, \varnothing)$. Indistinguishability against active adversaries will be modelled as relations between such sets of traces.

Example 2.3. Consider again the access-control protocol described in Example 2.2. Let $S = (\{\!\{P(k,n), R(k',r)\}\!\},\ \varnothing)$, with $k, k', n, r \in \mathcal{N}_{priv}$, a system consisting of a passport and a reader in parallel. The system has the following trace:

$$\begin{aligned}
S &\xrightarrow{\overline{c}\langle \mathsf{ax}_0\rangle} (\{\!\{P(k,n), R_0(k',r)\}\!\}, \Phi_0) \\
&\xrightarrow{c(\textsc{get_challenge})} (\{\!\{P_0(k,n), R_0(k',r)\}\!\},\ \Phi_0) \\
&\xrightarrow{\overline{c}\langle \mathsf{ax}_1\rangle} (\{\!\{P_1(k,n), R_0(k',r)\}\!\},\ \Phi_0 \cup \Phi_1) \\
&\xrightarrow{c(\mathsf{ax}_1)} (\{\!\{P_1(k,n), \overline{c}\langle \mathsf{senc}(n,r,k')\rangle\}\!\},\ \Phi_0 \cup \Phi_1) \\
&\xrightarrow{\overline{c}\langle \mathsf{ax}_2\rangle} (\{\!\{P_1(k,n), 0\}\!\},\ \Phi_0 \cup \Phi_1 \cup \Phi_2) \\
&\xrightarrow{c(\mathsf{ax}_2)} (\{\!\{\overline{c}\langle \alpha\rangle, 0\}\!\},\ \Phi_0 \cup \Phi_1 \cup \Phi_2)\})
\end{aligned}$$

with

$$\begin{aligned}
\Phi_0 &= \{\mathsf{ax}_0 \mapsto \textsc{get_challenge}\} \\
\Phi_1 &= \{\mathsf{ax}_1 \mapsto n\} \\
\Phi_2 &= \{\mathsf{ax}_2 \mapsto \mathsf{senc}(n,r,k')\} \\
P(k,n) &= c(x_0).\ \text{if } x_0 = \textsc{get_challenge} \text{ then } P_0(k,n) \\
P_0(k,n) &= \overline{c}\langle n\rangle.\ P_1(k,n) \\
R(k',r) &= \overline{c}\langle \textsc{get_challenge}\rangle.\ R_0(k',r)
\end{aligned}$$

and $\alpha = \textsc{ok}$ if $k = k'$, and $\alpha = \textsc{error}$ if $k \neq k'$. Note that the input action $c(\textsc{get_challenge})$ could be replaced by $c(\mathsf{ax}_0)$.$\triangle$

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.

$$
\begin{array}{llcll}
(\textsc{Out}) & (\{\!\{\overline{c}\langle u\rangle.P\}\!\} \cup \mathcal{P}, \Phi) & \xrightarrow{\overline{c}\langle \mathsf{ax}\rangle} & (\{\!\{P\}\!\} \cup \mathcal{P}, \Phi \cup \{\mathsf{ax} \mapsto u\}) & c \in Ch_{pub}, \mathsf{ax} \in \mathcal{AX} \setminus dom(\Phi) \\
(\textsc{In}) & (\{\!\{c(x).P\}\!\} \cup \mathcal{P}, \Phi) & \xrightarrow{c(\xi)} & (\{\!\{P[x \mapsto \xi\Phi]\}\!\} \cup \mathcal{P}, \Phi) & c \in Ch_{pub}, \xi \in \mathcal{T}(\mathcal{F}, \mathcal{N}_{pub} \cup dom(\Phi)) \\
(\textsc{Comm}) & (\{\!\{\overline{c}\langle u\rangle.P, c(x).Q\}\!\} \cup \mathcal{P}, \Phi) & \xrightarrow{\tau} & (\{\!\{P, Q[x \mapsto u]\}\!\} \cup \mathcal{P}, \Phi) & c \in Ch_{priv} \\
(\textsc{Par}) & (\{\!\{P_1 \mid \ldots \mid P_n\}\!\} \cup \mathcal{P}, \Phi) & \xrightarrow{\tau} & (\{\!\{P_1, \ldots, P_n\}\!\} \cup \mathcal{P}, \Phi) &
\end{array}
$$

Figure 2: Operational semantics of the applied pi-calculus

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 2.4. Consider again the model of passport and reader introduced in Example 2.2, and let us write

$$S(k, n, r) = P(k, n) \mid R(k, r)$$

a system consisting of a passport and a reader in parallel with a shared key k. The unlinkability property can be stated by the inability for the attacker to distinguish between two copies of the same passport interacting with readers, and two different passports. That is,

$$S(k, n, r) \mid S(k, n', r') \approx_{tr} S(k, n, r) \mid S(k', n', r')$$

with $k, k', n, n', r, r' \in \mathcal{N}_{priv}$ pairwise distinct. The inclusion $S(k, n, r) \mid S(k, n', r') \sqsubseteq_{tr} S(k, n, r) \mid S(k', n', r')$ indeed holds. However, this model of unlinkability is violated in that the converse inclusion does not hold. Indeed in the right-hand-side process, making a reader interact with the wrong passport produces an error message, which is not the case in the left-hand side (since the two systems share the same key). Formally, $\mathbb{T}(S(k, n, r) \mid S(k', n', r'))$ contains a trace t such that

$$
\begin{aligned}
tr(t) &= \tau\ \overline{c}\langle \mathsf{ax}_0\rangle\ c(\mathsf{ax}_0)\ \overline{c}\langle \mathsf{ax}_1\rangle\ c(\mathsf{ax}_1)\ \overline{c}\langle \mathsf{ax}_2\rangle\ c(\mathsf{ax}_2)\ \overline{c}\langle \mathsf{ax}_3\rangle \\
\Phi(t) &= \{\mathsf{ax}_0 \mapsto \textsc{get_challenge},\ \mathsf{ax}_1 \mapsto n, \\
&\qquad \mathsf{ax}_2 \mapsto \mathsf{senc}(n, r', k'),\ \mathsf{ax}_3 \mapsto \textsc{error}\}
\end{aligned}
$$

△

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 [CKR18a]. 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 [CD09, BDH15, CCCK16b, CKR18b]. It allows for partial-order reductions [BDH15], 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 [ABF18] as:

$$P_{fresh} \mid P_{fresh} \approx_{tr} Q_{fresh} \mid Q_{fresh}$$

where $P_{fresh} = \mathsf{new}\, e.\, \overline{c}\langle e\rangle.\, P(e)$, $Q_{fresh} = \mathsf{new}\, e.\, \overline{c}\langle e\rangle.\, Q(e)$. Such encodings however break determinacy and are thus incompatible with the partial-order reductions of [BDH15]. Our dedicated 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 to prove or disprove 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 3.1 . If $P \approx_s Q$ then $P \approx_{tr} Q$.

This is immediate as $t^2 \in \mathbb{T}(P, Q)$ entails $\mathsf{snd}(t^2) \in \mathbb{T}(Q)$.

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

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

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

Figure 3: Semantics on twin-processes

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: 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

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

with $a, b \in Ch_{pub}$ and $s \in Ch_{priv}$. These 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 trace equivalent. However the skeletons at toplevel constrain the session matchings, i.e. the application of rule (Match). Hence any trace executing an output on a or b is a false attack.

Finally false attacks cannot happen for determinate processes, i.e. the class of processes for which the partial-order reductions of [BDH15] were designed. A plain process P is determinate if it does not contain private channels and,

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

Proposition 3.2. If P, Q are determinate plain processes such that $P \approx_{tr} Q$ then $P \approx_s Q$.

The core argument is the uniqueness of 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. The proof can be found 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 [BAF05] conditional branchings were also required to result into the same control-flow. This condition has however been relaxed within [CB13]: the resulting diff-equivalence can be defined in our formalism as equivalence by session 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

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

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

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

This example is 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.

Relation to observational equivalence As a side result we also compare equivalence by session to observational equivalence, or more technically the equivalent notion of *labelled bisimilarity* as described in [ABF18, CKR18a]. Just as equivalence by session, this equivalence is known to be an intermediate refinement between diff-equivalence and trace equivalence [CD09]:

Lemma 3.3. $\approx_d \subseteq \approx_o \subseteq \approx_{tr}$. Besides, $\approx_o$ and $\approx_{tr}$ coincide for determinate processes.

In particular by Proposition 3.2 we obtain that trace equivalence, observational equivalence, and equivalence by session coincide for determinate processes. However they are incomparable in general:

Lemma 3.4. $\approx_s$ and $\approx_o$ are incomparable.

Proof. If we write $P = c(x).c(x)$ and $Q = c(x) \mid c(x)$, then $P \approx_o Q$ but $P \not\approx_s Q$. Besides, if $k_0, k_1, k_2 \in \mathcal{N}_{priv}$ we define

$$\begin{aligned} R(t_0,t_1,t_2) = \quad & \overline{c}\langle k_0\rangle \mid \overline{c}\langle k_1\rangle \mid \overline{c}\langle k_2\rangle \mid \\ & c(x).\ \text{if } x = k_0 \text{ then } \overline{c}\langle t_0\rangle \\ & \qquad \text{else if } x = k_1 \text{ then } \overline{c}\langle t_1\rangle \\ & \qquad \text{else if } x = k_2 \text{ then } \overline{c}\langle t_2\rangle \end{aligned}$$

If $a, b \in \mathcal{N}_{pub}$ distinct, we have $R(a,b,b) \approx_s R(b,a,a)$ but $R(a,b,b) \not\approx_o R(b,a,a)$. □

To sum up the relations between all equivalences:

Proposition 3.5. If $\approx \in \{\approx_o, \approx_s\}$ then $\approx_d \subsetneq \approx \subsetneq \approx_{tr}$ and, for determinate processes, $\approx = \approx_{tr}$.

