Logical Methods in Computer Science
Vol. 13(2:8)2017, pp. 1–48
https://lmcs.episciences.org/

Submitted Jul. 28, 2016
Published Jun. 08, 2017

A REDUCED SEMANTICS FOR DECIDING TRACE EQUIVALENCE

DAVID BAELDE [a], STÉPHANIE DELAUNE [b], AND LUCCA HIRSCHI [c]

[a,c] LSV, ENS Cachan & CNRS, Université Paris-Saclay, France
e-mail address: {baelde,hirschi}@lsv.ens-cachan.fr

[b] CNRS & IRISA, France
e-mail address: stephanie.delaune@irisa.fr

Abstract. Many privacy-type properties of security protocols can be modelled using trace equivalence properties in suitable process algebras. It has been shown that such properties can be decided for interesting classes of finite processes (*i.e.* without replication) by means of symbolic execution and constraint solving. However, this does not suffice to obtain practical tools. Current prototypes suffer from a classical combinatorial explosion problem caused by the exploration of many interleavings in the behaviour of processes. Mödersheim *et al.* [40] have tackled this problem for reachability properties using partial order reduction techniques. We revisit their work, generalize it and adapt it for equivalence checking. We obtain an optimisation in the form of a reduced symbolic semantics that eliminates redundant interleavings on the fly. The obtained partial order reduction technique has been integrated in a tool called Apte. We conducted complete benchmarks showing dramatic improvements.

1. Introduction

Security protocols are widely used today to secure transactions that rely on public channels like the Internet, where malicious agents may listen to communications and interfere with them. Security has a different meaning depending on the underlying application. It ranges from the confidentiality of data (medical files, secret keys, etc.) to, *e.g.* verifiability in electronic voting systems. Another example is the notion of privacy that appears in many contexts such as vote-privacy in electronic voting or untraceability in RFID technologies.

To achieve their security goals, security protocols rely on various cryptographic primitives such as symmetric and asymmetric encryptions, signatures, and hashes. Protocols also involve a high level of concurrency and are difficult to analyse by hand. Actually, many protocols have been shown to be flawed several years after their publication (and deployment). For example, a flaw has been discovered in the Single-Sign-On protocol used, *e.g.* by Google Apps. It has been shown that a malicious application could very easily get access to any

This work has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No 714955-POPSTAR), as well as the ANR projects JCJC VIP ANR-11-JS02-006 and Sequoia ANR-14-CE28-0030-01.

DOI:10.23638/LMCS-13(2:8)2017

other application (*e.g.* Gmail or Google Calendar) of their users [6]. This flaw has been found when analysing the protocol using formal methods, abstracting messages by a term algebra and using the Avantssar validation platform [8]. Another example is a flaw on vote-privacy discovered during the formal and manual analysis of an electronic voting protocol [27].

Formal symbolic methods have proved their usefulness for precisely analysing the security of protocols. Moreover, it allows one to benefit from machine support through the use of various existing techniques, ranging from model-checking to resolution and rewriting techniques. Nowdays, several verification tools are available, *e.g.* [13, 28, 7, 38, 43]. A synthesis of decidability and undecidability results for equivalence-based security properties, and an overview of existing verification tools that may be used to verify equivalence-based security properties can be found in [36].

In order to design decision procedures, a reasonable assumption is to bound the number of protocol sessions, thereby limiting the length of execution traces. Under such an hypothesis, a wide variety of model-checking approaches have been developed (*e.g.* [39, 47]), and several tools are now available to automatically verify cryptographic protocols, *e.g.* [46, 7]. A major challenge faced here is that one has to account for infinitely many behaviours of the attacker, who can generate arbitrary messages. In order to cope with this prolific attacker problem and obtain decision procedures, approaches based on symbolic semantics and constraint resolution have been proposed [39, 42]. This has lead to tools for verifying reachability-based security properties such as confidentiality [39] or, more recently, equivalence-based properties such as privacy [47, 20, 15]. Unfortunately, the resulting tools, especially those for checking equivalence (*e.g.* Apte [19], Spec [47], Akiss [16]) have a very limited practical impact because they scale badly. This is not surprising since they treat concurrency in a very naive way, exploring all possible symbolic interleavings of concurrent actions.

Related work. In standard model-checking approaches for concurrent systems, the interleaving problem is handled using partial order reduction (POR) techniques [41]. In a nutshell, these techniques aim to effectively exploit the fact that the order of execution of two independent (parallel) actions is irrelevant when checking reachability. The theory of partial order reduction is well developed in the context of reactive systems verification (*e.g.* [41, 11, 34]). However, as pointed out by Clarke *et al.* in [26], POR techniques from traditional model-checking cannot be directly applied in the context of security protocol verification. Indeed, the application to security requires one to keep track of the knowledge of the attacker, and to refer to this knowledge in a meaningful way (in particular to know which messages can be forged at some point to feed some input). Furthermore, security protocol analysis does not rely on the internal reduction of a protocol, but has to consider arbitrary execution contexts (representing interactions with arbitrary, active attackers). Thus, any input may depend on any output, since the attacker has the liberty of constructing arbitrary messages from past outputs. This results in a dependency relation which is *a priori* very large, rendering traditional POR arguments suboptimal, and calling for domain-specific techniques.

In order to improve existing verification tools for security protocols, one has to design POR techniques that integrate nicely with symbolic execution. This is necessary to precisely deal with infinite, structured data. In this task, we get some inspiration from Mödersheim *et al.* [40], who design a partial order reduction technique that blends well with symbolic execution in the context of security protocols verification. However, we shall see that their key insight is not fully exploited, and yields only a quite limited partial order reduction.

Moreover, they only consider reachability properties (like all previous work on POR for security protocol verification) while we seek an approach that is adequate for model-checking equivalence properties.

Contributions. In this paper, we revisit the work of [40] to obtain a partial order reduction technique for the verification of equivalence properties. Among the several definitions of equivalence that have been proposed, we consider *trace equivalence* in this paper: two processes are trace equivalent when they have the same sets of observable traces and, for each such trace, sequences of messages outputted by the two processes are *statically equivalent*, *i.e.* indistinguishable for the attacker. This notion is well-studied and several algorithms and tools support it [14, 24, 47, 20, 15]. Contrary to what happens for reachability-based properties, trace equivalence cannot be decided relying only on the reachable states. The sequence of actions that leads to this state plays a role. Hence, extra precautions have to be taken before discarding a particular interleaving: we have to ensure that this is done in both sides of the equivalence in a similar fashion. Our main contribution is an optimised form of equivalence that discards a lot of interleavings, and a proof that this reduced equivalence coincides with trace equivalence. Furthermore, our study brings an improvement of the original technique [40] that would apply equally well for reachability checking. On the practical side, we explain how we integrated our partial order reduction into the state-of-the art tool Apte [20], prove the correctness of this integration, and provide experimental results showing dramatic improvements. We believe that our presentation is generic enough to be easily adapted for other tools (provided that they are based on a forward symbolic exploration of traces combined with a constraint solving procedure). A big picture of the whole approach along with the new results is given in Figure 1. Vertically, it goes from the regular semantics, to symbolic semantics and Apte's semantics. Those semantics have variants when our optimisations are applied or not: no optimisation, only compression or compression plus reduction.

This paper essentially subsumes the conference paper that has been published in 2014 [9]. However, we consider here a generalization of the semantics used in [9]. This generalization notably allows us to capture the semantics used in Apte, which allows us to formally prove the integration of our optimisations in that tool. In addition, this paper incorporates proofs of all the results, additional examples, and an extensive related work section. Finally, it comes with a solid implementation in the tool Apte [19].

Outline. In Section 2, we introduce our model for security processes. We then consider the class of simple processes introduced in [22], with else branches and no replication. Then we present two successive optimisations in the form of refined semantics and associated trace equivalences. Section 3 presents a *compressed* semantics that limits interleavings by executing blocks of actions. Then, by adapting well-known argument, this is lifted to a symbolic semantics in Section 4. Section 5 presents the *reduced* semantics which makes use of *dependency constraints* to remove more interleavings. In Section 6, we explain how this reduced semantics has been integrated in the tool Apte, prove its correcteness, and give some benchmarks obtained on several case studies. Finally, Section 7 is devoted to related work, and concluding remarks are given in Section 8. An overview of the different semantics we will define and the results relating them is depicted in Figure 1. A table of symbols can be found in Appendix A.

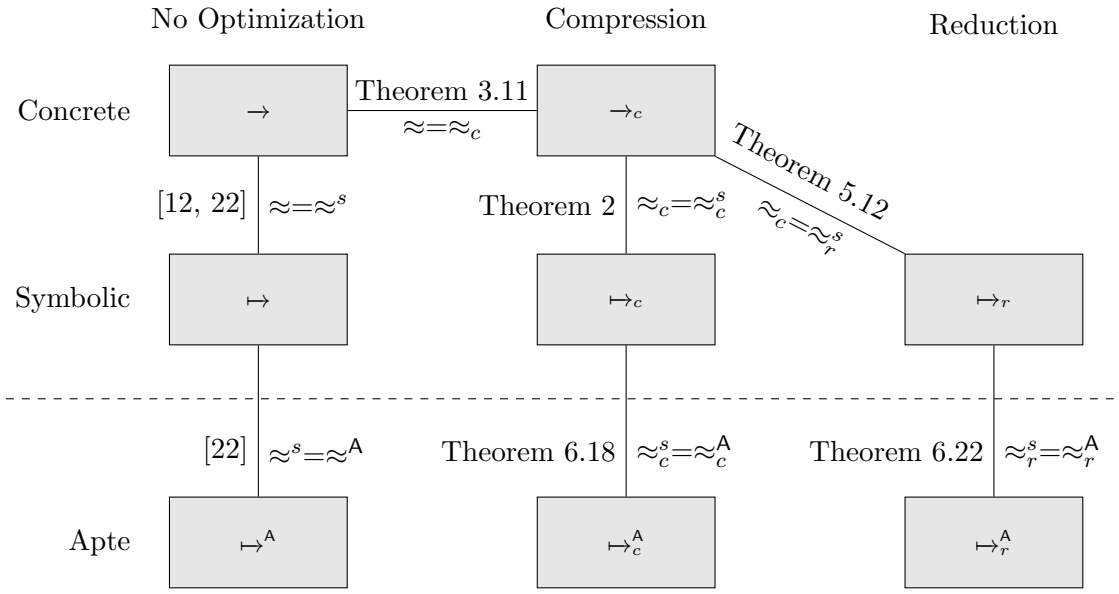


Figure 1: Overview of the paper

2. Model for security protocols

In this section, we introduce the cryptographic process calculus that we will use to describe security protocols. This calculus is close to the applied pi calculus [1]. We consider a semantics in the spirit of the one used in [9] but we also allow to block some actions depending on a validity predicate. This predicate can be chosen in such a way that no action is blocked, making the semantics as in [9]. It can also be chosen as in Apte as we eventually do in order to prove the integration of our optimisations into this tool.

2.1. **Syntax.** A protocol consists of some agents communicating on a network. Messages sent by agents are modeled using a term algebra. We assume two infinite and disjoint sets of variables, $\mathcal{X}$ and $\mathcal{W}$. Members of $\mathcal{X}$ are denoted x, y, z, whereas members of $\mathcal{W}$ are denoted w and used as *handles* for previously output terms. We also assume a set $\mathcal{N}$ of *names*, which are used for representing keys or nonces[1], and a signature Σ consisting of a finite set of function symbols. Terms are generated inductively from names, variables, and function symbols applied to other terms. For $S \subseteq \mathcal{X} \cup \mathcal{W} \cup \mathcal{N}$, the set of terms built from S by applying function symbols in Σ is denoted by $\mathcal{T}(\Sigma, S)$. We write $st(t)$ for the set of syntactic subterms of a term t. Terms in $\mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X})$ are denoted by u, v, etc. while terms in $\mathcal{T}(\Sigma, \mathcal{W})$ represent *recipes* (describing how the attacker built a term from the available outputs) and are written M, N, R. We write $fv(t)$ for the set of variables (from $\mathcal{X}$ or $\mathcal{W}$) occurring in a term t. A term is *ground* if it does not contain any variable, *i.e.* it belongs to $\mathcal{T}(\Sigma, \mathcal{N})$. One may rely on a sort system for terms, but its details are unimportant for this paper.

[1] Note that we do not have an explicit set of restricted (private) names. Actually, all names are restricted and public ones will be explicitly given to the attacker.

To model algebraic properties of cryptographic primitives, we consider an equational theory E. The theory will usually be generated from a finite set of axioms enjoying nice properties (*e.g.* convergence) but these aspects are irrelevant for the present work.

Example 2.1. *In order to model asymmetric encryption and pairing, we consider:*

$$\Sigma = \{\mathsf{aenc}(\cdot,\cdot),\ \mathsf{adec}(\cdot,\cdot),\ \mathsf{pk}(\cdot),\ \langle\cdot,\cdot\rangle,\ \pi_1(\cdot),\ \pi_2(\cdot)\}.$$

To take into account the properties of these operators, we consider the equational theory $\mathsf{E}_{\mathsf{aenc}}$ *generated by the three following equations:*

$$\mathsf{adec}(\mathsf{aenc}(x,\mathsf{pk}(y)),y) = x, \quad \pi_1(\langle x_1,x_2\rangle) = x_1, \ \textit{and} \ \ \pi_2(\langle x_1,x_2\rangle) = x_2.$$

For instance, we have $\pi_2(\mathsf{adec}(\mathsf{aenc}(\langle n,\mathsf{pk}(ska)\rangle,\mathsf{pk}(skb)),skb)) =_{\mathsf{E}_{\mathsf{aenc}}} \mathsf{pk}(ska)$.

Our model is parameterized by a notion of *message*, intuitively meant to represent terms that can actually be communicated by processes. Formally, we assume a special subset of ground terms $\mathcal{M}$, only requiring that it contains at least one public constant. Then, we say that a ground term u is *valid*, denoted $valid_{\mathcal{M}}(u)$, whenever for any $v \in st(u)$, we have that there exists $v' \in \mathcal{M}$ such that $v =_{\mathsf{E}} v'$. This notion of validity will be imposed on communicated terms. As we shall see, $\mathcal{M}$ can be chosen in such a way that the validity constraint allows us to discard some terms for which the computation of some parts fail. Note that $\mathcal{M}$ can also be chosen to be the set of all ground terms, yielding a trivial validity predicate that holds for all ground terms. The following developments are parametrized by $\mathcal{M}$.

Example 2.2. *The signature used in* Apte *is* $\Sigma = \Sigma_c \cup \Sigma_d$ *where:*

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

where Σ_0 *may contain some additional user-defined function symbols. The equational theory* $\mathsf{E}_{\mathsf{Apte}}$ *of* Apte *is an extension of the theory* $\mathsf{E}_{\mathsf{aenc}}$ *generated by adding the following equations:*

$$\mathsf{dec}(\mathsf{enc}(x,y),y) = x \qquad \mathsf{check}(\mathsf{sign}(x,y),\mathsf{vk}(y)) = x$$

The validity predicate used in the semantics of Apte *is obtained by taking* $\mathcal{M} = \mathcal{T}(\Sigma_c,\mathcal{N})$, i.e. *the ground terms built using constructor symbols. This choice allows us to discard terms for which a failure will happen during the computation and which therefore do not correspond to a message:* e.g. $\pi_1(\langle\mathsf{ok},\mathsf{dec}(\mathsf{enc}(a,k),k')\rangle)$ *is not valid since* $\mathsf{dec}(\mathsf{enc}(a,k),k')$ *is not equal modulo* $\mathsf{E}_{\mathsf{Apte}}$ *to a term in* $\mathcal{T}(\Sigma_c,\mathcal{N})$.

We do not need the full applied pi calculus [1] to represent security protocols. Here, we only consider public channels and we assume that each process communicates on a dedicated channel. Formally, we assume a set $\mathcal{C}$ of *channels* and we consider the fragment of *simple processes* without replication built on *basic processes* as defined in [22]. A basic process represents a party in a protocol, which may sequentially perform actions such as waiting for a message, checking that a message has a certain form, or outputting a message. Then, a simple process is a parallel composition of such basic processes playing on distinct channels.

Definition 2.3 (basic/simple process)**.** *The set of* basic processes *on $c \in \mathcal{C}$ is defined using the following grammar (where $u, v \in \mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X})$ and $x \in \mathcal{X}$):*

$$\begin{array}{rcll} P, Q & := & 0 & \textit{null} \\ & | & \texttt{if } u = v \texttt{ then } P \texttt{ else } Q & \textit{conditional} \\ & | & \texttt{in}(c, x).P & \textit{input} \\ & | & \texttt{out}(c, u).P & \textit{output} \end{array}$$

A simple process *$\mathcal{P} = \{P_1, \ldots, P_n\}$ is a multiset of basic processes P_i on pairwise distinct channels c_i. We assume that null processes are removed.*

Intuitively, a multiset of basic processes denotes a parallel composition. For conciseness, we often omit brackets, null processes, and even "$\texttt{else } 0$". Basic processes are denoted by the letters P and Q, whereas simple processes are denoted using $\mathcal{P}$ and $\mathcal{Q}$.

During an execution, the attacker learns the messages that have been sent on the different public channels. Those messages are organized into a *frame*.

Definition 2.4 (frame)**.** *A* frame *Φ is a substitution whose domain is included in $\mathcal{W}$ and image is included in $\mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X})$. It is written $\{w \triangleright u, \ldots\}$. A frame is* ground *when its image only contains ground terms.*

In the remainder of this paper, we will actually only consider ground frames that are made of valid terms.

An *extended simple process* (denoted A or B) is a pair made of a simple process and a frame. Similarly, we define *extended basic processes*. When the context makes it clear, we may omit "extended" and simply call them *simple processes* and *basic processes*.

Example 2.5. *We consider the protocol given in* [2] *designed for authenticating an agent with another one without revealing their identities to other participants. In this protocol, A is willing to engage in communication with B and wants to be sure that she is indeed talking to B and not to an attacker who is trying to impersonate B. However, A does not want to compromise her privacy by revealing her identity or the identity of B more broadly. The participants A and B proceed as follows:*

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

First A sends to B a nonce N_a and her public key encrypted with the public key of B. If the message is of the expected form then B sends to A the nonce N_a, a freshly generated nonce N_b and his public key, all of this being encrypted with the public key of A. Moreover, if the message received by B is not of the expected form then B sends out a "decoy" message: $\{N_b\}_{\mathsf{pub}_B}$. This message should basically look like B's other message from the point of view of an outsider.

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

$$\begin{array}{rcl} P(ska, pkb) & \stackrel{\mathsf{def}}{=} & \texttt{out}(c_A, \mathsf{aenc}(\langle n_a, \mathsf{pk}(ska)\rangle, pkb)). \\ & & \texttt{in}(c_A, x). \\ & & \texttt{if } \langle \pi_1(\mathsf{adec}(x, ska)), \pi_2(\pi_2(\mathsf{adec}(x, ska)))\rangle = \langle n_a, pkb\rangle \texttt{ then } 0 \end{array}$$

Here, we are only considering the authentication protocol. A more comprehensive model should include the access to an application in case of a success. Similarly, a session of role B

played by agent b with a can be modeled by the following basic process, where $N = \mathsf{adec}(y, skb)$.

$$\begin{array}{rcl} Q(skb, pka) & \stackrel{\mathsf{def}}{=} & \mathtt{in}(c_B, y). \\ & & \mathtt{if}\ \pi_2(N) = pka\ \mathtt{then}\ \mathtt{out}(c_B, \mathsf{aenc}(\langle \pi_1(N), \langle n_b, \mathsf{pk}(skb)\rangle\rangle, pka)) \\ & & \qquad\qquad \mathtt{else}\ \mathtt{out}(c_B, \mathsf{aenc}(n_b, \mathsf{pk}(skb))) \end{array}$$

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

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

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

2.2. **Semantics.** We first define a standard concrete semantics using a relation over *ground* extended simple processes, *i.e.* extended simple processes $(\mathcal{P}; \Phi)$ such that $fv(\mathcal{P}) = \emptyset$ (as said above, we also assume that Φ contains only valid ground terms). The semantics of a ground extended simple process $(\mathcal{P}; \Phi)$ is induced by the relation $\xrightarrow{a}$ over ground extended simple processes as defined in Figure 2.

$$\textsc{In} \quad (\{\mathtt{in}(c, x).Q\} \uplus \mathcal{P}; \Phi) \xrightarrow{\mathtt{in}(c, M)} (\{Q\{x \mapsto u\}\} \uplus \mathcal{P}; \Phi) \quad \text{if } M \in \mathcal{T}(\Sigma, \mathrm{dom}(\Phi)),\ valid(M\Phi), valid(u) \text{ and } M\Phi =_{\mathsf{E}} u$$

$$\textsc{Out} \quad (\{\mathtt{out}(c, u).Q\} \uplus \mathcal{P}; \Phi) \xrightarrow{\mathtt{out}(c, w)} (\{Q\} \uplus \mathcal{P}; \Phi \cup \{w \triangleright u\}) \quad \text{if } w \text{ is a fresh variable, and } valid(u)$$

$$\textsc{Then} \quad (\{\mathtt{if}\ u = v\ \mathtt{then}\ Q_1\ \mathtt{else}\ Q_2\} \uplus \mathcal{P}; \Phi) \xrightarrow{\tau} (\{Q_1\} \uplus \mathcal{P}; \Phi) \quad \text{if } u =_{\mathsf{E}} v,\ valid(u), \text{ and } valid(v)$$

$$\textsc{Else} \quad (\{\mathtt{if}\ u = v\ \mathtt{then}\ Q_1\ \mathtt{else}\ Q_2\} \uplus \mathcal{P}; \Phi) \xrightarrow{\tau} (\{Q_2\} \uplus \mathcal{P}; \Phi) \quad \text{if } u \neq_{\mathsf{E}} v \text{ or } \neg valid(u) \text{ or } \neg valid(v)$$

where $c \in \mathcal{C}, w \in \mathcal{W}$ and $x \in \mathcal{X}$.

Figure 2: Concrete semantics

A process may input any valid term that an attacker can build (rule In): $\{x \mapsto u\}$ is a substitution that replaces any occurrence of x with u. Once a recipe M is fixed, we may note that there are still different instances of the rule, but only in the sense that u is chosen modulo the equational theory E. In practice, of course, not all such u are enumerated. How this is achieved in practice is orthogonal to the theoretical development carried out here. In the Out rule, we enrich the attacker's knowledge by adding the newly output term u, with a fresh handle w, to the frame. The two remaining rules are unobservable (τ action) from the point of view of the attacker. When $\mathcal{M}$ contains all the ground terms, $valid(u)$ is true for any term u and this semantics coincides with the one defined in [9]. However, this parameter gives us enough flexibility to obtain a semantics similar to the one used in Apte, and therefore formally prove in Section 6 how to integrate our techniques in Apte.

The relation $A \xrightarrow{a_1 \cdot \ldots \cdot a_k} B$ between extended simple processes, where $k \geq 0$ and each a_i is an observable or a τ action, is defined in the usual way. We also consider the relation $\stackrel{\mathsf{tr}}{\Longrightarrow}$

defined as follows: $A \stackrel{\mathsf{tr}}{\Longrightarrow} B$ if, and only if, there exists $a_1 \cdot \ldots \cdot a_k$ such that $A \xrightarrow{a_1 \cdot \ldots \cdot a_k} B$, and tr is obtained from $a_1 \cdot \ldots \cdot a_k$ by erasing all occurrences of τ.