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 later sections.

Definition 3.2. 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 universal and existantial optimisations in a modular way:

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 $(\mathbb{O}^\forall, \mathbb{O}^\exists_{\text{all}})$ (resp. $(\mathbb{O}^\forall_{\text{all}}, \mathbb{O}^\exists)$). 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 [BDH15], 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.

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 integer word 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:

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

the rules (In) and (Out) preserve labels:

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

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

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

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

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. Still, this is rather weak: for example the recipe $\mathsf{proj}_1(\langle n, \mathsf{ax}\rangle)$ is artificially dependent in the axiom ax, preventing optimisations. Such spurious dependencies can be erased using the following notion:

Definition 4.2 (recipe equivalence). Two input transitions

$$(\mathcal{P}, \Phi) \xrightarrow{[\,c(\xi_1)\,]^\ell} A \qquad\qquad (\mathcal{P}, \Phi) \xrightarrow{[\,c(\xi_2)\,]^\ell} A$$

are said *recipe equivalent* when $\xi_1\Phi =_E \xi_2\Phi$. Two traces are recipe equivalent if one can be obtained from the other by replacing some transitions by recipe-equivalent ones.

The rest of this section formalises the intuition that equivalence by session can be studied up to recipe-equivalent rewriting of traces, and arbitrary permutation of their independent actions. Proofs can be found in Appendix C.1.

Correctness of por techniques If $\mathsf{tr} = \alpha_1 \cdots \alpha_n$ and π is a permutation of $[\![1, n]\!]$, we write

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

This is an action of the group of permutations of $[\![1, n]\!]$ on action words of size n. We say that π *permutes independent actions* of tr if either $\pi = \mathrm{id}$, or $\pi = \pi_0 \circ (i\ \ i{+}1)$ with $\alpha_i \parallel \alpha_{i+1}$ and π_0 permutes independent actions of $(i\ \ i+1).\mathrm{tr}$. Such permutations preserve the group structure of permutations, in the sense of these two straightforward propositions:

Proposition 4.1 (composition). If π permutes independent actions of tr, and π' permutes independent actions of $\pi.\mathrm{tr}$, then $\pi' \circ \pi$ permutes independent actions of tr.

Proposition 4.2 (inversion). If π permutes independent actions of tr, then π^{-1} permutes independent actions of $\pi.\mathrm{tr}$.

We will use these two properties implicitly in many proofs. But more importantly, the action of permutations on trace words can be lifted to traces:

Proposition 4.3. If $t : A \overset{\mathrm{tr}}{\Longrightarrow} B$ and π permutes independent actions of tr, then $A \overset{\pi.\mathrm{tr}}{\Longrightarrow} B$. This trace is unique if we take labels into account, and will be referred as $\pi.t$.

Together with recipe equivalence, this is the core notion for defining partial-order reductions. We gather them into $\equiv_{\mathrm{por}}$ the smallest equivalence relation over traces containing recipe equivalence and such that $t \equiv_{\mathrm{por}} \pi.t$ when π permutes independent actions of t. The result below justifies that quotients by $\equiv_{\mathrm{por}}$ result in correct refinements.

Proposition 4.4 (correctness of por). 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 exists $t' \equiv_{\mathrm{por}} t_{ext}$, where t is a prefix of t_{ext} such that $t' \in \mathbb{O}^{\forall}_1$. 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) by exploiting Proposition 4.4. We formalise this idea by using reduction strategies based on *polarity* patterns.

Polarities and phases We assign polarities to processes depending on their toplevel actions: public inputs are positive (+1), public outputs and parallels are overwhelmingly negative ($-\infty$), and others are null.

$$\begin{array}{lll} polar(c(x).P) = 1 & polar(\overline{c}\langle u\rangle.P) = -\infty & c \in Ch_{pub} \\ polar(d(x).P) = 0 & polar(\overline{d}\langle u\rangle.P) = 0 & d \in Ch_{priv} \\ polar(0) = 0 & polar(P \mid Q) = -\infty & \end{array}$$

This notion is lifted to extended processes by summing:

$$polar((\mathcal{P}, \Phi)) = \sum_{R \in \mathcal{P}} polar(R)\,.$$

In particular, any extended process that contains an executable parallel operator or output has polarity $-\infty$, and executing public inputs makes polarity nonincreasing. We then identify the trace patterns at the core of our partial-order reductions. We say that a trace

$$t : A_0 \xrightarrow{[\,a_1\,]^{L_1}} \cdots \xrightarrow{[\,a_n\,]^{L_n}} A_n$$

- is a *negative phase* when all transitions are outputs or parallels, and $polar(A_n) \neq -\infty$.
- is a *null phase* when $polar(A_0) \geqslant 0$, $n = 1$ and the transition is an internal communication.
- is a *positive phase* when $polar(A_0) > 0$, all transitions are inputs, all L_i's are equal, and $polar(A_0) > polar(A_n)$.

Rephrasing, a negative phase executes all available outputs and parallels, a null phase is one internal communication, and a positive phase executes a whole chain of inputs. Note that only negative phases may be empty.

Basic compression The first optimisation is to only consider traces that can be decomposed into phases. Formally we write $\mathbb{O}^{\forall}_{\mathrm{c},b}$ the set of traces of the form

$$t : b_0^- \cdot b_1^+ \cdot b_1^- \cdot b_2^+ \cdot b_2^- \cdots b_n^+ \cdot b_n^-$$

where each b_i^+ is a positive or null phase, and each b_i^- is a negative phase. We show in Appendix C.3 that any maximal trace can be decomposed this way after application of a well-chosen permutation of independent actions. Hence the following result by Proposition 4.4:

Proposition 4.5. $\mathbb{O}^{\forall}_{\mathrm{c},b}$ is a correct refinement of $\mathbb{O}^{\forall}_{\mathrm{all}}$.

Determinism of negative phases Negative phases are non-deterministic by essence, but the underlying combinatorial explosion is artificial in that most of the actions within negative phases are independent: we show that they can actually be executed purely deterministically.

Let us fix an arbitrary total ordering $\preccurlyeq$ on labelled actions. A negative phase b^-, with $tr(b^-) = \alpha_1 \cdots \alpha_n$, is said *consistent* when for all $i < n$ such that $\alpha_i \parallel \alpha_{i+1}$, we have $\alpha_i \preccurlyeq \alpha_{i+1}$. We write $\mathbb{O}^{\forall}_{\mathrm{c}}$ the subset of $\mathbb{O}^{\forall}_{\mathrm{c},b}$ of traces whose negative phases are all consistent.

Proposition 4.6. $\mathbb{O}^{\forall}_{\mathrm{c}}$ is a correct refinement of $\mathbb{O}^{\forall}_{\mathrm{c},b}$.

Proof. By Proposition 4.4, it sufficies to prove that for all negative phases b^-, there exists π permuting independent actions of b^- such that $\pi.b^-$ is consistent. This follows from a well-founded induction on $tr(b^-)$ w.r.t the lexicographic extension of $\preccurlyeq$ on words of actions. □

4.3 Reduction optimisations

So far we compressed traces into sequences of phases. Now we show how independent phases can be reordered to reduce even further the complexity. These optimisations are inspired from the reduced semantics and improper blocks [BDH15], althought our work differs in its development.

Blocks A *block* is a positive or null phase followed by a negative phase. Any trace of $\mathbb{O}^{\forall}_{\mathsf{c}}$ is therefore composed of an initial negative phase and a sequence of blocks. Two blocks b and b' are said *independent*, written $b \parallel b'$, if all actions of the former are independent of all actions of the latter. Analogously to actions, we refer to permutations π permuting independent blocks of traces of $\mathbb{O}^{\forall}_{\mathsf{c}}$. All related notations and results can be cast to blocks thanks to the following straightforward proposition:

Proposition 4.7. Let $t : b_p \cdots b_n$ a sequence of blocks, $\mathsf{tr}_i = tr(b_i)$. If π permutes independent blocks of $\mathsf{tr} = tr(t)$, then there is π' permuting independent actions of tr s.t.

$$\pi'.\mathsf{tr} = \pi.\mathsf{tr} = \mathsf{tr}_{\pi(p)} \cdots \mathsf{tr}_{\pi(n)}.$$

Note in particular the corollary of Proposition 4.4 that will be at the core of the results of this section, where $\equiv_{\text{b-por}}$ is the analogue of $\equiv_{\text{por}}$ where permutation of independent actions is replaced by permutation of independent blocks:

Corollary 4.8. Let $\mathbb{O}^{\forall}_1 \subseteq \mathbb{O}^{\forall}_2 \subseteq \mathbb{O}^{\forall}_{\mathsf{c}}$. We assume that for all $t \in \mathbb{O}^{\forall}_2$, there exists $t' \equiv_{\text{b-por}} t$ such that $t' \in \mathbb{O}^{\forall}_1$. Then $\mathbb{O}^{\forall}_1$ is a correct refinement of $\mathbb{O}^{\forall}_2$.

Improper blocks Blocks may contain a negative phase that does not bring new knowledge to the attacker through public outputs. Such blocks can always be relegated to the end of traces, intuitively because they are not essential to execute other blocks. Formally, we say that a block

$$b : (\mathcal{P}, \Phi) \overset{\mathsf{tr}}{\Longrightarrow} (\mathcal{Q}, \Phi \cup \{\mathsf{ax}_1 \mapsto t_1, \ldots, \mathsf{ax}_n \mapsto t_n\})$$

is *improper* if

(1) all labels appearing in tr do not appear in $\mathcal{Q}$, except maybe on null processes; and

(2) for all $i \in [\![1, n]\!]$, t_i is deducible from Φ, that is, there exists a recipe ξ_i such that $\xi_i\Phi =_E t_i$.

This is more general than the improper blocks defined in [BDH15, CKR19], that require $n = 0$. Our finer optimisation captures for example outputs of public error codes in the model of the e-passport in Example 2.2 (ERROR, OK). We write $\mathbb{O}^{\forall}_{\mathsf{c+i}}$ the subset of $\mathbb{O}^{\forall}_{\mathsf{c}}$ of traces not containing an improper block followed by a proper block.

Proposition 4.9. $\mathbb{O}^{\forall}_{\mathsf{c+i}}$ is a correct refinement of $\mathbb{O}^{\forall}_{\mathsf{c}}$.

Proof. By Corollary 4.8, it sufficies to show that for all traces of $\mathbb{O}^{\forall}_{\mathsf{c}}$, there exists a recipe-equivalent trace whose improper blocks are independent of all blocks following them. By Item (2) of the definition, by replacing each occurrence of ax_i by ξ_i in all input actions, we obtain a recipe-equivalent trace whose improper blocks are data independent of all blocks following them. Sequential indendence is then justified by Item (1). □

Note that when restricting the definition to $n = 0$, we obtain a weaker optimisation than the one presented in [BDH15]. The latter indeed manages to additionally restrict to traces that contain at most one improper block. This, however, relies on determinate-specific arguments that are unsound for equivalence by session in general.

Lexicographic reduction Finally, as sequences of independent blocks can be permuted arbitrarily, we define an optimisation that fixes their order. Concretely we let $\preccurlyeq$ an ordering on blocks insensitive to recipes, and such that independent blocks are always strictly comparable. We define a predicate $\mathsf{Minimal}(t, b)$ that tells whether adding the block b at the end of t still results in a minimal trace w.r.t. the lexicographic extension of $\preccurlyeq$.

$\mathsf{Minimal}(b^-, b)$	b^- negative phase
$\mathsf{Minimal}(b_1 \cdots b_n, b)$	if $\neg(b_n \parallel b)$
$\mathsf{Minimal}(b_1 \cdots b_n, b)$	if $b_n \prec b$ and $\mathsf{Minimal}(b_1 \cdots b_{n-1}, b)$

We say that b is *allowed after* t if $\mathsf{Minimal}(t, b')$ for all b' recipe equivalent to b. This strengthens the optimisation by discarding spurious data dependencies. Then $\mathbb{O}^{\forall}_{\mathsf{c+r}} \subseteq \mathbb{O}^{\forall}_{\mathsf{c}}$ is defined by the following inference rules.

$$\frac{b^- \text{ negative phase}}{b^- \in \mathbb{O}^{\forall}_{\mathsf{c+r}}} \qquad \frac{t \in \mathbb{O}^{\forall}_{\mathsf{c+r}} \qquad b \text{ allowed after } t}{t \cdot b \in \mathbb{O}^{\forall}_{\mathsf{c+r}}}$$

To account for improper blocks, we write $\mathbb{O}^{\forall}_{\text{por}}$ the set of traces of the form $t : b^- \cdot t_p \cdot t_i$, where b^- is a negative phase, $t_p \in \mathbb{O}^{\forall}_{\mathsf{c+r}}$ only contains proper blocks, and $t_i \in \mathbb{O}^{\forall}_{\mathsf{c+r}}$ only contains improper blocks. The correctness of this optimisation relies on Corollary 4.8 and is proved in Appendix C.4.

Proposition 4.10. $\mathbb{O}^{\forall}_{\text{por}}$ is a correct refinement of $\mathbb{O}^{\forall}_{\mathsf{c+i}}$.

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, namely the group of all permutations on $[\![1, n]\!]$. Then we denote by $\vec{P}$ and $\pi.\vec{P}$ the tuples of plain processes

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

We assume $\equiv$ an equivalence relation on tuples of processes that is stable under the action of permutations, i.e. for all tuples $\vec{P}, \vec{Q}$ of size n and $\pi \in S_n$

$$\vec{P} \equiv \vec{Q} \quad \Rightarrow \quad \pi.\vec{P} \equiv \pi.\vec{Q} \,. \tag{1}$$

Process redundancy w.r.t. $\equiv$ is then simply captured by 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 stabiliser, 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 the permutations $(1\ \ 2)$, $(3\ \ 4)$ and $(3\ \ 5)$. △

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

Proof. Consider the function $(\pi, \vec{P}) \mapsto \pi.\vec{P}$. It is a group action of S_n on the set of tuples quotiented by the equivalence relation $\equiv$, thanks to Equation (1). $\mathsf{Stab}_{\equiv}(\vec{P})$ is a stabiliser of this action, hence the conclusion. □

This formalisation takes root in classical work in model-checking formalising the symmetries of systems by the group of their automorphisms [ES96]. Our optimisations consist of identifying suitable equivalence relations $\equiv$ and refining the trace space based on the analysis of stabilisers.

5.2 Structural equivalence

We exhibit an equivalence identifying processes that have an identical structure (up to associativity and commutativity of parallel operators) and whose data are equivalent w.r.t. the equational theory and alpha-renaming of private names. This will be the basis of our symmetry-based refinements.

We define *structural equivalence* $\equiv$ on plain processes as the smallest equivalence 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 (that is, composition of equivalent processes with either a same process in parallel, or an input, output, or conditional instruction at toplevel). To account for the equational theory, we extend it to

$$\frac{\sigma, \sigma' \text{ substitutions} \qquad P\sigma \equiv Q\sigma' \qquad \forall x \in \mathcal{X}, x\sigma =_E x\sigma'}{P\sigma \equiv_E Q\sigma'}$$

Besides we add alpha equivalence of private names: intuitively, two agents executing the same protocol are behaving similarly even though they use their own session nonces. Formally if A is an extended process we define the relation $\equiv_\alpha^A$ on tuples of processes by

$$\frac{\forall i, P_i \equiv_E Q_i \qquad \varrho : \mathcal{N}_{priv} \to \mathcal{N}_{priv} \text{ bij.} \qquad \varrho_{|names(A)} = \mathsf{id}}{\langle P_1, \ldots, P_n \rangle \equiv_\alpha^A \langle Q_1, \ldots, Q_n \rangle \varrho}$$

That is, only names outside of A (frame included) may be renamed. To conclude, it is straightforward that this relation satisfies the requirements of Section 5.1, i.e.:

Proposition 5.2. For all extended process A, $\equiv_\alpha^A$ is an equivalence relation stable under the action of permutations (in the sense of Equation (1)).

5.3 Universal symmetry optimisation

We first present a universal optimisation, i.e. a refinement of $\mathbb{O}^{\forall}_{\mathsf{por}}$. It captures the idea that, when considering the traces of several parallel protocol sessions, starting the trace by an action from one session or an other does not make a substantial difference. To formalise this idea, let us consider a compressed trace

$$t : [\,P\,]^{\varepsilon} \overset{\mathsf{tr}}{\Longrightarrow} (\{\!\{[\,P_i\,]^{\ell_i}\}\!\}_{i=1}^{n}, \Phi_0) \qquad \in \mathbb{O}^{\forall}_{\mathsf{c}}$$

The goal is to exhibit conditions discarding some potential positive phases following t. Technically speaking, we require that the symmetries observed in t are reflected in one way or an other in $\mathbb{T}(P, Q)$. In the technical formalisation of these symmetries, we refer to traces $t^2 \in \mathbb{T}(P, Q)$ such that $\mathsf{fst}(t^2) = t$ using the following notations:

$$t^2 : ([\,P\,]^{\varepsilon}, Q) \overset{\mathsf{tr}}{\Longrightarrow} (\{\!\{([\,P_i\,]^{\ell_i}, Q_i)\}\!\}, \Phi_0, \Phi_1) \tag{2}$$

Homogeneous symmetry We first define a notion of symmetry within P that is reflected in the matching with Q. If $a, b \in [\![1, n]\!]$ and $\pi = (a\ \ b)$, we write $a \longleftrightarrow_1 b$ when

$$\pi \in \mathsf{Stab}_{\equiv_\alpha^{\Phi_0}}(\langle P_1, \ldots, P_n \rangle)$$

and for all traces t^2 verifying the hypotheses and notations of Equation (2), there exists a trace of the form

$$([\,P\,]^{\epsilon}, Q) \overset{\mathsf{tr}}{\Rightarrow} (\{\!\{([\,P_i\,]^{\ell_i}, Q_{\pi(i)})\}\!\}_{i=1}^{n}, \Phi_0, \Phi_1)\,.$$

Intuitively, the first condition expresses that P_a and P_b have the same traces, and the second ensures that they can be matched by the same sessions of Q. In particular for proving the equivalence by session of P and Q, executing a block starting in P_a or P_b results into a similar analysis.

Heterogeneous symmetry We now define a notion of symmetry capturing redundancy occurring at the same time in P and Q. If $a, b \in [\![1, n]\!]$ and $\pi = (a\ \ b)$, we write $a \longleftrightarrow_2 b$ when there exists ϱ permutation of Ch_{pub} such that

$$\pi.\langle P_1, \ldots, P_{a-1}, P_a\varrho, P_{a+1}, \ldots, P_b\varrho, \ldots, P_n\rangle \equiv_{\alpha}^{\Phi_0} \vec{P}$$

and for all traces t^2 verifying the hypotheses and notations of Equation (2), we have

$$\pi.\langle Q_1, \ldots, Q_{a-1}, Q_a\varrho, Q_{a+1}, \ldots, Q_b\varrho, \ldots, Q_n\rangle \equiv_{\alpha}^{\Phi_1} \vec{Q}$$

These two conditions express symmetries up to channel renaming. Indeed public channels do not interfere with the data flow of traces and, therefore, two processes that are structurally-equivalent up to bijective renaming of public channels still have a similar execution flow.