Example 2.6. *Consider the simple process $(\mathcal{P}; \Phi_0)$ introduced in Example 2.5 (with $\mathcal{M}$ equal to $\mathcal{T}(\Sigma_c, \mathcal{N})$ as in* Apte*). We have:*

$$(\mathcal{P}; \Phi_0) \xrightarrow{\mathtt{out}(c_A, w_3) \cdot \mathtt{in}(c_B, w_3) \cdot \tau \cdot \mathtt{out}(c_B, w_4) \cdot \mathtt{in}(c_A, w_4) \cdot \tau} (\emptyset; \Phi).$$

This trace corresponds to the normal execution of one instance of the protocol. The two silent actions have been triggered using the Then *rule. The resulting frame Φ is as follows:*

$$\Phi_0 \uplus \{w_3 \rhd \mathsf{aenc}(\langle n_a, \mathsf{pk}(ska)\rangle, \mathsf{pk}(skb)),\ w_4 \rhd \mathsf{aenc}(\langle n_a, \langle n_b, \mathsf{pk}(skb)\rangle\rangle, \mathsf{pk}(ska))\}.$$

2.3. **Trace equivalence.** Many interesting security properties, such as privacy-type properties studied, *e.g.* in [5], are formalized using the notion of *trace equivalence*. Before defining trace equivalence, we first introduce the notion of *static equivalence* that compares sequences of messages.

Definition 2.7 (static equivalence)**.** *Two frames Φ and Φ' are in* static equivalence*, $\Phi \sim \Phi'$, when we have that $\mathrm{dom}(\Phi) = \mathrm{dom}(\Phi')$, and:*

- *$\mathit{valid}(M\Phi) \ \Leftrightarrow \ \mathit{valid}(M\Phi')$ for any term $M \in \mathcal{T}(\Sigma, \mathrm{dom}(\Phi))$; and*
- *$M\Phi =_{\mathsf{E}} N\Phi \ \Leftrightarrow \ M\Phi' =_{\mathsf{E}} N\Phi'$ for any terms $M, N \in \mathcal{T}(\Sigma, \mathrm{dom}(\Phi))$ such that $\mathit{valid}(M\Phi)$ and $\mathit{valid}(N\Phi)$.*

Intuitively, two frames are equivalent if an attacker cannot see the difference between the two situations they represent, *i.e.* they satisfy the same equalities and failures.

Example 2.8. *Consider the frame Φ given in Example 2.6 and the frame Φ' below:*

$$\Phi' \stackrel{\mathsf{def}}{=} \Phi_0 \uplus \{\ w_3 \rhd \mathsf{aenc}(\langle n_a, \mathsf{pk}(ska')\rangle, \mathsf{pk}(skb)),\ \ w_4 \rhd \mathsf{aenc}(n_b, \mathsf{pk}(skb))\ \}.$$

We have that $\Phi \sim \Phi'$. This is a non-trivial equivalence. Intuitively, it holds since the attacker is not able to decrypt any of the ciphertexts, and each ciphertext contains a nonce that prevents him to build it from its components.

Now, if we decide to give access to n_a to the attacker, i.e. *considering $\Phi_+ = \Phi \uplus \{w_5 \rhd n_a\}$ and $\Phi'_+ = \Phi' \uplus \{w_5 \rhd n_a\}$, then the two frames Φ_+ and Φ'_+ are not in static equivalence anymore as witnessed by $M = \mathsf{aenc}(\langle w_5, w_1\rangle, w_2)$ and $N = w_3$. Indeed, we have that $M\Phi_+ =_{\mathsf{E}_{\mathsf{aenc}}} N\Phi_+$ whereas $M\Phi'_+ \neq_{\mathsf{E}_{\mathsf{aenc}}} N\Phi'_+$, and all these witnesses are valid.*

Definition 2.9 (trace equivalence)**.** *Let A and B be two extended simple processes. We have that $A \sqsubseteq B$ if, for every sequence of actions tr such that $A \stackrel{\mathsf{tr}}{\Longrightarrow} (\mathcal{P}; \Phi)$, there exists $(\mathcal{P}'; \Phi')$ such that $B \stackrel{\mathsf{tr}}{\Longrightarrow} (\mathcal{P}'; \Phi')$ and $\Phi \sim \Phi'$. The processes A and B are* trace equivalent*, denoted by $A \approx B$, if $A \sqsubseteq B$ and $B \sqsubseteq A$.*

Example 2.10. *Intuitively, the private authentication protocol presented in Example 2.5 preserves anonymity if an attacker cannot distinguish whether b is willing to talk to a (represented by the process $Q(skb, \mathsf{pk}(ska))$) or willing to talk to a' (represented by the process $Q(skb, \mathsf{pk}(ska'))$), provided a, a' and b are honest participants. This can be expressed relying on the following equivalence:*

$$(Q(skb, \mathsf{pk}(ska)); \Phi_0) \stackrel{?}{\approx} (Q(skb, \mathsf{pk}(ska')); \Phi_0).$$

For illustration purposes, we also consider a variant of the process Q, denoted Q_0, where its `else` *branch has been replaced by* 0 *(*i.e. *the null process). We will see that the "decoy" message plays a crucial role to ensure privacy. We have that:*

$$(Q_0(skb, \mathsf{pk}(ska)); \Phi_0) \xrightarrow{\mathtt{in}(c_B, \mathsf{aenc}(\langle w_1, w_1\rangle, w_2)) \cdot \tau \cdot \mathtt{out}(c_B, w_3)} (\emptyset; \Phi)$$

where $\Phi = \Phi_0 \uplus \{w_3 \triangleright \mathsf{aenc}(\langle \mathsf{pk}(ska), \langle n_b, \mathsf{pk}(skb)\rangle\rangle, \mathsf{pk}(ska))\}$*. We may note that this trace does not correspond to a normal execution of the protocol. Still, the first input is fed with the message* $\mathsf{aenc}(\langle \mathsf{pk}(ska), \mathsf{pk}(ska)\rangle, \mathsf{pk}(skb))$ *which is a message of the expected format from the point of view of the process* $Q_0(skb, \mathsf{pk}(ska))$*. Therefore, once conditionals are positively evaluated, the output* $\mathtt{out}(c_B, w_3)$ *can be triggered.*

This trace has no counterpart in $(Q_0(skb, \mathsf{pk}(ska')); \Phi_0)$*. Indeed, we have that:*

$$(Q_0(skb, \mathsf{pk}(ska')); \Phi_0) \xrightarrow{\mathtt{in}(c_B, \mathsf{aenc}(\langle w_1, w_1\rangle, w_2)) \cdot \tau} (\emptyset; \Phi_0).$$

Hence, we have that $(Q_0(skb, \mathsf{pk}(ska)); \Phi_0) \not\approx (Q_0(skb, \mathsf{pk}(ska')); \Phi_0)$*.*

However, it is the case that $(Q(skb, \mathsf{pk}(ska)); \Phi_0) \approx (Q(skb, \mathsf{pk}(ska')); \Phi_0)$*. This equivalence can be checked using the tool* Apte [17] *within few seconds for a simple scenario as the one considered here, and that takes few minutes/days as soon as we want to consider 2/3 sessions of each role.*

3. Compression based on grouping actions

Our first refinement of the semantics, which we call compression, is closely related to focusing from proof theory [4]: we will assign a polarity to processes and constrain the shape of executed traces based on those polarities. This will provide a first significant reduction of the number of traces to consider when checking equivalence-based properties between simple processes. Moreover, compression can easily be used as a replacement for the usual semantics in verification algorithms.

The key idea is to force processes to perform all enabled output actions as soon as possible. In our setting, we can even safely force them to perform a complete *block* of input actions followed by ouput actions.

Example 3.1. *Consider the process* $(\mathcal{P}; \Phi)$ *with* $\mathcal{P} = \{\mathtt{in}(c_1, x).P_1,\ \mathtt{out}(c_2, b).P_2\}$*. In order to reach* $(\{P_1\{x \mapsto u\},\ P_2\}; \Phi \cup \{w \triangleright b\})$*, we have to execute the action* $\mathtt{in}(c_1, x)$ *(using a recipe* M *that allows one to deduce* u*) and the action* $\mathtt{out}(c_2, b)$ *(giving us a label of the form* $\mathtt{out}(c_2, w)$*). In case of reachability properties, the execution order of these actions only matters if* M *uses* w*. Thus we can safely perform the outputs in priority.*

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

3.1. **Compressed semantics.** We now introduce the compressed semantics. Compression is an optimisation, since it removes some interleavings. But it also gives rise to convenient "macro-actions", called *blocks*, that combine a sequence of inputs followed by some outputs, potentially hiding silent actions. Manipulating those blocks rather than indiviual actions makes it easier to define our second optimisation.

For sake of simplicity, we consider *initial* simple processes. A simple process $A = (\mathcal{P}; \Phi)$ is *initial* if for any $P \in \mathcal{P}$, we have that $P = 0$, $P = \mathtt{in}(c, x).P'$ or $P = \mathtt{out}(c, u).P'$ for some term u such that $\neg valid(u)$. In other words, each basic process composing A starts with an input unless it is blocked due to an unfeasible output.

Example 3.2. *Continuing Example 2.5,* $(\{P(ska, \mathsf{pk}(skb)), Q(skb, \mathsf{pk}(ska))\}; \Phi_0)$ *is not initial. Instead, we may consider* $(\{P_{\mathsf{init}}, Q(skb, \mathsf{pk}(ska))\}; \Phi_0)$ *where*

$$P_{\mathsf{init}} \stackrel{\mathsf{def}}{=} \mathtt{in}(c_A, z).\mathtt{if}\ z = \mathsf{start}\ \mathtt{then}\ P(ska, \mathsf{pk}(skb))$$

assuming that start *is a (public) constant in our signature.*

$$\textsc{In}\quad \frac{(P;\Phi) \xrightarrow{\mathtt{in}(c,M)} (P';\Phi') \quad (P';\Phi') \xrightarrow{\mathsf{tr}} \!\!\!\!\rightarrow_{i^*} (P'';\Phi'')}{(P;\Phi) \xrightarrow{\mathtt{in}(c,M)\cdot\mathsf{tr}} \!\!\!\!\rightarrow_{\ell} (P'';\Phi'')} \text{ with } \ell \in \{i^*; i^+\}$$

$$\textsc{Out}\quad \frac{(P;\Phi) \xrightarrow{\mathtt{out}(c,w)} (P';\Phi') \quad (P';\Phi') \xrightarrow{\mathsf{tr}} \!\!\!\!\rightarrow_{o^*} (P'';\Phi'')}{(P;\Phi) \xrightarrow{\mathtt{out}(c,w)\cdot\mathsf{tr}} \!\!\!\!\rightarrow_{\ell} (P'';\Phi'')} \text{ with } \ell \in \{i^*; o^*\}$$

$$\textsc{Tau}\quad \frac{(P;\Phi) \xrightarrow{\tau} (P';\Phi') \quad (P';\Phi') \xrightarrow{\mathsf{tr}} \!\!\!\!\rightarrow_{\ell} (P'';\Phi'')}{(P;\Phi) \xrightarrow{\mathsf{tr}} \!\!\!\!\rightarrow_{\ell} (P'';\Phi'')} \text{ with } \ell \in \{o^*; i^+; i^*\}$$

$$\textsc{Proper}\quad \frac{}{(0;\Phi) \xrightarrow{\epsilon} \!\!\!\!\rightarrow_{o^*} (0;\Phi)} \quad \frac{}{(\mathtt{in}(c,x).P;\Phi) \xrightarrow{\epsilon} \!\!\!\!\rightarrow_{o^*} (\mathtt{in}(c,x).P;\Phi)} \quad \frac{\neg valid(u)}{(\mathtt{out}(c,u).P;\Phi) \xrightarrow{\epsilon} \!\!\!\!\rightarrow_{o^*} (\mathtt{out}(c,u).P;\Phi)}$$

$$\textsc{Improper}\quad \frac{}{(0;\Phi) \xrightarrow{\epsilon} \!\!\!\!\rightarrow_{i^*} (\bot;\Phi)} \quad \frac{\neg valid(u)}{(\mathtt{out}(c,u).P;\Phi) \xrightarrow{\epsilon} \!\!\!\!\rightarrow_{i^*} (\bot;\Phi)}$$

Figure 3: Focused semantics on extended basic processes

The main idea of the compressed semantics is to ensure that when a basic process starts executing some actions, it actually executes a maximal block of actions. In analogy with focusing in sequent calculus, we say that the basic process takes the focus, and can only release it under particular conditions. We define in Figure 3 how blocks can be executed by extended basic processes. In that semantics, the label ℓ denotes the stage of the execution, starting with i^+, then i^* after the first input and o^* after the first output.