Reduction by symmetry All in all, we model symmetries in equivalence proofs by $\longleftrightarrow$ the smallest equivalence relation containing $\longleftrightarrow_1$ and $\longleftrightarrow_2$. The idea of our optimisation is then, when choosing a positive phase to execute after t, to consider only one input per equivalence class of $\longleftrightarrow$.

This representant should however be picked carefully to avoid interference with the lexicographic reduction introduced in Section 4.3. For that we refer again to the ordering on blocks $\preccurlyeq$ introduced in that section. This ordering can be lifted to a total ordering on the set

$$\{i \in [\![1, n]\!] \mid polar(P_i) > 0\}$$

by writing $i \preccurlyeq j$ when $b_i \preccurlyeq b_j$ for b_i and b_j arbitrary blocks starting by an action labelled, respectively, ℓ_i and ℓ_j. This cast is indeed well defined and total thanks to the assumptions that the original ordering on blocks is insensitive to recipes, and always relate independent blocks.

We thus qualify an input transition on process P_a following the trace t as *well-formed* if a is minimal w.r.t. $\preccurlyeq$ within its equivalence class for $\longleftrightarrow$. A trace is well-formed when all such transitions are, and we define the optimisation $\mathbb{O}^{\forall}_{\mathsf{sym}}$ as the set of well-formed traces of $\mathbb{O}^{\forall}_{\mathsf{c}}$. The correctness of this refinement is proved in Appendix D.1.

Proposition 5.3. $\mathbb{O}^{\forall}_{\mathsf{por}} \cap \mathbb{O}^{\forall}_{\mathsf{sym}}$ is a correct refinement of $\mathbb{O}^{\forall}_{\mathsf{por}}$.

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 then we do not need to consider redundant matchings. For instance, suppose that we need to match $P_1 \mid P_2$ with $Q \mid Q$. Just considering the identity permutation would be sufficient, and the permutation $(1\ \ 2)$ should be considered as redundant. Formally, let us consider an instance of the rule (Match)

$$(\mathcal{P}^2 \cup \{\!\{(P, Q)\}\!\}, \Phi_0, \Phi_1) \xrightarrow{\tau}_s (\mathcal{P}^2 \cup \{\!\{(P_i, Q_{\pi(i)})\}\!\}_{i=1}^{n}, \Phi_0, \Phi_1) \tag{3}$$

with $P = P_1 \mid \cdots \mid P_n$ and $Q = Q_1 \mid \cdots \mid Q_n$. We let $A = (\mathsf{snd}(\mathcal{P}^2), \Phi_0)$, and define the relation on permutations

$$\pi \sim \pi' \quad \textit{iff} \quad \exists u \in \mathsf{Stab}_{\equiv_{\alpha}^{A}}(\vec{Q}),\ \pi' = \pi \circ u\,.$$

Proposition 5.4. $\sim$ is an equivalence relation.

Proof. This essentially follows from Proposition 5.1, i.e. from the fact that $\mathsf{Stab}_{\equiv_{\alpha}^{A}}(\vec{Q})$ is a group. Reflexivity: a group of permutations contains the identity; symmetry: a group is closed by inverse; transitivity: a group is closed by composition. □

Let us say that an instance of Equation (3) is *well-formed* when π is minimal within its equivalence class for $\sim$, w.r.t. an arbitrary total ordering on permutations. We denote by $\mathbb{O}^{\exists}_{\mathsf{sym}}$ the set of traces of extended twin-processes whose instances of (3) are all well-formed. The correctness of this optimisation is stated below and proved in Appendix D.2.

Proposition 5.5. $\mathbb{O}^{\exists}_{\mathsf{sym}}$ is a correct refinement of $\mathbb{O}^{\exists}_{\mathsf{all}}$.

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 [CKR18b] and its underlying theory [CKR18a], 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 ... else ..., is represented by *equations* $u =^? v$ and *disequations* $u \neq^? v$.

To represent infinitely many processes, [CKR18a] 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 consider again the simplified model of BAC of Example 2.2. When executing the passport process $P(k, n)$ until reaching the success token ok, the constraints aggregate as

$$C = X_0 \vdash^? x_0 \ \wedge\ x_0 =^? \texttt{GET_CHALLENGE} \ \wedge \\ X \vdash^? x \ \wedge\ \mathsf{sdec}(x, k) =^? n$$

Intuitively the internal constraint solver will gradually deduce that solutions to this constraint need to map x to a term of the form $\mathsf{senc}(y_1, y_2, y_3)$, and will add the equations $y_1 =^? n$ and $y_3 =^? k$. △

Partition tree To decide trace equivalence between two plain 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

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

In the root of the partition tree, P and Q will be labeled by 0, i.e. the root will contain the two symbolic processes

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

There is only a single bijection between their labels, i.e. the identity $0 \mapsto 0$. Upon receiving this initial node, DeepSec applies the symbolic transition corresponding to our rule (Par), hence generating the two symbolic processes

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

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

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

These bijections are kept within the node of the partition tree and updated along side the other transformation rules of DeepSec. For obvious performance reasons, we cannot represent them by a naive enumeration of all process permutations. Fortunately, the skeleton requirement ensures

an invariant that the set S of session matchings between two processes A and B is always of the form

$$S = \{\pi \mid \forall i, \forall \ell \in C_i, \pi(\ell) \in D_i\}$$

where the sets $C_1, \ldots, C_n$ form a partition of the labels of A and $D_1, \ldots, D_n$ a partition of the labels of B. In particular, S can succinctly be stored as a simple association list of equivalence classes.

Decision of equivalence Finally, as our trace refinements depend on two sets $\mathbb{O}^\forall$ and $\mathbb{O}^\exists$, we annotate each symbolic process in the node by $\forall$, $\exists$ or $\forall\exists$ tags. They mark whether the trace from the root of the partition tree to the tagged process is determined to be in $\mathbb{O}^\forall$, $\mathbb{O}^\exists$ or both respectively. For instance, the two initial symbolic processes in the root of the partition tree are labeled by $\forall\exists$. We also provide a decision procedure for inclusion by session $\sqsubseteq_s$ that consists of tagging one of the initial processes as $\forall$ and the other one as $\exists$.

The 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 non-input actions in priority. Therefore on a process $\overline{c}\langle u\rangle.P \mid c(x).Q$, we prevent DeepSec from generating a node corresponding to the execution of the input due to the presence of the output.

The integration of other optimisations is more technical in a symbolic setting, in particular the *lexicographic reduction* $\mathbb{O}^\forall_{\mathsf{c+r}}$ described in Section 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 definition 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, it is correct to only partially implement such optimisations.

7 Experiments

In practice Based on the high-level description of the previous section, we extended the implementation of DeepSec to decide equivalence by session of P and Q. Upon completing an analysis, two cases can arise:

(1) The two processes are proved equivalent by session. Then they are also trace equivalent by Proposition 3.1.

(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 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 actions of this false attack could lead to a true attack. For instance, this heuristic allowed us to disprove trace equivalence in some analyses of $n \geqslant 3$ sessions of BAC. When our heuristic cannot discover a true attack, the result is not conclusive: the processes may well be trace equivalent or not. We leave 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;
- 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 (number of parallel processes that the model file exhibits).

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 BAC We modelled the simplified analysis of unlinkability in the BAC protocol described in Examples 2.2 and 2.4 as a simple example to compare our prototype and DeepSec in terms of scope and efficiency. For that we gather several variants of the analysis:

- for 2 sessions: both DeepSec and our prototype are able to find an attack trace (the same trace violates trace equivalence and equivalence by session).
- for 3 sessions: DeepSec times out and our prototype finds a false attack. This is due to the fact that, by executing outputs in priority (recall our por), more intermediate actions are available to match the trace. However our simple heuristic manages to reconstruct a true attack trace by delaying some output actions.
- we also consider a variant of the analysis where we remove the GET_CHALLENGE messages from the protocol. Our prototype now finds a false attack and the heuristic reconstructs a true attack for 2 sessions, but fails to conclude for 3 sessions.

BAC We also studied a more realistic model of BAC, accounting for more involved messages. The baseline version of DeepSec still fails to analyse 3 or more sessions, while our prototype reaches up to 5 sessions. On one side of the equivalence 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 We also consider the Helios protocol for electronic voting [Adi08]. 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 [CS13]. Vote privacy is formalised as in our running example, using a vote-swapping model, that is, we want to prove the equivalence of two situations where two honest votes have been exchanged.

A reduction result of Arapinis et al. [ACK16] ensures that, for such models, 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 [CKR18a]. However, when revoting is allowed, as it is the case for Helios, one needs to consider all scenarios when the tally accepts 7 ballots. In particular, it is not sufficient to consider only re-votes 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 fails to prove after 12h of computation any scenarios where more than one honest revote is emitted. In [ACK16] 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 [ACK16] 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 [BCG+15]. 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.

About the modelling of mixnets. The version of Helios we analyse relies on a mixnet, which 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 this can be performed using two models (where $c \in Ch_{priv}$):

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

Protocol	scenario	# roles	DeepSec baseline		DeepSec eq. by session	
Toy BAC	2 identical	4	↯	<1s	↯	<1s
	2 identical + 1 fresh	6		⏱	↯	<1s
Toy BAC no GET_CHALLENGE	2 identical	4	↯	<1s	↯	<1s
	2 identical + 1 fresh	6		⏱	✗	<1s
BAC	1 identical + 1 fresh	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

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.

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 [CGT18]. We also studied anonymity in a model of the AKA protocol deployed in 3G telephony networks [AMR+12] (without XOR), presented in the previous version of DeepSec [CKR18a].

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 the 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 has already been shown to satisfy equivalence.

References

[ABF18] Martín Abadi, Bruno Blanchet, and Cédric Fournet. The applied pi calculus: Mobile values, new names, and secure communication. *J. ACM*, 2018.

[ACK16] Myrto Arapinis, Véronique Cortier, and Steve Kremer. When are three voters enough for privacy properties? In *European Symposium on Research in Computer Security (ESORICS)*, 2016.

[Adi08] B. Adida. Helios: web-based open-audit voting. In *Conference on Security symposium (SS)*, 2008.

[AMR+12] M. Arapinis, L. Mancini, E. Ritter, M. Ryan, N. Golde, K. Redon, and R. Borgaonkar. New privacy issues in mobile telephony: fix and verification. In *ACM Conference on Computer and Communications Security (CCS)*, 2012.

[BAF05] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated verification of selected equivalences for security protocols. In *20th IEEE Symposium on Logic in Computer Science (LICS)*, 2005.

[BAF08] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated verification of selected equivalences for security protocols. *J. Log. Algebr. Program.*, 2008.

[BBK17] Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. Verified models and reference implementations for the TLS 1.3 standard candidate. In *IEEE Symposium on Security and Privacy (S&P)*, 2017.

[BCG+15] David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. A comprehensive analysis of game-based ballot privacy definitions. In *IEEE Symposium on Security and Privacy (S&P)*, 2015.

[BDH14] David Baelde, Stéphanie Delaune, and Lucca Hirschi. A reduced semantics for deciding trace equivalence using constraint systems. In *International Conference on Principles of Security and Trust (POST)*, 2014.

[BDH15] David Baelde, Stéphanie Delaune, and Lucca Hirschi. Partial order reduction for security protocols. *International Conference on Concurrency Theory (CONCUR)*, 2015.

[BDH18a] David Baelde, Stéphanie Delaune, and Lucca Hirschi. POR for security protocol equivalences - beyond action determinism. In *European Symposium on Research in Computer Security (ESORICS)*, 2018.

[BDH+18b] David A. Basin, Jannik Dreier, Lucca Hirschi, Sasa Radomirovic, Ralf Sasse, and Vincent Stettler. A formal analysis of 5g authentication. In *ACM Conference on Computer and Communications Security (CCS)*, 2018.

[BDS15] David A. Basin, Jannik Dreier, and Ralf Sasse. Automated symbolic proofs of observational equivalence. In *ACM Conference on Computer and Communications Security (CCS)*, 2015.

[CB13] Vincent Cheval and Bruno Blanchet. Proving more observational equivalences with proverif. In *Proceedings of the 2nd International Conference on Principles of Security and Trust (POST'13)*, 2013.

[CCCK16a] Rohit Chadha, Vincent Cheval, Ştefan Ciobâcă, and Steve Kremer. Automated verification of equivalence properties of cryptographic protocol. *ACM Transactions on Computational Logic*, 2016.

[CCCK16b] Rohit Chadha, Vincent Cheval, Ştefan Ciobâcă, and Steve Kremer. Automated verification of equivalence properties of cryptographic protocols. *ACM Transactions on Computational Logic*, 2016.

[CD09] Véronique Cortier and Stéphanie Delaune. A method for proving observational equivalence. In *IEEE Computer Security Foundations Symposium (CSF)*, 2009.

[CDD17] Véronique Cortier, Stéphanie Delaune, and Antoine Dallon. Sat-equiv: an efficient tool for equivalence properties. In *IEEE Computer Security Foundations Symposium (CSF)*, 2017.

[CGT18] Véronique Cortier, David Galindo, and Mathieu Turuani. A formal analysis of the neuchâtel e-voting protocol. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2018.

[CHH+17] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. A comprehensive symbolic analysis of TLS 1.3. In *ACM Conference on Computer and Communications Security (CCS)*, 2017.

[CJM03] Edmund M. Clarke, Somesh Jha, and Wilfredo R. Marrero. Efficient verification of security protocols using partial-order reductions. *STTT*, 2003.

[CKR18a] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. DEEPSEC: Deciding Equivalence Properties in Security Protocols – Theory and Practice. In *IEEE Symposium on Security and Privacy (S&P)*, 2018.

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

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

[CS13] Véronique Cortier and Ben Smyth. Attacking and fixing helios: An analysis of ballot secrecy. *Journal of*

Computer Security, 2013.

[DY81] D. Dolev and A.C. Yao. On the security of public key protocols. In *Symposium on Foundations of Computer Science (FOCS)*, 1981.

[ES96] E Allen Emerson and A Prasad Sistla. Symmetry and model checking. *Formal methods in system design*, 1996.

[For04] PKI Task Force. PKI for machine readable travel documents offering ICC read-only access. Technical report, International Civil Aviation Organization, 2004.

[MVB10] Sebastian Mödersheim, Luca Viganò, and David A. Basin. Constraint differentiation: Search-space reduction for the constraint-based analysis of security protocols. *Journal of Computer Security*, 2010.

[SEMM14] Sonia Santiago, Santiago Escobar, Catherine Meadows, and José Meseguer. A formal definition of protocol indistinguishability and its verification using Maude-NPA. In *International Workshop on Security and Trust Management (STM)*, 2014.

[TNH16] Alwen Tiu, Nam Nguyen, and Ross Horne. SPEC: an equivalence checker for security protocols. In *Asian Symposium on Programming Languages and Systems (APLAS)*, 2016.

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;
- we use it in Appendix B for proving the completeness of equivalence by session for determinate processes.

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.1 . 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.2. Let P, Q be plain processes in $\rightsquigarrow$-normal form such that $\mathsf{skel}(P) = \mathsf{skel}(Q)$. The following points are equivalent:

(1) $P \sqsubseteq_s Q$

(2) for all $t \in \mathbb{T}(P)$, there exist $t' \in \mathbb{T}(Q)$ and a session matching for t and t', such that $tr(t) = tr(t')$ (labels removed) and $\Phi(t) \sim \Phi(t')$

B Proofs of Section 3

In this section we give a detailed proof of the proposition:

Proposition 3.2. If P, Q are determinate plain processes such that $P \approx_{tr} Q$ then $P \approx_s Q$.

In the proof, by slight abuse of notation, we may say that an extended process is determinate. We also cast the notion of skeleton to extended processes by writing

$$\mathsf{skel}((\mathcal{P}, \Phi)) = \mathsf{skel}(\mathcal{P}) = \bigcup_{P \in \mathcal{P}} \mathsf{skel}(P)\,,$$

and to traces with

$$\mathsf{skel}(A_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} A_n) = \mathsf{skel}(A_0) \cdot \mathsf{skel}(A_1) \cdot \ldots \cdot \mathsf{skel}(A_n)\,.$$

That is, the skeleton of a trace is the sequence of the skeletons of the processes of which it is composed. Thus, if

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

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.1. 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.2. We say that $A = (\{\!\{P_1, \ldots, P_n\}\!\}, \Phi)$ 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.2. 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,

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 due to the $\rightsquigarrow$-normalisation, we know that neither of A_1 nor B_1 contain a parall operator, hence the result.

▹ *case 2:* $\alpha_1 \neq \tau$.

By definition $\mathbb{T}_\tau(B_0)$, we know that the rule (Par) is not applicable to B_0; neither to A_0 by $\mathsf{Inv}(A_0, B_0)$ Item (iv), which means that traces of $\mathbb{T}_\tau(B_0)$ may start by an application of rules (In) or (Out). Using this and the fact that $\mathsf{skel}(A_0) = \mathsf{skel}(B_0)$ ($\mathsf{Inv}(A_0, B_0)$ Item (ii)), we obtain that there exists a transition $A_0 \xrightarrow{\alpha_1} A_1$. The conclusion can now follow from the induction hypothesis applied to A_1, B_1; but to apply it we have to prove that $\varphi(A_1, B_1)$ and $\mathsf{Inv}(A_1, B_1)$ hold.

→ *proof that* $\varphi(A_1, B_1)$.

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

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

(i) A_0 and B_0 are determinate and determinacy is preserved by transitions.
(ii) By applying $\varphi(A_0, B_0)$ with the trace $t_0 : A_0 \xrightarrow{\alpha_1} A_1$, we obtain a trace $t_0' : B_0 \xrightarrow{\alpha_1} B_1'$ such that $\mathsf{skel}(A_1) = \mathsf{skel}(B_1')$. But by determinacy of B_0, the transition $B_0 \xrightarrow{\alpha_1} B_1$ is the only transition from B_0 that has label α_1, hence $B_1 = B_1'$ and the conclusion.
(iii) Identical proof as that of Item (ii) above, using the fact that $A_1 \sim B_1'$ instead of $\mathsf{skel}(A_1) = \mathsf{skel}(B_1')$.
(iv) Let us write

$$A_0 = (\{\!\{P_0\}\!\} \cup \mathcal{P}, \Phi) \qquad A_1 = (\{\!\{P_1\}\!\} \cup \mathcal{P}, \Phi')$$
$$B_0 = (\{\!\{Q_0\}\!\} \cup \mathcal{Q}, \Psi) \qquad B_1 = (\{\!\{Q_1\}\!\} \cup \mathcal{Q}, \Psi')$$

As we argued already at the beginning of *case 2*, neither $\mathcal{P}$ nor $\mathcal{Q}$ contain processes with parallel operators at their roots. Therefore, we only have to prove that P_1 has a parallel operator at its root *iff* Q_1 does. For cardinality reasons, this a direct corollary of the following points:
- $\mathsf{skel}(P_0) = \mathsf{skel}(Q_0)$ (same action α_1 being executable at topelevel),
- $\mathsf{skel}(A_0) = \mathsf{skel}(B_0)$ (hypothesis $\mathsf{Inv}(A_0, B_0)$), and
- $\mathsf{skel}(A_1) = \mathsf{skel}(B_1)$ (Item (ii) proved above). □

Proof of (1)⇒(2). The proof will follow in the steps as the other implication (we construct the trace t' by induction on the length of t while maintaining the invariant Inv).