Example 3.3. *Going back to Example 2.10, we have that:*

$$(Q_0(skb, \mathsf{pk}(ska)); \Phi_0) \xrightarrow{\mathtt{in}(c_B, \mathsf{aenc}(\langle w_1, w_1\rangle, w_2))\cdot\mathtt{out}(c_B, w_3)} \!\!\!\!\rightarrow_{i^+} (0;\Phi)$$

where Φ *is as given in Example 2.10. As illustrated by the proof tree below, we also have* $(Q_0(skb, \mathsf{pk}(ska')); \Phi_0) \xrightarrow{\mathsf{tr}}\!\!\!\!\rightarrow_{i^+} (\bot; \Phi_0)$ *with* $\mathsf{tr} = \mathtt{in}(c_B, \mathsf{aenc}(\langle w_1, w_1\rangle, w_2))$.

$$\dfrac{(Q_0(skb, \mathsf{pk}(ska')); \Phi_0) \xrightarrow{\mathsf{tr}} (Q'; \Phi_0) \quad \dfrac{(Q'; \Phi_0) \xrightarrow{\tau} (0; \Phi_0) \quad \dfrac{}{(0; \Phi_0) \xrightarrow{\epsilon}\!\!\!\!\rightarrow_{i^*} (\bot; \Phi_0)}\ \textsc{Improper}}{(Q'; \Phi_0) \xrightarrow{\epsilon}\!\!\!\!\rightarrow_{i^*} (\bot; \Phi_0)}\ \textsc{Tau}}{(Q_0(skb, \mathsf{pk}(ska')); \Phi_0) \xrightarrow{\mathsf{tr}}\!\!\!\!\rightarrow_{i^+} (\bot; \Phi_0)}\ \textsc{In}$$

where $Q' \stackrel{\mathsf{def}}{=} \mathtt{if}\ \mathsf{pk}(ska) = \mathsf{pk}(ska')\ \mathtt{then}\ \mathtt{out}(c_B, u)$ *for some message* u.

Then we define the relation $\rightarrow_c$ between extended simple processes as the least reflexive transitive relation satisfying the rules given in Figure 4.

$$\textsc{Block}\ \dfrac{(Q; \Phi) \xrightarrow{\mathsf{tr}}\!\!\!\!\rightarrow_{i^+} (Q'; \Phi') \quad Q' \neq \bot}{(\{Q\} \uplus \mathcal{P}; \Phi) \xrightarrow{\mathsf{tr}}_c (\{Q'\} \uplus \mathcal{P}; \Phi')} \qquad \textsc{Failure}\ \dfrac{(Q; \Phi) \xrightarrow{\mathsf{tr}}\!\!\!\!\rightarrow_{i^+} (Q'; \Phi') \quad Q' = \bot}{(\{Q\} \uplus \mathcal{P}; \Phi) \xrightarrow{\mathsf{tr}}_c (\emptyset; \Phi')}$$

Figure 4: Compressed semantics on extended simple processes

A basic process is allowed to *properly* end a block execution when it has performed outputs and it cannot perform any more output or unobservable action (τ). Accordingly, we call *proper block* a non-empty sequence of inputs followed by a non-empty sequence of outputs, all on the same channel. For completeness, we also allow blocks to be terminated *improperly*, when the process that is executing has performed inputs but no output, and has reached the null process 0 or an output which is blocked. Accordingly, we call *improper block* a non-empty sequence of inputs on the same channel.

Example 3.4. *Continuing Example 3.3, using the rule* BLOCK*, we can derive*

$$(\{P_{\mathsf{init}}, Q_0(skb, \mathsf{pk}(ska))\}; \Phi_0) \xrightarrow{\mathtt{in}(c_B, \mathsf{aenc}(\langle w_1, w_1\rangle, w_2)) \cdot \mathtt{out}(c_B, w_3)}_c (P_{\mathsf{init}}; \Phi)$$

where P_{init} *is defined in Example 3.2. We can also derive*

$$(\{P_{\mathsf{init}}, Q_0(skb, \mathsf{pk}(ska'))\}; \Phi_0) \xrightarrow{\mathtt{in}(c_B, \mathsf{aenc}(\langle w_1, w_1\rangle, w_2))}_c (\emptyset; \Phi_0)$$

using the rule FAILURE*. Note that the resulting simple process is reduced to* $\emptyset$ *even though* P_{init} *has never been executed.*

At first sight, killing the whole process when applying the rule FAILURE may seem too strong. However, even if this kind of scenario is observable by the attacker, it does not bring him any new knowledge, hence it plays only a limited role in trace equivalence: it is in fact sufficient to consider such improper blocks only at the end of traces.

Example 3.5. *Consider* $\mathcal{P} = \{\ \mathtt{in}(c, x).\mathtt{in}(c, y),\ \mathtt{in}(c', x')\ \}$*. Its compressed traces are of the form* $\mathtt{in}(c, M).\mathtt{in}(c, N)$ *and* $\mathtt{in}(c', M')$*. The concatenation of those two improper traces cannot be executed in the compressed semantics. Intuitively, we do not loose anything for trace equivalence, because if a process can exhibit those two improper blocks they must be in parallel and hence considering their combination is redundant.*

We now define the notions of *compressed trace equivalence* (denoted $\approx_c$) and *compressed trace inclusion* (denoted $\sqsubseteq_c$), similarly to $\approx$ and $\sqsubseteq$ but relying on $\rightarrow_c$ instead of $\Rightarrow$.

Definition 3.6 (compressed trace equivalence)**.** *Let A and B be two extended simple processes. We have that $A \sqsubseteq_c B$ if, for every sequence of actions tr such that $A \xrightarrow{\mathsf{tr}}_c (\mathcal{P}; \Phi)$, there exists $(\mathcal{P}'; \Phi')$ such that $B \xrightarrow{\mathsf{tr}}_c (\mathcal{P}'; \Phi')$ and $\Phi \sim \Phi'$. The processes A and B are* compressed trace equivalent*, denoted by $A \approx_c B$, if $A \sqsubseteq_c B$ and $B \sqsubseteq_c A$.*

Example 3.7. *We have that $(\{P_{\mathsf{init}}, Q_0(skb, \mathsf{pk}(ska))\}; \Phi_0) \not\approx_c (\{P_{\mathsf{init}}, Q_0(skb, \mathsf{pk}(ska'))\}; \Phi_0)$. The trace $\mathtt{in}(c_B, \mathsf{aenc}(\langle w_1, w_1\rangle, w_2)) \cdot \mathtt{out}(c_B, w_3)$ exhibited in Example 3.4 is executable from $(\{P_{\mathsf{init}}, Q_0(skb, \mathsf{pk}(ska))\}; \Phi_0)$. However, this trace has no counterpart when starting with $(\{P_{\mathsf{init}}, Q_0(skb, \mathsf{pk}(ska'))\}; \Phi_0)$.*

3.2. **Soundness and completeness.** We shall now establish soundness and completeness of the compressed semantics. More precisely, we show that the two relations $\approx$ and $\approx_c$ coincide on initial simple processes (Theorem 3.11). All the proofs of this section are given in Appendix B.

Intuitively, we can always permute output (resp. input) actions occurring on distinct channels, and we can also permute an output with an input if the outputted message is not used to build the inputted term. More formally, we define an *independence relation $\mathcal{I}_a$ over actions* as the least symmetric relation satisfying:

- $\mathtt{out}(c_i, w_i)\ \mathcal{I}_a\ \mathtt{out}(c_j, w_j)$ and $\mathtt{in}(c_i, M_i)\ \mathcal{I}_a\ \mathtt{in}(c_j, M_j)$ as soon as $c_i \neq c_j$,
- $\mathtt{out}(c_i, w_i)\ \mathcal{I}_a\ \mathtt{in}(c_j, M_j)$ when in addition $w_i \notin fv(M_j)$.

Then, we consider $=_{\mathcal{I}_a}$ to be the least congruence (w.r.t. concatenation) satisfying:

$$\mathsf{act} \cdot \mathsf{act}' =_{\mathcal{I}_a} \mathsf{act}' \cdot \mathsf{act} \text{ for all } \mathsf{act} \text{ and } \mathsf{act}' \text{ with } \mathsf{act}\ \mathcal{I}_a\ \mathsf{act}',$$

and we show that processes are equally able to execute equivalent (w.r.t. $=_{\mathcal{I}_a}$) traces.

Lemma 3.8. *Let A, A' be two extended simple processes and tr, tr' be such that $\mathsf{tr} =_{\mathcal{I}_a} \mathsf{tr}'$. We have that $A \xRightarrow{\mathsf{tr}} A'$ if, and only if, $A \xRightarrow{\mathsf{tr}'} A'$.*

Now, considering traces that are only made of proper blocks, a strong relationship can be established between the two semantics.

Proposition 3.9. *Let A, A' be two simple extended processes, and tr be a trace made of proper blocks such that $A \xrightarrow{\mathsf{tr}}_c A'$. Then we have that $A \xRightarrow{\mathsf{tr}} A'$.*

Actually, the result stated in Proposition 3.9 immediately follows from the observation that $\rightarrow_c$ is included in $\Rightarrow$ for traces made of proper blocks since for them Failure cannot arise.

Proposition 3.10. *Let A, A' be two initial simple processes, and tr be a trace made of proper blocks such that $A \xRightarrow{\mathsf{tr}} A'$. Then, we have that $A \xrightarrow{\mathsf{tr}}_c A'$.*

This result is more involved and relies on the additional hypothesis that A and A' have to be initial to ensure that no Failure will arise.

Theorem 3.11. *Let A and B be two initial simple processes. We have that*

$$A \approx B \iff A \approx_c B.$$

Proof sketch, details in Appendix B. The main difficulty is that Proposition 3.10 only considers traces composed of proper blocks whereas we have to consider all traces. To prove the $\Rightarrow$ implication, we have to pay attention to the last block of the compressed trace that

can be an improper one (composed of several inputs on a channel c). The $\Leftarrow$ implication is more difficult since we have to consider a trace tr of a process A that is an interleaving of some prefix of proper and improper blocks. We will first complete it with tr^+ to obtain an interleaving of proper and improper blocks. We then reorder the actions to obtain a trace tr' such that $\mathsf{tr} \cdot \mathsf{tr}^+ =_{\mathcal{I}_a} \mathsf{tr}'$ and $\mathsf{tr}' = \mathsf{tr}_{\mathrm{io}} \cdot \mathsf{tr}_{\mathrm{in}}$ where $\mathsf{tr}_{\mathrm{io}}$ is made of proper blocks while $\mathsf{tr}_{\mathrm{in}}$ is made of improper blocks. For each improper block b of $\mathsf{tr}_{\mathrm{in}}$, we show by applying Lemma 3.8 and Proposition 3.10 that A is able to perform $\mathsf{tr}_{\mathrm{io}}$ in the compressed semantics and the resulting extended process can execute the improper block b. We thus have that A is able to perform $\mathsf{tr}_{\mathrm{io}} \cdot b$ in the compressed semantics and thus B as well. Finally, we show that the executions of all those (concurrent) blocks b can be put together, obtaining that B can perform tr', and thus tr as well. □

Note that, as illustrated by the following example, the two underlying notions of trace inclusion do *not* coincide.

Example 3.12. *Let* $\mathcal{P} = \{\mathtt{in}(c, x)\}$ *and* $\mathcal{Q} = \{\mathtt{in}(c, x).\mathtt{out}(c, n)\}$ *accompanied with an arbitrary frame* Φ*. We have* $(\mathcal{P}; \Phi) \sqsubseteq (\mathcal{Q}; \Phi)$ *but* $(\mathcal{P}; \Phi) \not\sqsubseteq_c (\mathcal{Q}; \Phi)$ *since in the compressed semantics* $(\mathcal{Q}; \Phi)$ *is not allowed to stop its execution after its first input.*

4. Deciding trace equivalence via constraint solving

In this section, we propose a symbolic semantics for our compressed semantics following, *e.g.* [39, 12]. Such a semantics avoids potentially infinite branching of our compressed semantics due to inputs from the environment. Correctness is maintained by associating with each process a set of constraints on terms.

4.1. Constraint systems.

Following the notations of [12], we consider a new set $\mathcal{X}^2$ of *second-order variables*, denoted by X, Y, etc. We shall use those variables to abstract over recipes. We denote by $fv^2(o)$ the set of free second-order variables of an object o, typically a constraint system. To prevent ambiguities, we shall use fv^1 instead of fv for free first-order variables.

Definition 4.1 (constraint system)**.** *A* constraint system $\mathcal{C} = (\Phi; \mathcal{S})$ *consists of a frame* Φ*, and a set of constraints* $\mathcal{S}$*. We consider three kinds of* constraints*:*

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

where $D \subseteq \mathcal{W}$*,* $X \in \mathcal{X}^2$*,* $x \in \mathcal{X}$ *and* $u, v \in \mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X})$*.*

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

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

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

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

Example 4.2. *Continuing Example 2.5, let* $\Phi = \Phi_0 \uplus \{w_3 \triangleright \mathsf{aenc}(\langle \pi_2(N), \langle n_b, \mathsf{pk}(skb)\rangle\rangle, \mathsf{pk}(ska))\}$ *with* $N = \mathsf{adec}(y, skb)$*, and* $\mathcal{S}$ *be a set containing two constraints:*

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

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

Our notion of well-formed constraint systems is in line with what is used, *e.g.* in [39, 12]. We use a simpler variant here that is sufficient for our purpose.

Definition 4.3 (solution)**.** *A* solution *of a constraint system* $\mathcal{C} = (\Phi; \mathcal{S})$ *is a substitution* θ *such that* $\mathrm{dom}(\theta) = fv^2(\mathcal{C})$*, and* $X\theta \in \mathcal{T}(\Sigma, D_{\mathcal{C}}(X))$ *for any* $X \in \mathrm{dom}(\theta)$*. Moreover, we require that there exists a ground substitution* λ *with* $\mathrm{dom}(\lambda) = fv^1(\mathcal{C})$ *such that:*

- *for every* $D \vdash^?_X x$ *in* $\mathcal{S}$*, we have* $(X\theta)(\Phi\lambda) =_{\mathsf{E}} x\lambda$*,* $valid((X\theta)(\Phi\lambda))$*, and* $valid(x\lambda)$*;*
- *for every* $u =^? v$ *in* $\mathcal{S}$*, we have* $u\lambda =_{\mathsf{E}} v\lambda$*,* $valid(u\lambda)$*, and* $valid(v\lambda)$*; and*
- *for every* $u \neq^? v$ *in* $\mathcal{S}$*, we have* $u\lambda \neq_{\mathsf{E}} v\lambda$*, or* $\neg valid(u\lambda)$*, or* $\neg valid(v\lambda)$*.*

Moreover, we require that all the terms occurring in $\Phi\lambda$ *are valid. The set of solutions of a constraint system* $\mathcal{C}$ *is denoted* $\mathsf{Sol}(\mathcal{C})$*. Since we consider constraint systems that are well-formed, the substitution* λ *is unique modulo* E *given* $\theta \in \mathsf{Sol}(\mathcal{C})$*. We denote it by* λ_θ *when* $\mathcal{C}$ *is clear from the context.*

Note that the validity constraints in the notion of solution of symbolic processes reflect the validity constraints of the concrete semantics (*i.e.* outputted and inputted terms must be valid and the equality between terms requires the two terms to be valid). Since we consider well-formed constraint systems, we may note that the substitution λ above is not obtained through unification. This substitution is entirely determined (modulo E) from θ by considering the deducibility constraints only.

Example 4.4. *Consider again the constraint system* $\mathcal{C}$ *given in Example 4.2. We have that* $\theta = \{Y \mapsto \mathsf{aenc}(\langle w_1, w_1\rangle, w_2)\}$ *is a solution of* $\mathcal{C}$*. Its associated first-order solution is* $\lambda_\theta = \{y \mapsto \mathsf{aenc}(\langle \mathsf{pk}(ska), \mathsf{pk}(ska)\rangle, \mathsf{pk}(skb))\}$*.*

4.2. **Symbolic processes: syntax and semantics.** Given an extended simple process $(\mathcal{P}; \Phi)$, we compute the constraint systems capturing its possible executions, starting from the symbolic process $(\mathcal{P}; \Phi; \emptyset)$. Note that we are now manipulating processes that are not ground anymore, but may contain free variables.

Definition 4.5 (symbolic process)**.** *A symbolic process is a tuple* $(\mathcal{P}; \Phi; \mathcal{S})$ *where* $(\Phi; \mathcal{S})$ *is a constraint system and* $fv^1(\mathcal{P}) \subseteq (fv^1(\mathcal{S}) \cap \mathcal{X})$*.*

We give in Figure 5 a standard symbolic semantics for symbolic basic processes. From this semantics given on symbolic basic processes only, we derive a semantics on simple symbolic processes in a natural way:

$$\frac{(P; \Phi; \mathcal{S}) \overset{\alpha}{\mapsto} (P'; \Phi'; \mathcal{S}')}{(\{P\} \uplus \mathcal{P}; \Phi; \mathcal{S}) \overset{\alpha}{\mapsto} (\{P'\} \uplus \mathcal{P}; \Phi'; \mathcal{S}')}$$

$$\textsc{In} \quad (\mathtt{in}(c,y).P;\Phi;\mathcal{S}) \xmapsto{\mathtt{in}(c,X)} (P\{y \mapsto x\};\Phi;\mathcal{S} \cup \{\mathrm{dom}(\Phi) \vdash^{?}_{X} x\})$$
where X (resp. x) is a fresh second-order (resp. first-order) variable

$$\textsc{Out} \quad (\mathtt{out}(c,u).P;\Phi;\mathcal{S}) \xmapsto{\mathtt{out}(c,w)} (P;\Phi \cup \{w \triangleright u\};\mathcal{S})$$
where w is a fresh first-order variable

$$\textsc{Then} \quad (\mathtt{if}\ u = v\ \mathtt{then}\ P\ \mathtt{else}\ Q;\Phi;\mathcal{S}) \xmapsto{\tau} (P;\Phi;\mathcal{S} \cup \{u =^{?} v\})$$

$$\textsc{Else} \quad (\mathtt{if}\ u = v\ \mathtt{then}\ P\ \mathtt{else}\ Q;\Phi;\mathcal{S}) \xmapsto{\tau} (Q;\Phi;\mathcal{S} \cup \{u \neq^{?} v\})$$

Figure 5: Symbolic semantics for symbolic basic processes

We can also derive our compressed symbolic semantics $\xmapsto{\mathsf{tr}}_c$ following the same pattern as for the concrete semantics (see Figure 6). We consider interleavings that execute maximal blocks of actions, and we allow improper termination of a block only at the end of a trace. Note that the $\neg valid(u)$ conditions of the third Proper rule and the second Improper rule are replaced by $u \neq^{?} u$ constraints in their symbolic counterparts.

$$\textsc{In} \quad \frac{(P;\Phi;\mathcal{S}) \xmapsto{\mathtt{in}(c,X)} (P';\Phi';\mathcal{S}') \quad (P';\Phi';\mathcal{S}') \xmapsto{\mathsf{tr}}\!\!\twoheadrightarrow_{i^*} (P'';\Phi'';\mathcal{S}'')}{(P;\Phi;\mathcal{S}) \xmapsto{\mathtt{in}(c,X).\mathsf{tr}}\!\!\twoheadrightarrow_{\ell} (P'';\Phi'';\mathcal{S}'')} \text{ with } \ell \in \{i^*; i^+\}$$

$$\textsc{Out} \quad \frac{(P;\Phi;\mathcal{S}) \xmapsto{\mathtt{out}(c,w)} (P';\Phi';\mathcal{S}') \quad (P';\Phi';\mathcal{S}') \xmapsto{\mathsf{tr}}\!\!\twoheadrightarrow_{o^*} (P'';\Phi'';\mathcal{S}'')}{(P;\Phi;\mathcal{S}) \xmapsto{\mathtt{out}(c,w).\mathsf{tr}}\!\!\twoheadrightarrow_{\ell} (P'';\Phi'';\mathcal{S}'')} \text{ with } \ell \in \{i^*; o^*\}$$

$$\textsc{Tau} \quad \frac{(P;\Phi;\mathcal{S}) \xmapsto{\tau} (P';\Phi';\mathcal{S}') \quad (P';\Phi';\mathcal{S}') \xmapsto{\mathsf{tr}}\!\!\twoheadrightarrow_{\ell} (P'';\Phi'';\mathcal{S}'')}{(P;\Phi;\mathcal{S}) \xmapsto{\mathsf{tr}}\!\!\twoheadrightarrow_{\ell} (P'';\Phi'';\mathcal{S}'')} \text{ with } \ell \in \{o^*; i^+; i^*\}$$

$$\textsc{Proper} \quad \frac{}{(0;\Phi;\mathcal{S}) \xmapsto{\epsilon}\!\!\twoheadrightarrow_{o^*} (0;\Phi;\mathcal{S})} \quad \frac{}{(\mathtt{in}(c,x).P;\Phi;\mathcal{S}) \xmapsto{\epsilon}\!\!\twoheadrightarrow_{o^*} (\mathtt{in}(c,x).P;\Phi;\mathcal{S})}$$

$$\frac{}{(\mathtt{out}(c,u).P;\Phi;\mathcal{S}) \xmapsto{\epsilon}\!\!\twoheadrightarrow_{o^*} (\mathtt{out}(c,u).P;\Phi;\mathcal{S} \cup \{u \neq^{?} u\})}$$

$$\textsc{Improper} \quad \frac{}{(0;\Phi;\mathcal{S}) \xmapsto{\epsilon}\!\!\twoheadrightarrow_{i^*} (\bot;\Phi;\mathcal{S})} \quad \frac{}{(\mathtt{out}(c,u).P;\Phi;\mathcal{S}) \xmapsto{\epsilon}\!\!\twoheadrightarrow_{i^*} (\bot;\Phi;\mathcal{S} \cup \{u \neq^{?} u\})}$$

$$\textsc{Block} \quad \frac{(Q;\Phi;\mathcal{S}) \xmapsto{\mathsf{tr}}\!\!\twoheadrightarrow_{i^+} (Q';\Phi';\mathcal{S}') \quad Q' \neq \bot}{(\{Q\} \uplus \mathcal{P};\Phi;\mathcal{S}) \xmapsto{\mathsf{tr}}_c (\{Q'\} \uplus \mathcal{P};\Phi';\mathcal{S}')}$$

$$\textsc{Failure} \quad \frac{(Q;\Phi;\mathcal{S}) \xmapsto{\mathsf{tr}}\!\!\twoheadrightarrow_{i^+} (Q';\Phi';\mathcal{S}') \quad Q' = \bot}{(\{Q\} \uplus \mathcal{P};\Phi;\mathcal{S}) \xmapsto{\mathsf{tr}}_c (\emptyset;\Phi';\mathcal{S}')}$$

Figure 6: Compressed symbolic semantics

Example 4.6. *Continuing Example 2.5, we have that* $(\{Q_0(skb, \mathsf{pk}(ska))\};\Phi_0;\emptyset) \xmapsto{\mathsf{tr}}_c (\emptyset;\Phi;\mathcal{S})$ *where:*

- $\mathsf{tr} = \mathtt{in}(c_B, Y) \cdot \mathtt{out}(c_B, w_3)$*, and*
- $\mathcal{C} = (\Phi;\mathcal{S})$ *is the constraint system defined in Example 4.2.*

We are now able to define the notion of equivalence associated to these two semantics, namely symbolic trace equivalence (denoted $\approx^s$) and symbolic compressed trace equivalence (denoted $\approx^s_c$). For a trace tr, we note $\mathrm{obs}(\mathsf{tr})$ the trace obtained from tr by removing all τ actions.