More formally, we prove that for all extended processes A_0, B_0 such that $A_0 \approx_{tr} B_0$ 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 $s \in \mathsf{skel}(A_1) \setminus \mathsf{skel}(B_1)$. 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.2 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.3 . 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

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

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

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

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.2 to B.3 justify the following corollary—that actually appears to be stronger than the expected Proposition 3.2.

Corollary B.4. If P and Q are determinate plain processes in $\rightsquigarrow$-normal form, $P \approx_{tr} Q$ *iff* $P \sqsubseteq_s Q$.

C Proofs of Section 4

C.1 Permutability of independent actions

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 4.3 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 $\mathsf{ax} \notin \mathit{dom}(\Phi)$ 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 $\mathsf{ax} \neq \mathsf{ax}'$, $\{\mathsf{ax}, \mathsf{ax}'\} \cap \mathit{dom}(\Phi) = \varnothing$. 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 4.3 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 $\mathit{tr}(u)$ (labels included). We then construct $\pi.t$ by induction on the hypothesis that π permutes independent actions of $\mathit{tr}(t)$. Let us write

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

If $\pi = \mathsf{id}$ it sufficies to choose $\pi.t = t$. Otherwise let us write $\pi = \pi_0 \circ (i \;\; i+1)$ with $\alpha_i \parallel \alpha_{i+1}$ and π_0 permutes independent actions of $\mathsf{tr}' = \alpha_p \cdots \alpha_{i-1}\alpha_{i+1}\alpha_i\alpha_{i+2} \cdots \alpha_n$. By Proposition C.1, there exists a trace

$$u : A_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_{i-1}} A_{i-1} \overset{\alpha_{i+1}\alpha_i}{\Longrightarrow} A_{i+1} \xrightarrow{\alpha_{i+2}} \cdots \xrightarrow{\alpha_n} A_n$$

such that for all $u^2 : A^2 \overset{\mathsf{tr}'}{\Longrightarrow}_s B^2$ verifying $\mathsf{fst}(u^2) = u$, there exists $t^2 : A^2 \overset{\mathsf{tr}}{\Longrightarrow}_s B^2$ such that $\mathsf{fst}(t^2) = t$. Then since π_0 permutes independent actions of $\mathsf{tr}' = tr(u)$, it sufficies to choose $\pi.t = \pi_0.u$ by induction hypothesis. □

Then we can easily extend this result to $\equiv_{\mathsf{por}}$.

Proposition C.3. Let $t : A \overset{\mathsf{tr}}{\Longrightarrow} B$ be a trace and $t' \equiv_{\mathsf{por}} t$. Then writing $tr(t') = \mathsf{tr}'$ we have $t' : A \overset{\mathsf{tr}'}{\Longrightarrow} B$ and, for all $u^2 : A^2 \overset{\mathsf{tr}'}{\Longrightarrow} B^2$ such that $t' = \mathsf{fst}(u^2) \sim \mathsf{snd}(t^2)$, there exists $t^2 : A^2 \overset{\mathsf{tr}}{\Longrightarrow} B^2$ such that $t = \mathsf{fst}(t^2) \sim \mathsf{snd}(t^2)$.

Proof. For the sake of reference, let us write $H(t, t')$ the property to prove. We reason by induction on the hypothesis $t \equiv_{\mathsf{por}} t'$.

- *case 1*: $t' = \pi.t$, π permutes independent actions of t.

  Direct consequence of Proposition C.2.

- *case 2*: t' is recipe-equivalent to t.

  Let u^2 with $t' = \mathsf{fst}(u^2) \sim \mathsf{snd}(u^2)$. By static equivence, for any recipes ξ_1, ξ_2 such that

$$\xi_1\Phi(\mathsf{fst}(u^2)) =_E \xi_2\Phi(\mathsf{fst}(u^2)),$$

  we also have

$$\xi_1\Phi(\mathsf{snd}(u^2)) =_E \xi_2\Phi(\mathsf{snd}(u^2)).$$

  In particular, t^2 can be obtained by operating on the second component of u^2 the same recipe transformations that have been operated to transform $t' = \mathsf{fst}(u^2)$ into t.

- *case 3*: (transitivity) $H(t, s)$ and $H(s, t')$ for some trace s.

  Let u^2 with $t' = \mathsf{fst}(u^2) \sim \mathsf{snd}(u^2)$. By hypothesis $H(s, t')$ there exists s^2 such that $s = \mathsf{fst}(s^2) \sim \mathsf{snd}(s^2)$. Hence the result by hypothesis $H(t, s)$. □

And finally we have the Proposition 4.4 that is a corollary of this result.

Proposition 4.4 (correctness of por). Let $\mathbb{O}_1^\forall \subseteq \mathbb{O}_2^\forall$ be universal optimisations. We assume that for all $t \in \mathbb{O}_2^\forall$, there exists $t' \equiv_{\mathsf{por}} t_{ext}$, where t is a prefix of t_{ext} such that $t' \in \mathbb{O}_1^\forall$. Then $\mathbb{O}_1^\forall$ is a correct refinement of $\mathbb{O}_2^\forall$.

Proof. Let $\approx_i = \sqsubseteq_i \cap \sqsupseteq_i$ the notion of equivalence induced by $\mathbb{O}_i^\forall$. The inclusion $\approx_2 \subseteq \approx_1$ is immediate. Let us then assume $P \sqsubseteq_1 Q$ and prove $P \sqsubseteq_2 Q$. Let $t \in \mathbb{T}(P) \cap \mathbb{O}_2^\forall$. Without loss of generality, we assume t maximal, i.e. that there are no transitions possible from its last process. Therefore by hypothesis, there exists $t' \equiv_{\mathsf{por}} t$ such that $t' \in \mathbb{O}_1^\forall$. Since $P \sqsubseteq_1 Q$, there is $u^2 \in \mathbb{T}(P, Q)$ such that

$$t' = \mathsf{fst}(u^2) \sim \mathsf{snd}(u^2).$$

Therefore by Proposition C.3, there exists $t^2 \in \mathbb{T}(P, Q)$ such that $t = \mathsf{fst}(t^2) \sim \mathsf{snd}(t^2)$. □

C.2 Additional results

We provide some utility results on independent permutations of actions. First, about composition of permutations:

Proposition C.4. Let t be a trace, π permuting independent actions of t, and π' permuting independent actions of $\pi.t$. Then $\pi.\pi'.t = (\pi \circ \pi').t$.

Proof. By definition, if $t : A \overset{\mathsf{tr}}{\Longrightarrow} B$, $\pi.\pi'.t$ is the unique trace of the form $A \overset{\pi.\pi'.\mathsf{tr}}{\Longrightarrow} B$, and $(\pi \circ \pi').t$ is the unique trace of the form $A \overset{(\pi\circ\pi').\mathsf{tr}}{\Longrightarrow} B$. Hence the result since $\pi.\pi'.\mathsf{tr} = (\pi \circ \pi').\mathsf{tr}$ by definition of a group action. □

This formally justifies that the group-action properties of $(\pi, \mathsf{tr}) \mapsto \pi.\mathsf{tr}$ carry on to traces. Then, we also discuss the domain extension of permutations. If π is a permutation of $[\![1, n]\!]$, we define π_{+p}^{+q} permutation of $[\![1, n+p+q]\!]$ by

$$\pi_{+p}^{+q}(x) = \begin{cases} p + \pi(x-p) & \text{if } p < x \leqslant n+p \\ x & \text{otherwise} \end{cases}$$

in particular, the following result is immediate:

Proposition C.5 (extension). If π permutes independent actions of v, $\pi_{+|u|}^{+|w|}$ permutes independent actions of uvw.

C.3 Decomposition into phases

In this section we prove correct the refinement at the very basis of our partial-order reductions, namely that all traces can be decomposed into phases (modulo permutation of independent actions).

Proposition 4.5. $\mathbb{O}_{\mathsf{c},b}^\forall$ is a correct refinement of $\mathbb{O}_{\mathsf{all}}^\forall$.

Proof. By Proposition 4.4, it sufficies to prove that for all traces t that are maximal (i.e. whose last process is irreducible), there exists π permuting independent actions of t such that $\pi.t$ can be decomposed into phases.

We prove this by induction on the length of t. If t is empty the result is immediate: π is the identity and the phase decomposition consists of a unique empty negative block. Otherwise let us write

$$t : (A \xrightarrow{\alpha} B) \cdot t'.$$

Note in particular that the trace t' is also maximal. By induction hypothesis, there exists π' permuting independent actions of t' such that

$$\pi'.t' = b_0^- \cdot b_1^+ \cdot b_1^- \cdot b_2^+ \cdot b_2^- \cdots b_n^+ \cdot b_n^-$$

where each b_i^+ is a positive or null phase, and each b_i^- is a negative phase.

▹ *case 1*: α is an output or a parallel action.

Then $(A \xrightarrow{\alpha} B) \cdot b_0^-$ is a negative phase and it sufficies to choose $\pi = (\pi')^{+0}_{+1}$.

▹ *case 2*: $\alpha = [\,\tau\,]^{\ell_1|\ell_2}$ (internal communication).