Definition 4.7. *Let $A = (\mathcal{P};\Phi)$ and $B = (\mathcal{Q};\Psi)$ be two simple processes. We have that $A \sqsubseteq^s B$ when, for every trace tr such that $(\mathcal{P};\Phi;\emptyset) \overset{\mathsf{tr}}{\longmapsto} (\mathcal{P}';\Phi';\mathcal{S}_A)$, for every $\theta \in \mathsf{Sol}(\Phi';\mathcal{S}_A)$, we have that:*

- *$(\mathcal{Q};\Psi;\emptyset) \overset{\mathsf{tr}'}{\longmapsto} (\mathcal{Q}';\Psi';\mathcal{S}_B)$ where $\mathrm{obs}(\mathsf{tr}') = \mathrm{obs}(\mathsf{tr})$ with $\theta \in \mathsf{Sol}(\Psi';\mathcal{S}_B)$, and*
- *$\Phi'\lambda^A_\theta \sim \Psi'\lambda^B_\theta$ where λ^A_θ (resp. λ^B_θ) is the substitution associated to θ w.r.t. $(\Phi';\mathcal{S}_A)$ (resp. $(\Psi';\mathcal{S}_B)$).*

We have that A and B are in trace equivalence w.r.t. $\mapsto$*, denoted $A \approx^s B$, if $A \sqsubseteq^s B$ and $B \sqsubseteq^s A$.*

We derive similarly the notion of trace equivalence induced by $\mapsto_c$. We do not have to take care of the τ actions since they are performed implicitly in the compressed semantics.

Definition 4.8. *Let $A = (\mathcal{P};\Phi)$ and $B = (\mathcal{Q};\Psi)$ be two extended simple processes. We have that $A \sqsubseteq^s_c B$ when, for every trace tr such that $(\mathcal{P};\Phi;\emptyset) \overset{\mathsf{tr}}{\longmapsto}_c (\mathcal{P}';\Phi';\mathcal{S}_A)$, for every $\theta \in \mathsf{Sol}(\Phi';\mathcal{S}_A)$, we have that:*

- *$(\mathcal{Q};\Psi;\emptyset) \overset{\mathsf{tr}}{\longmapsto}_c (\mathcal{Q}';\Psi';\mathcal{S}_B)$ with $\theta \in \mathsf{Sol}(\Psi';\mathcal{S}_B)$, and*
- *$\Phi'\lambda^A_\theta \sim \Psi'\lambda^B_\theta$ where λ^A_θ (resp. λ^B_θ) is the substitution associated to θ w.r.t. $(\Phi';\mathcal{S}_A)$ (resp. $(\Psi';\mathcal{S}_B)$).*

We have that A and B are in trace equivalence w.r.t. $\mapsto_c$*, denoted $A \approx^s_c B$, if $A \sqsubseteq^s_c B$ and $B \sqsubseteq^s_c A$.*

Example 4.9. *We have that $(\{Q_0(skb, \mathsf{pk}(ska))\};\Phi_0) \not\sqsubseteq^s_c (\{Q_0(skb, \mathsf{pk}(ska'))\};\Phi_0)$. Continuing Example 4.6, we have seen that:*

- *$(\{Q_0(skb, \mathsf{pk}(ska))\};\Phi_0;\emptyset) \overset{\mathsf{tr}}{\longmapsto}_c (\emptyset;\Phi;\mathcal{S})$ (see Example 4.6), and*
- *$\theta = \{Y \mapsto \mathsf{aenc}(\langle w_1, w_1\rangle, w_2)\}$ is a solution of $\mathcal{C} = (\Phi;\mathcal{S})$ (see Example 4.4).*

The only symbolic process that is reachable from $(\{Q_0(skb, \mathsf{pk}(ska'))\};\Phi_0;\emptyset)$ using tr is $(\emptyset;\Phi';\mathcal{S}')$ with:

- *$\Phi' = \Phi_0 \uplus \{w_3 \triangleright \mathsf{aenc}(\langle \pi_2(N), \langle n_b, \mathsf{pk}(skb)\rangle\rangle, \mathsf{pk}(ska'))\}$, and*
- *$\mathcal{S}' = \big\{\{w_0, w_1, w_2\} \vdash^?_Y y;\ \ \pi_2(N) =^? \mathsf{pk}(ska')\big\}$.*

One can check that θ is not a solution of $(\Phi';\mathcal{S}')$.

For processes without replication, the symbolic transition system induced by $\mapsto$ (resp $\mapsto_c$) is essentially finite. Indeed, the choice of fresh names for handles and second-order variables does not matter, and therefore the relations $\mapsto$ and $\mapsto_c$ are essentially finitely branching. Moreover, the length of traces of a simple process is obviously bounded. Thus, deciding (symbolic) trace equivalence between processes boils down to the problem of deciding a notion of equivalence between sets of constraint systems. This problem is well-studied and several procedures already exist [12, 24], *e.g.* Apte [20] (see Section 6).

4.3. **Soundness and completeness.** It is well-known that the symbolic semantics $\mapsto$ is sound and complete w.r.t. $\rightarrow$, and therefore that the two underlying notions of equivalence, namely $\approx^s$ and $\approx$, coincide. This has been proved for instance in [12, 22]. Using the same approach, we can show soundness and completeness of our symbolic compressed semantics w.r.t. our concrete compressed semantics. We have:

- *Soundness*: each transition in the compressed symbolic semantics represents a set of transitions that can be done in the concrete compressed semantics.
- *Completeness*: each transition in the compressed semantics can be matched by a transition in the compressed symbolic semantics.

These results are formally expressed in Proposition 4.10 and Proposition 4.11 below. These propositions are simple consequences of similar propositions that link the (small-step) symbolic semantics and the (small-step) standard semantics. Lifting these results to the compressed semantics is straightforward since both semantics are built using exactly the same scheme (see Figures 3 and 6).

Proposition 4.10. *Let $(\mathcal{P};\Phi)$ be an extended simple process such that $(\mathcal{P};\Phi;\emptyset) \overset{\mathsf{tr}}{\longmapsto}_c (\mathcal{P}';\Phi';\mathcal{S}')$, and $\theta \in \mathsf{Sol}(\Phi';\mathcal{S}')$. We have that $(\mathcal{P};\Phi) \overset{\mathsf{tr}\theta}{\longrightarrow}_c (\mathcal{P}'\lambda;\Phi'\lambda)$ where λ is the first-order solution of $(\mathcal{P}';\Phi';\mathcal{S}')$ associated to θ.*

Proposition 4.11. *Let $(\mathcal{P};\Phi)$ be an extended simple process such that $(\mathcal{P};\Phi) \overset{\mathsf{tr}}{\longrightarrow}_c (\mathcal{P}';\Phi')$. There exists a symbolic process $(\mathcal{P}_s;\Phi_s;\mathcal{S})$, a solution $\theta \in \mathsf{Sol}(\Phi_s;\mathcal{S})$, and a sequence tr_s such that:*

- $(\mathcal{P};\Phi;\emptyset) \overset{\mathsf{tr}_s}{\longmapsto}_c (\mathcal{P}_s;\Phi_s;\mathcal{S})$;
- $(\mathcal{P}';\Phi') = (\mathcal{P}_s\lambda;\Phi_s\lambda)$; *and*
- $\mathsf{tr} = \mathsf{tr}_s\theta$

where λ is the first-order solution of $(\mathcal{P}_s;\Phi_s;\mathcal{S})$ associated to θ.

Finally, relying on these two results, we can establish that symbolic trace equivalence ($\approx^s_c$) exactly captures compressed trace equivalence ($\approx_c$). Actually, both inclusions can be established separately.

Theorem 4.12. *For any extended simple processes A and B, we have that:*

$$A \sqsubseteq_c B \iff A \sqsubseteq^s_c B.$$

As an immediate consequence of Theorem 3.11 and Theorem 4.12, we obtain that the relations $\approx$ and $\approx^s_c$ coincide.

Corollary 4.13. *For any initial simple processes A and B, we have that:*

$$A \approx B \iff A \approx^s_c B.$$

5. Reduction using dependency constraints

Unlike compression, which is essentially based on the input/output nature of actions, our second optimisation takes into account the exchanged messages. Let us first illustrate one simple instance of our optimisation and how dependency constraints [40] may be used to incorporate it into symbolic semantics.

Example 5.1. *Let* $P_i = \mathtt{in}(c_i, x_i).\mathtt{out}(c_i, u_i).P_i'$ *with* $i \in \{1, 2\}$*, and* $\Phi_0 = \{w_0 \triangleright n\}$ *be a ground frame. We consider the simple process* $A = (\{P_1, P_2\}; \Phi_0)$*, and the two symbolic interleavings depicted in Figure 7. The two resulting symbolic processes are of the form* $(\{P_1', P_2'\}; \Phi; \mathcal{S}_i)$ *where* $\Phi = \Phi_0 \uplus \{w_1 \triangleright u_1, w_2 \triangleright u_2\}$*,*

$$\mathcal{S}_1 = \left\{w_0 \vdash^?_{X_1} x_1;\ w_0, w_1 \vdash^?_{X_2} x_2\right\},\ \text{and}\ \mathcal{S}_2 = \left\{w_0 \vdash^?_{X_2} x_2;\ w_0, w_2 \vdash^?_{X_1} x_1\right\}.$$

The sets of concrete processes that these two symbolic processes represent are different, which means that we cannot discard any of those interleavings. However, these sets have a significant overlap corresponding to concrete instances of the interleaved blocks that are actually independent, i.e. *where the output of one block is not necessary to obtain the input of the next block. In order to avoid considering such concrete processes twice, we may add a* dependency constraint $X_1 \ltimes w_2$ *in* $\mathcal{S}_2$*, whose purpose is to discard all solutions* θ *such that the message* $x_1\lambda_\theta$ *can be derived without using* $w_2 \triangleright u_2\lambda_\theta$*. For instance, the concrete trace* $\mathtt{in}(c_2, w_0) \cdot \mathtt{out}(c_2, w_2) \cdot \mathtt{in}(c_1, w_0) \cdot \mathtt{out}(c_1, w_1)$ *would be discarded thanks to this new constraint.*

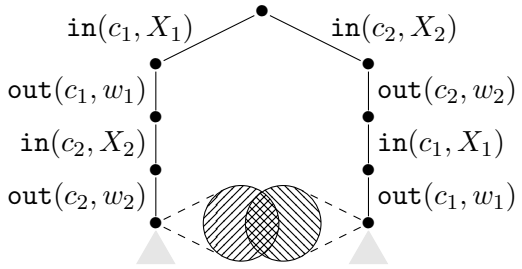


Figure 7: Two symbolic compressed traces (Example 5.1)

The idea of [40] is to accumulate dependency constraints generated whenever such a pattern is detected in an execution, and use an adapted constraint resolution procedure to narrow and eventually discard the constrained symbolic states. We seek to exploit similar ideas for optimising the verification of trace equivalence rather than reachability. This requires extra care, since pruning traces as described above may break completeness when considering trace equivalence. As before, the key to obtain a valid optimisation will be to discard traces in a similar way on the two processes being compared. In addition to handling this necessary subtlety, we also propose a new proof technique for justifying dependency constraints. The generality of that technique allows us to add more dependency constraints, taking into account more patterns than the simple one from the previous example.

5.1. **Reduced semantics.** We start by introducing *dependency constraints.*

Definition 5.2. *A* dependency constraint *is a constraint of the form* $\overrightarrow{X} \ltimes \overrightarrow{w}$ *where* $\overrightarrow{X}$ *is a vector of second-order variables in* $\mathcal{X}^2$*, and* $\overrightarrow{w}$ *is a vector of handles,* i.e. *variables in* $\mathcal{W}$*.*

Given a constraint system $\mathcal{C} = (\Phi; \mathcal{S})$*, a set* $\mathcal{S}_D$ *of dependency constraints, and* $\theta \in \mathsf{Sol}(\mathcal{C})$*. We write* $\theta \models_{(\Phi;\mathcal{S})} \mathcal{S}_D$ *when* θ *also satisfies the dependency constraints in* $\mathcal{S}_D$*,* i.e. *when for each* $\overrightarrow{X} \ltimes \overrightarrow{w} \in \mathcal{S}_D$ *there is some* $X_i \in \overrightarrow{X}$ *such that for all recipes* $M \in \mathcal{T}(\Sigma, D_{\mathcal{C}}(X_i))$ *satisfying* $M(\Phi\lambda_\theta) =_{\mathsf{E}} (X_i\theta)(\Phi\lambda_\theta)$ *and valid*$(M(\Phi\lambda_\theta))$*, we have that* $fv^1(M) \cap \overrightarrow{w} \neq \emptyset$ *where* λ_θ *is the substitution associated to* θ *w.r.t.* $(\Phi; \mathcal{S})$*.*

Intuitively, a dependency constraint $\overrightarrow{X} \ltimes \overrightarrow{w}$ is satisfied as soon as at least one message among those in $(\overrightarrow{X}\theta)(\Phi\lambda_\theta)$ can only be deduced by using a message stored in $\overrightarrow{w}$.

Example 5.3. *Continuing Example 5.1, assume that $u_1 = u_2 = n$ and let $\theta = \{X_1 \mapsto w_2; X_2 \mapsto w_0\}$. We have that $\theta \in \mathsf{Sol}(\mathcal{C}_2)$ and the substitution associated to θ w.r.t. $\mathcal{C}_2$ is $\lambda^2_\theta = \{x_1 \mapsto n; x_2 \mapsto n\}$. However, θ does not satisfy the dependency constraint $X_1 \ltimes w_2$. Indeed, we have that $w_0(\Phi\lambda^2_\theta) =_{\mathsf{E}} (X_1\theta)(\Phi\lambda^2_\theta)$ whereas $\{w_0\} \cap \{w_2\} = \emptyset$. Intuitively, this means that there is no good reason to postpone the execution of the block on channel c_1 if the output on c_2 is not useful to build the message used in input on c_1.*

We shall now define formally how dependency constraints will be added to our constraint systems. For this, we fix an arbitrary total order $\prec$ on channels. Intuitively, this order expresses which executions should be favored, and which should be allowed only under dependency constraints. To simplify the presentation, we use the notation $\mathtt{io}_c(\overrightarrow{X}, \overrightarrow{w})$ as a shortcut for $\mathtt{in}(c, X_1)\cdot\ldots\cdot\mathtt{in}(c, X_\ell)\cdot\mathtt{out}(c, w_1)\cdot\ldots\cdot\mathtt{out}(c, w_k)$ assuming that $\overrightarrow{X} = (X_1, \ldots, X_\ell)$ and $\overrightarrow{w} = (w_1, \ldots, w_k)$. Note that $\overrightarrow{X}$ and/or $\overrightarrow{w}$ may be empty.

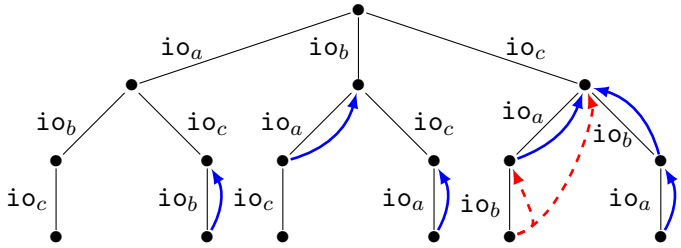
Definition 5.4 (generation of dependency constraints)**.** *Let c be a channel, and $\mathsf{tr} = \mathtt{io}_{c_1}(\overrightarrow{X_1}, \overrightarrow{w_1}) \cdot \ldots \cdot \mathtt{io}_{c_n}(\overrightarrow{X_n}, \overrightarrow{w_n})$ be a trace. If there exists a rank $k \leq n$ such that $c \prec c_k$ and $c_i \prec c$ for all $k < i \leq n$, then $\mathrm{dep}(\mathsf{tr}, c) = \{\, w \mid w \in \overrightarrow{w_i}$ with $k \leq i \leq n\}$. Otherwise, we have that $\mathrm{dep}(\mathsf{tr}, c) = \emptyset$.*

Then, given a trace tr, we define $\mathrm{Deps}(\mathsf{tr})$ by $\mathrm{Deps}(\epsilon) = \emptyset$ and

$$\mathrm{Deps}\left(\mathsf{tr} \cdot \mathtt{io}_c(\overrightarrow{X}, \overrightarrow{w})\right) = \begin{cases} \mathrm{Deps}(\mathsf{tr}) \cup \{\overrightarrow{X} \ltimes \mathrm{dep}(\mathsf{tr}, c)\} & \text{if } \mathrm{dep}(\mathsf{tr}, c) \neq \emptyset \\ \mathrm{Deps}(\mathsf{tr}) & \text{otherwise} \end{cases}$$

Intuitively, $\mathrm{Deps}(\mathsf{tr})$ corresponds to the accumulation of the dependency constraints generated for all prefixes of tr.

Example 5.5. *Let a, b, and c be channels in $\mathcal{C}$ such that $a \prec b \prec c$. The dependency constraints generated during the symbolic execution of a simple process of the form $(\{\mathtt{in}(a, x_a).\mathtt{out}(a, u_a), \mathtt{in}(b, x_b).\mathtt{out}(b, u_b), \mathtt{in}(c, x_c).\mathtt{out}(c, u_c)\}; \Phi)$ are depicted below.*



We use io_i as a shortcut for $\mathtt{in}(i, X_i)\cdot\mathtt{out}(i, w_i)$ and we represent dependency constraints using arrows. For instance, on the trace $\mathsf{io}_a \cdot \mathsf{io}_c \cdot \mathsf{io}_b$, a dependency constraint of the form $X_b \ltimes w_c$ (represented by the left-most arrow) is generated. Now, on the trace $\mathsf{io}_c \cdot \mathsf{io}_a \cdot \mathsf{io}_b$ we add $X_a \ltimes w_c$ after the second transition, and $X_b \ltimes \{w_c, w_a\}$ (represented by the dashed 2-arrow) after the third transition. Intuitively, the latter constraint expresses that io_b is only allowed to come after io_c if it depends on it, possibly indirectly through io_a.

Dependency constraints give rise to a new notion of trace equivalence, which further refines the previous ones.

Definition 5.6 (reduced trace equivalence)**.** *Let $A = (\mathcal{P}; \Phi)$ and $B = (\mathcal{Q}; \Psi)$ be two extended simple processes. We have that $A \sqsubseteq_r^s B$ when, for every sequence tr such that $(\mathcal{P}; \Phi; \emptyset) \xmapsto{\mathsf{tr}}_c (\mathcal{P}'; \Phi'; \mathcal{S}_A)$, for every $\theta \in \mathsf{Sol}(\Phi'; \mathcal{S}_A)$ such that $\theta \models_{(\Phi'; \mathcal{S}_A)} \mathrm{Deps}(\mathsf{tr})$, we have that:*

- *$(\mathcal{Q}; \Psi; \emptyset) \xmapsto{\mathsf{tr}}_c (\mathcal{Q}'; \Psi'; \mathcal{S}_B)$ with $\theta \in \mathsf{Sol}(\Psi'; \mathcal{S}_B)$, and $\theta \models_{(\Psi'; \mathcal{S}_B)} \mathrm{Deps}(\mathsf{tr})$;*
- *$\Phi' \lambda_\theta^A \sim \Psi' \lambda_\theta^B$ where λ_θ^A (resp. λ_θ^B) is the substitution associated to θ w.r.t. $(\Phi'; \mathcal{S}_A)$ (resp. $(\Psi'; \mathcal{S}_B)$).*

We have that A and B are in reduced trace equivalence*, denoted $A \approx_r^s B$, if $A \sqsubseteq_r^s B$ and $B \sqsubseteq_r^s A$.*

5.2. **Soundness and completeness.** In order to establish that $\approx_c^s$ and $\approx_r^s$ coincide, we shall study more carefully concrete traces, consisting of proper blocks possibly followed by a single improper block. We will then define a precise characterization of executions whose associated solution satisfies dependency constraints. We denote by $\mathcal{B}$ the set of blocks $\mathtt{io}_c(\overrightarrow{M}, \overrightarrow{w})$ such that $c \in \mathcal{C}$, $M_i \in \mathcal{T}(\Sigma, \mathcal{W})$ for each $M_i \in \overrightarrow{M}$, and $w_j \in \mathcal{W}$ for each $w_j \in \overrightarrow{w}$. In this section, a concrete trace is seen as a sequence of blocks, *i.e.* it belongs to $\mathcal{B}^*$.

Definition 5.7 (independence between blocks)**.** *Two blocks $b_1 = \mathtt{io}_{c_1}(\overrightarrow{M_1}, \overrightarrow{w_1})$ and $b_2 = \mathtt{io}_{c_2}(\overrightarrow{M_2}, \overrightarrow{w_2})$ are* independent*, written $b_1 \mid\mid b_2$, when $c_1 \neq c_2$ and none of the variables of $\overrightarrow{w_2}$ occurs in $\overrightarrow{M_1}$, and none of the variables of $\overrightarrow{w_1}$ occurs in $\overrightarrow{M_2}$. Otherwise the blocks are* dependent*.*

It is easy to see that independent blocks that are proper can be permuted in a compressed trace without affecting the executability and the result of executing that trace. It is not the case for improper blocks, which can only be performed at the very end of a compressed execution.

However, this notion of independence based on recipes is too restrictive: it may introduce spurious dependencies. Indeed, it is often possible to make two blocks dependent by slightly modifying recipes without altering the inputted messages. For instance, w' does not occur in recipe $M = w$ but does in $M' = \pi_1(\langle w, w' \rangle)$ while M' induces the same message as M. We thus define a more permissive notion of equivalence over traces, which allows permutations of independent blocks but also changes of recipes that preserve messages. During these permutations, we require that (concrete) traces remain *plausible*.

Definition 5.8 (plausible)**.** *A trace tr is* plausible *if for any input $\mathtt{in}(c, M)$ such that $\mathsf{tr} = \mathsf{tr}_0 \cdot \mathtt{in}(c, M) \cdot \mathsf{tr}_2$, we have $M \in \mathcal{T}(\Sigma, \mathcal{W}_0)$ where $\mathcal{W}_0$ is the set of handles occurring in tr_0.*

Given two blocks $b_1 = \mathtt{io}_{c_1}(\overrightarrow{M_1}, \overrightarrow{w_1})$ and $b_2 = \mathtt{io}_{c_2}(\overrightarrow{M_2}, \overrightarrow{w_2})$, we note $(b_1 =_{\mathsf{E}} b_2)\Phi$ when $\overrightarrow{M_1}\Phi =_{\mathsf{E}} \overrightarrow{M_2}\Phi$, $\mathit{valid}(M_1\Phi)$, $\mathit{valid}(M_2\Phi)$, and $\overrightarrow{w_1} = \overrightarrow{w_2}$. Intuitively, the two blocks only differ by a change of recipes such that the underlying messages are kept unchanged. We lift this notion to sequences of blocks, *i.e.* $(\mathsf{tr} =_{\mathsf{E}} \mathsf{tr}')\Phi$, in the natural way.

Definition 5.9. *Given a frame Φ, the relation $\equiv_\Phi$ is the smallest equivalence over plausible traces (made of blocks) such that:*

(1) $\mathsf{tr} \cdot b_1 \cdot b_2 \cdot \mathsf{tr}' \equiv_\Phi \mathsf{tr} \cdot b_2 \cdot b_1 \cdot \mathsf{tr}'$ *when $b_1 \mid\mid b_2$; and*
(2) $\mathsf{tr} \cdot b_1 \cdot \mathsf{tr}' \equiv_\Phi \mathsf{tr} \cdot b_2 \cdot \mathsf{tr}'$ *when $(b_1 =_{\mathsf{E}} b_2)\Phi$.*

Lemma 5.10. *Let* $A \xrightarrow{\mathsf{tr}}_c (\mathcal{P};\Phi)$ *with* tr *be a trace made of proper blocks. We have that* $A \xrightarrow{\mathsf{tr}'}_c (\mathcal{P};\Phi)$ *for any* $\mathsf{tr}' \equiv_\Phi \mathsf{tr}$.