Let us write E the multiset of actions of the word $\mathsf{tr}(b_0^-)$. We partition it into $E = F \uplus G$ where

$$F = \{\!\{\beta \in E \mid \alpha \parallel \beta\}\!\}$$
$$G = \{\!\{[\,a\,]^\ell \in E \mid \ell_1 \preccurlyeq_{pref} \ell \text{ or } \ell_2 \preccurlyeq_{pref} \ell\}\!\}$$

where $\preccurlyeq_{pref}$ refers to the prefix ordering on words. This is indeed a partition of E thanks to the invariant that any label appearing in t' is either incomparable with ℓ_1 and ℓ_2, or a suffix of ℓ_1 or ℓ_2. For the same reason, all actions in F are independent of all actions in G: it is therefore straightforward to construct π_0^- permuting independent actions of b_0^- such that

$$\pi_0^-.b_0^- : B \xRightarrow{\mathsf{tr}_F \cdot \mathsf{tr}_G} C \qquad \mathsf{tr}_F \in F^\star \qquad \mathsf{tr}_G \in G^\star\,.$$

Then, we let σ permuting independent actions of

$$s = (A \xrightarrow{\alpha} B) \cdot (\pi_0^-.b_0^-)$$

such that $\sigma.s = A \xRightarrow{\mathsf{tr}_F} B' \xrightarrow{\alpha} B'' \xRightarrow{\mathsf{tr}_G} C$. By definition of F and G, we have $polar(B') \neq \infty$. And by the hypothesis that b_0^- is a negative phase, its last process C has not a polarity of $-\infty$ neither. Therefore $A \xRightarrow{\mathsf{tr}_F} B'$ and $B'' \xRightarrow{\mathsf{tr}_G} C$ are negative phases. All in all, it sufficies to choose

$$\pi = \sigma^{+p}_{+0} \circ (\pi_0^-)^{+p}_{+1} \circ (\pi')^{+0}_{+1} \quad \text{with } p = \sum_{i=1}^{n} |b_i^+| + |b_i^-|$$

▹ *case 3*: $\alpha = [\,c(\xi)\,]^\ell$.

Let us write

$$A = (\{\!\{[\,c(x).P\,]^\ell\}\!\} \cup \mathcal{P}, \Phi) \qquad B = (\{\!\{[\,P'\,]^\ell\}\!\} \cup \mathcal{P}, \Phi)$$

If the label ℓ does not appear in $tr(t')$, then by maximality of t it needs be that $polar(P') = 0$ and $(A \xrightarrow{\alpha} B)$ is therefore a positive phase. In particular $\pi'.t$ is already decomposed into phases and it sufficies to choose $\pi = (\pi')^{+0}_{+1}$.

Otherwise assume that ℓ appears in $tr(t')$. We write

$$\mathsf{tr}_i^- = tr(b_i^-) \qquad\qquad \mathsf{tr}_i^+ = tr(b_i^+)$$

We also consider the phase of t' in which the first action of P' is executed, i.e. the first phase b such that ℓ appears in $tr(b)$. Note that, thanks to the invariant that any label appearing in t' is either incomparable or a suffix of ℓ, α is independent of all actions of all phases of t' preceding b.

▹ *case 3a*: $b = b_i^+$ is a positive or null phase.

Then we fix σ permuting independent actions of

$$\alpha \cdot \mathsf{tr} \qquad \text{with } \mathsf{tr} = \mathsf{tr}_0^- \cdot \mathsf{tr}_1^+ \cdot \mathsf{tr}_1^- \cdots \mathsf{tr}_{i-1}^+ \cdot \mathsf{tr}_{i-1}^-$$

such that, writing $s = (A \xrightarrow{\alpha} B) \cdot b_0^- \cdot b_1^+ \cdot b_1^- \cdots b_{i-1}^+ \cdot b_{i-1}^-$,

$$\sigma.s : A \xRightarrow{\mathsf{tr}} A' \xrightarrow{\alpha} A''\,.$$

If $b = b_i^+$ is a null phase, $A' \xrightarrow{\alpha} A''$ is a positive phase. If b is a positive phase, $(A' \xrightarrow{\alpha} A'') \cdot b$ is a positive phase too. In both cases, it sufficies to choose

$$\pi = \sigma^{+p}_{+0} \circ (\pi')^{+0}_{+1} \qquad \text{with } p = \sum_{j=i}^{n} |b_j^+| + |b_j^-|$$

▹ *case 3b*: $b = b_i^-$ is a negative phase.

Similarly to case 2, we fix E the multiset of actions appearing in the word tr_i^- and we partition it as $E = F \uplus G$

$$F = \{\beta \in E \mid \alpha \parallel \beta\} \quad G = \{[\,a\,]^{\ell'} \in E \mid \ell \preccurlyeq_{pref} \ell'\}\,.$$

And again we let π_i^- permuting tr_i^- such that

$$\pi_i^-.b_i^- : R \xRightarrow{\mathsf{tr}_F} S \xRightarrow{\mathsf{tr}_G} T \qquad \mathsf{tr}_F \in F^\star \qquad \mathsf{tr}_G \in G^\star\,.$$

Then we let σ permuting independent actions of

$$\alpha \cdot \mathsf{tr} \cdot \mathsf{tr}_F \qquad \text{with } \mathsf{tr} = \mathsf{tr}_0^- \cdot \mathsf{tr}_1^+ \cdot \mathsf{tr}_1^- \cdots \mathsf{tr}_i^+$$

such that, writing $s = (A \xrightarrow{\alpha} B) \cdot b_0^- \cdot b_1^+ \cdot b_1^- \cdots b_i^+ \cdot (\pi_i^-.b_i^-)$,

$$\sigma.s : A \xRightarrow{\mathsf{tr}} A' \xRightarrow{\mathsf{tr}_F} A'' \xrightarrow{\alpha} S \xRightarrow{\mathsf{tr}_G} T\,.$$

For the same reason as in case 2, $A' \xRightarrow{\mathsf{tr}_F} A''$ and $S \xRightarrow{\mathsf{tr}_G} T$ are negative phases. It therefore sufficies to choose

$$\pi = \sigma^{+p}_{+0} \circ (\pi_i^-)^{p}_{1+|\mathsf{tr}|} \circ (\pi')^{+0}_{+1} \quad \text{with } p = \sum_{j=i+1}^{n} |b_j^+| + |b_j^-| \quad \square$$

C.4 Lexicographic reduction

In this section we prove the correctness of the optimisation $\mathbb{O}^{\vee}_{\mathsf{c+i+r}}$ introduced in Section 4.3. We recall that we assume a total ordering $\preccurlyeq$ on blocks that is insensitive to recipes. We write $\preccurlyeq_{lex}$ the lexicographic extension of $\preccurlyeq$ on

words of same length of blocks (i.e two words of different length are always incomparable w.r.t. $\preccurlyeq_{lex}$). The core of the proof is to establish a link between $\mathbb{O}^{\forall}_{\text{c+r}}$ and

$$\mathbb{O}^{\forall}_{lex} = \{t \in \mathbb{O}^{\forall}_{\text{c}} \mid t \text{ minimal}\}$$

where *minimal* means minimal within its equivalence class for $\equiv_{\text{b-por}}$ w.r.t. $\preccurlyeq_{lex}$.

Proposition C.6 . $\mathbb{O}^{\forall}_{lex} \subseteq \mathbb{O}^{\forall}_{\text{c+r}}$

Proof. Let $t \in \mathbb{O}^{\forall}_{lex}$ and prove by induction on the number of blocks of t that $t \in \mathbb{O}^{\forall}_{\text{c+r}}$. If t is a single negative phase, the conclusion follows from the definition. Otherwise let us write $t : b^- \cdot b_1 \cdots b_n$. Since lexicographic minimality is preserved by prefix,we have $t' \in \mathbb{O}^{\forall}_{lex}$ with

$$t' : b^- \cdot b_1 \cdots b_{n-1} \,.$$

Then by induction hypothesis we obtain $t' \in \mathbb{O}^{\forall}_{\text{c+r}}$. To conclude, by definition of $\mathbb{O}^{\forall}_{\text{c+r}}$, it now sufficies to prove that b_n is allowed after t'.

Suppose by contradiction that it is not, and let b recipe equivalent to b_n such that $\mathsf{Minimal}(t', b)$ does not hold. By a quick induction on the hypothesis $\mathsf{Minimal}(t', b)$, we can show that there exists $i \in [\![2, n-1]\!]$ such that

(1) $b_i \prec \cdots \prec b_{n-1} \prec b$

(2) $b \prec b_{i-1}$

(3) $\forall j \in [\![i, n-1]\!], b_i \parallel b$.

In particular $\pi = (i \;\; i{+}1 \;\; \cdots \;\; n{-}1 \;\; n)$ permutes independent actions of $t' \cdot b$ and $\pi.(t' \cdot b) \preccurlyeq_{lex} t' \cdot b$. Since $\preccurlyeq$ is insensitive to recipes, this contradicts the minimality of t. □

Also note that, thanks to Proposition C.5, we also obtain the useful property that $\equiv_{\text{b-por}}$ is closed by context:

Proposition C.7 . If $v \equiv_{\text{b-por}} v'$ then $uvw \equiv_{\text{b-por}} uv'w$.

Proof. Easy induction on the hypothesis $v \equiv_{\text{b-por}} v'$. □

Using these two properties, we can eventually prove the correctness of $\mathbb{O}^{\forall}_{\text{por}}$ by relying on Corollary 4.8.

Proposition 4.10 . $\mathbb{O}^{\forall}_{\text{por}}$ is a correct refinement of $\mathbb{O}^{\forall}_{\text{c+i}}$.

Proof. Let $\mathbb{O}^{\forall}$ the set of traces of the form $t : b^- \cdot t_p \cdot t_i$, where b^- is a negative phase, $t_p \in \mathbb{O}^{\forall}_{lex}$ only contains proper blocks, and $t_i \in \mathbb{O}^{\forall}_{lex}$ only contains improper blocks. By Proposition C.6, $\mathbb{O}^{\forall} \subseteq \mathbb{O}^{\forall}_{\text{por}}$. By partial implementability (Proposition 3.8) it therefore sufficies to prove that $\mathbb{O}^{\forall}$ is a correct refinement of $\mathbb{O}^{\forall}_{\text{c+i}}$.

We rely on Corollary 4.8, i.e. we prove that for any $t \in \mathbb{O}^{\forall}_{\text{c+i}}$, there is $t' \equiv_{\text{b-por}} t$ such that $t' \in \mathbb{O}^{\forall}_{lex}$. We write

$$t : b^- \cdot t_p \cdot t_i \qquad \text{(notations of the definition)}$$

and let t'_p and t'_i the $\preccurlyeq_{lex}$-minimal elements of the $\equiv_{\text{b-por}}$-equivalence classes of, respectively, t_p and t_i. Naturally $t_p \equiv_{\text{b-por}} t'_p$ and $t_i \equiv_{\text{b-por}} t'_i$. Therefore $t' : b^- \cdot t'_p \cdot t'_i \in \mathbb{O}^{\forall}_{lex}$. But by closure under context (Proposition C.7) we also have $t' \equiv_{\text{b-por}} t$, hence the conclusion. □

D Proofs of Section 5

D.1 Universal symmetries

In this section we prove the correctness of the universal reduction by symmetry, i.e.

Proposition 5.3 . $\mathbb{O}^{\forall}_{\text{por}} \cap \mathbb{O}^{\forall}_{\text{sym}}$ is a correct refinement of $\mathbb{O}^{\forall}_{\text{por}}$.

First of all we isolate the core property that symmetric processes verify, which will be the key argument for the proofs of this section. Let us consider a compressed trace

$$t : [\,P\,]^{\varepsilon} \overset{\text{tr}}{\Longrightarrow} (\{\!\{[\,P_i\,]^{\ell_i}\}\!\}_{i=1}^{n}, \Phi_0) \qquad \in \mathbb{O}^{\forall}_{\text{c}}$$

and let $a, b \in [\![1, n]\!]$ and $\pi = (a \;\; b)$. We also assume that there exists ϱ_c permutation of Ch_{pub} such that

$$\pi.\langle P_1, \ldots, P_{a-1}, P_a\varrho_c, P_{a+1}, \ldots, P_b\varrho_c, \ldots, P_n\rangle \equiv^{\Phi_0}_{\alpha} \vec{P} \qquad (4)$$

Proposition D.1 . Let a trace of the form

$$s \;=\; t \cdot ((\mathcal{P}_0, \Phi_0)\sigma \xrightarrow{[\,a_1\,]^{L_1}} \cdots \xrightarrow{[\,a_n\,]^{L_n}} (\mathcal{P}_n, \Phi_n)\sigma) \qquad (\star)$$

for some substitution σ and $L_1 = \ell_b$. Then there exists ϱ permutation of $\mathcal{N}_{priv}$, and $\sigma' =_E \sigma$ and a trace

$$t \cdot ((\mathcal{Q}_0, \Phi_0)\sigma'\varrho \xrightarrow{[\,a_1\varrho_c\,]^{L'_1}} \cdots \xrightarrow{[\,a_n\varrho_c\,]^{L'_n}} (\mathcal{Q}_n, \Phi_n)\sigma'\varrho)$$

and π a bijection of labels such that

(1) if $\ell \in dom(\pi)$ then ℓ_a is a prefix of ℓ, and if $\ell \in im(\pi)$ then ℓ_b is a prefix of ℓ. Besides if ℓ_a or ℓ_b is a prefix of ℓ and ℓ appears in the trace s, then $\ell \in dom(\pi) \cup im(\pi)$.

(2) $\pi(\ell_a) = \ell_b$

(3) if $\ell, \ell \cdot i \in dom(\pi)$ then $\pi(\ell \cdot i) = \pi(\ell) \cdot j$ for some j

(4) if $L_i \in dom(\pi)$ then $L'_i = \pi(L_i)$; if $L_i \in im(\pi)$ then $L'_i = \pi^{-1}(L_i)$; otherwise $L'_i = L_i$.

(5) if $\mathcal{P}_i$ contains a process P labelled $\ell \in dom(\pi)$ (resp. $\ell \in im(\pi)$), then $\mathcal{Q}_i$ contains a process Q labelled $\pi(\ell)$ (resp. $\pi^{-1}(\ell)$) such that $P_i \equiv Q_i\sigma'\varrho$ (if $i \notin \{a, b\}$) and $P_i \equiv Q_i\sigma'\varrho\varrho_c$ (if $i \in \{a, b\}$).

Proof. The case $n = 1$ follows from hypothesis (Equation (4)), and the proposition can then be proved by induction on n. □

In particular we obtain the following two core arguments:

Proposition D.2. Assume $a \longleftrightarrow_1 b$ and that, for all traces s of the form of Equation ($\star$) (with $L_1 = \ell_a$), there exists $s^2 \in \mathbb{T}(P, Q)$ such that $s = \mathsf{fst}(s^2) \sim \mathsf{snd}(s^2)$. Then for all traces s of the form of Equation ($\star$) (with $L_1 = \ell_b$), there exists $s^2 \in \mathbb{T}(P, Q)$ such that $s = \mathsf{fst}(s^2) \sim \mathsf{snd}(s^2)$.

Proof. Let a trace s of the form of Equation ($\star$) (with $L_1 = \ell_b$). With $\varrho_c = \mathsf{id}$ we consider the trace s' given by Proposition D.1 (we will use the same notations). In particular we have $L'_1 = \ell_a$. By hypothesis, there therefore exists $u^2 \in \mathbb{T}(P, Q)$ such that $s' = \mathsf{fst}(u^2) \sim \mathsf{snd}(u^2)$. Using now the hypothesis that $a \longleftrightarrow_1 b$, there also exists $v^2 \in \mathbb{T}(P, Q)$ matching the first $|\mathsf{tr}|$ actions of s' and whose processes matching P_a and P_b are swapped. From u^2 and v^2 we can then construct by induction on n a trace s^2 such that $s = \mathsf{fst}(s^2) \sim \mathsf{snd}(s^2)$. □

Proposition D.3. Assume $a \longleftrightarrow_1 b$ and that, for all traces s of the form of Equation ($\star$) (with $L_1 = \ell_a$), there exists $s^2 \in \mathbb{T}(P, Q)$ such that $s = \mathsf{fst}(s^2) \sim \mathsf{snd}(s^2)$. Then for all traces s of the form of Equation ($\star$) (with $L_1 = \ell_b$), there exists $s^2 \in \mathbb{T}(P, Q)$ such that $s = \mathsf{fst}(s^2) \sim \mathsf{snd}(s^2)$.

Proof. Let a trace s of the form of Equation ($\star$) (with $L_1 = \ell_b$). We consider the trace s' given by Proposition D.1 (we will use the same notations). In particular we have $L'_1 = \ell_a$. By hypothesis, and using the characterisation of Appendix A, there exists $u \in \mathbb{T}(Q)$ and a session matching for s', u such that $tr(s') = tr(u)$ and $\Phi(s') \sim \Phi(u)$. Thanks to the hypothesis $a \longleftrightarrow_2 b$, we can then apply Proposition D.1 to the trace u too. In particular there exists a trace $u' \in \mathbb{T}(Q)$ such that $tr(t) = tr(u')$ and $\Phi(t) \sim \Phi(u')$, hence the conclusion. □

Using these two propositions we obtain the straightforward corollary (by induction on the hypothesis that $a \longleftrightarrow b$) that justifies Proposition 5.3:

Corollary D.4. Assume $a \longleftrightarrow b$ and that, for all traces s of the form of Equation ($\star$) (with $L_1 = \ell_a$), there exists $s^2 \in \mathbb{T}(P, Q)$ such that $s = \mathsf{fst}(s^2) \sim \mathsf{snd}(s^2)$. Then for all traces s of the form of Equation ($\star$) (with $L_1 = \ell_b$), there exists $s^2 \in \mathbb{T}(P, Q)$ such that $s = \mathsf{fst}(s^2) \sim \mathsf{snd}(s^2)$.

D.2 Existential symmetries

In this section we prove the existential optimisation relying on symmetries of matchings:

Proposition 5.5. $\mathbb{O}^{\exists}_{\mathsf{sym}}$ is a correct refinement of $\mathbb{O}^{\exists}_{\mathsf{all}}$.

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

$$(\{\!\{(P_i, Q_i)\}\!\}_{i=1}^n, \Phi_0, \Phi_1) \equiv_\alpha (\{\!\{(P_i, Q'_i\varrho)\}\!\}_{i=1}^n, \Phi_0, \Phi'_1\varrho)$$

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.5. $\equiv_\alpha$ 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 $t\varrho =_E t'\varrho$. □

Proposition D.6. The relation $\equiv_\alpha$ has the properties:

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

(*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 $\Phi \sim \Phi\varrho$ for any bijective renaming of private names ϱ. The property (*iii*) is immediate. □

Proof of Proposition 5.5. We write $\approx$ the notion of equivalence induced by the optimisation $\mathbb{O}^{\exists}_{\mathsf{sym}}$. 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.6 (Item (*i*)), there exists $s^2 \in \mathbb{O}^{\exists}_{\mathsf{sym}}$ such that $t^2 \equiv_\alpha s^2$. Therefore we have, by Items (*i*) and (*ii*),

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