This result is easily proved, following from the fact that proper compressed executions are preserved by the two generators of $\equiv_\Phi$. The first case is given by Lemma 3.8. The second one follows from a simple observation of the transition rules: only the derived messages matter, while the recipes that are used to derive them are irrelevant (as long as validity is ensured).

We established that compressed executions are preserved by changes of traces within $\equiv_\Phi$-equivalence classes. We shall now prove that, by keeping only executions satisfying dependency constraints, we actually select exactly one representative in this class.

We lift the ordering on channels to blocks: $\mathtt{io}_c(\overrightarrow{M},\overrightarrow{w}) \prec \mathtt{io}_{c'}(\overrightarrow{M'},\overrightarrow{w'})$ if and only if $c \prec c'$. Finally, we define $\prec$ on concrete traces as the lexicographic extension of the order on blocks. Given a frame Φ, we say that a plausible trace tr is Φ-minimal if it is minimal in its equivalence class modulo $\equiv_\Phi$.

Lemma 5.11. *Let* $A \overset{\mathsf{tr}}{\mapsto}_c (\mathcal{P};\Phi;\mathcal{S})$ *and* $\theta \in \mathsf{Sol}(\Phi;\mathcal{S})$. *We have that* $\mathsf{tr}\theta$ *is* $\Phi\lambda_\theta$*-minimal if, and only if,* $\theta \models_{(\Phi;\mathcal{S})} \mathrm{Deps}(\mathsf{tr})$.

Proof. Let A and $(\mathcal{P};\Phi;\mathcal{S})$ be such that $A \overset{\mathsf{tr}}{\mapsto}_c (\mathcal{P};\Phi;\mathcal{S})$ and $\theta \in \mathsf{Sol}(\Phi;\mathcal{S})$. Let λ_θ be the substitution associated to θ w.r.t. $(\Phi;\mathcal{S})$.

($\Rightarrow$) We first show that if $\mathsf{tr}\theta$ is $\Phi\lambda_\theta$-minimal then $\theta \models_{(\Phi;\mathcal{S})} \mathrm{Deps}(\mathsf{tr})$, by induction on the length of the trace tr. The base case, *i.e.* $\mathsf{tr} = \epsilon$, is straightforward since $\mathrm{Deps}(\mathsf{tr}) = \emptyset$. Now, assume that $\mathsf{tr} = \mathsf{tr}_0 \cdot b$ for some block b and $A \overset{\mathsf{tr}_0}{\mapsto}_c (\mathcal{P}_0;\Phi_0;\mathcal{S}_0) \overset{b}{\mapsto}_c (\mathcal{P};\Phi;\mathcal{S})$. Let θ_0 be the substitution θ restricted to variables occurring in $(\Phi_0;\mathcal{S}_0)$, and λ_{θ_0} be the associated first-order substitution. We have that $\theta_0 \in \mathsf{Sol}(\Phi_0;\mathcal{S}_0)$, λ_{θ_0} coincides with λ_θ on variables occurring in $(\Phi_0;\mathcal{S}_0)$, and $\Phi_0\lambda_{\theta_0}$ coincides with $\Phi\lambda_\theta$ on the domain of $\Phi_0\lambda_{\theta_0}$. As a prefix of $\mathsf{tr}\theta$, we have that $\mathsf{tr}_0\theta$ is $\Phi\lambda_\theta$-minimal. We can thus apply our induction hypothesis on $A \overset{\mathsf{tr}_0}{\mapsto}_c (\mathcal{P}_0;\Phi_0;\mathcal{S}_0)$ and $\theta_0 \in \mathsf{Sol}(\Phi_0;\mathcal{S}_0)$. Assume that $b = \mathtt{io}_c(\overrightarrow{X},\overrightarrow{w})$. If $\mathrm{dep}(\mathsf{tr}_0,c) = \emptyset$, we immediately conclude. Otherwise, it only remains to show that $\theta \models_{(\Phi;\mathcal{S})} \overrightarrow{X} \ltimes \mathrm{dep}(\mathsf{tr}_0,c)$. By definition of the generation of dependency constraints, we know that tr_0 is of the form $\mathsf{tr}'_0 \cdot b_{c_0} \cdot b_{c_1} \cdot \ldots \cdot b_{c_n}$ where:

- $\forall\, 0 \le i \le n$, $b_{c_i} = \mathsf{io}_{c_i}(\overrightarrow{X_i},\overrightarrow{w_i})$,
- $c \prec c_0$ and $c_i \prec c$ for all $0 < i \le n$; and
- $\mathrm{dep}(\mathsf{tr},c) = \{w \mid w \in \overrightarrow{w_i} \text{ with } 0 \le i \le n\}$.

Assume that the dependency constraint is not satisfied, this means that for some $\overrightarrow{M}$ such that $(\overrightarrow{X}\theta)(\Phi\lambda_\theta) =_{\mathsf{E}} (\overrightarrow{M})(\Phi\lambda_\theta)$ and $\mathit{valid}((\overrightarrow{M})(\Phi\lambda_\theta))$, we have that $fv^1(\overrightarrow{M}) \cap \{w \mid w \in \overrightarrow{w_i} \text{ with } 0 \le i \le n\} = \emptyset$. Therefore, we have that

$$\begin{array}{rcl} \mathsf{tr}\theta & = & \mathsf{tr}_0\theta \cdot b\theta \\ & = & \mathsf{tr}'_0\theta \cdot b_{c_0}\theta \cdot b_{c_1}\theta \cdot \ldots \cdot b_{c_n}\theta \cdot \mathtt{io}_c(\overrightarrow{X\theta},\overrightarrow{w}) \\ & \equiv_{\Phi\lambda_\theta} & \mathsf{tr}'_0\theta \cdot \mathtt{io}_c(\overrightarrow{M},\overrightarrow{w}) \cdot b_{c_0}\theta \cdot b_{c_1}\theta \cdot \ldots \cdot b_{c_n}\theta. \end{array}$$

Since $c \prec c_0$, this would contradict the $\Phi\lambda_\theta$-minimality of $\mathsf{tr}\theta$. Hence the result.

($\Leftarrow$) Now, assuming that $\mathsf{tr}\theta$ is not $\Phi\lambda_\theta$-minimal, we shall establish that there is a dependency constraint $\overrightarrow{X} \ltimes \overrightarrow{w} \in \mathrm{Deps}(\mathsf{tr})$ that is not satisfied by θ. Let tr_m be a $\Phi\lambda_\theta$-minimal trace of the equivalence class of $\mathsf{tr}\theta$. We have in particular $\mathsf{tr}_m \equiv_{\Phi\lambda_\theta} \mathsf{tr}\theta$ and $\mathsf{tr}_m \prec \mathsf{tr}\theta$.

Let tr_0 (resp. tr^0_m) be the longest prefix of tr (resp. tr_m) such that $(\mathsf{tr}_0\theta =_{\mathsf{E}} \mathsf{tr}^0_m)\Phi\lambda_\theta$. We have that $\mathsf{tr} = \mathsf{tr}_0 \cdot b \cdot \mathsf{tr}_1$ and $\mathsf{tr}_m = \mathsf{tr}^0_m \cdot b_m \cdot \mathsf{tr}^1_m$ with $c_m \prec c$ where c_m (resp. c) is the channel used in block b_m (resp. b). By definition of $\equiv_{\Phi\lambda_\theta}$, block b_m must have a counterpart in tr and, more precisely, in tr_1. We thus have a more precise decomposition of tr: $\mathsf{tr} = \mathsf{tr}_0 \cdot b \cdot \mathsf{tr}_{11} \cdot b'_m \cdot \mathsf{tr}_{12}$ such that $(b'_m\theta =_{\mathsf{E}} b_m)\Phi\lambda_\theta$.

Let $b'_m = \mathtt{io}_{c_m}(\overrightarrow{X}, \overrightarrow{w})$. We now show that the constraint $\overrightarrow{X} \ltimes \mathsf{dep}(\mathsf{tr}_0 \cdot b \cdot \mathsf{tr}_{11}, c_m)$ is in $\mathsf{Deps}(\mathsf{tr})$ and is not satisfied by θ, implying $\theta \not\models_{(\Phi;\mathcal{S})} \mathsf{Deps}(\mathsf{tr})$. We have seen that $(b'_m\theta =_{\mathsf{E}} b_m)\Phi\lambda_\theta$ and $c_m \prec c$. Since $c_m \prec c$, by definition of $\mathsf{dep}(\cdot,\cdot)$ we deduce that $\emptyset \neq \mathsf{dep}(\mathsf{tr}_0 \cdot b \cdot \mathsf{tr}_{11}, c_m) \subseteq \{\ w \mid \mathtt{out}(d, w) \text{ occurs in } b \cdot \mathsf{tr}_{11} \text{ for some } d, w\ \}$. But, since we also know that $(b'_m\theta =_{\mathsf{E}} b_m)\Phi\lambda_\theta$ and $\mathsf{tr}_m = \mathsf{tr}^0_m \cdot b_m \cdot \mathsf{tr}^1_m$ is a plausible trace, we have that $b_m = \mathtt{io}_{c_m}(\overrightarrow{M}, \overrightarrow{w_m})$ for some recipes M such that $\overrightarrow{M}\Phi\lambda_\theta =_{\mathsf{E}} (\overrightarrow{X}\theta)\Phi\lambda_\theta$, $\mathit{valid}(\overrightarrow{M}\Phi\lambda_\theta)$, and $fv^1(M) \cap \mathsf{dep}(\mathsf{tr}_0 \cdot b \cdot \mathsf{tr}_{11}, c_m) = \emptyset$. This allows us to conclude that $\theta \not\models_{(\Phi;\mathcal{S})} \mathsf{Deps}(\mathsf{tr})$. □

We are now able to show that the notion of trace equivalence based on this reduced semantics coincides with the compressed one (as well as its symbolic counterpart as given in Definition 4.8). Even though the reduced semantics is based on the symbolic compressed semantics, it is more natural to establish the theorem by going back to the concrete compressed semantics, because we have to consider a concrete execution to check whether dependency constraints are satisfied or not in our reduced semantics anyway.

Theorem 5.12. *For any extended simple processes A and B, we have that:*

$$A \sqsubseteq_c B \text{ if and only if } A \sqsubseteq^s_r B.$$

Proof. Let $A = (\mathcal{P}; \Phi)$ and $B = (\mathcal{Q}; \Psi)$ be two extended simple processes.

($\Rightarrow$) Consider an execution of the form $(\mathcal{P}; \Phi; \emptyset) \xrightarrow{\mathsf{tr}_s}_c (\mathcal{P}_s; \Phi_s; \mathcal{S}_A)$ and a substitution $\theta \in \mathsf{Sol}(\Phi_s; \mathcal{S}_A)$ such that $\theta \models_{(\Phi_s;\mathcal{S}_A)} \mathsf{Deps}(\mathsf{tr}_s)$. Thanks to Proposition 4.10, we have that $(\mathcal{P}; \Phi) \xrightarrow{\mathsf{tr}_s\theta}_c (\mathcal{P}_s\lambda^A_\theta; \Phi_s\lambda^A_\theta)$ where λ^A_θ is the substitution associated to θ w.r.t. $(\Phi_s; \mathcal{S}_A)$.

Since $A \sqsubseteq_c B$, we deduce that there exists $(\mathcal{Q}'; \Psi')$ such that:

$$B \stackrel{\mathsf{def}}{=} (\mathcal{Q}; \Psi) \xrightarrow{\mathsf{tr}_s\theta}_c (\mathcal{Q}'; \Psi') \text{ and } \Phi_s\lambda^A_\theta \sim \Psi'$$

Relying on Proposition 4.11, we deduce that there exists $(\mathcal{Q}_s; \Psi_s; \mathcal{S}_B)$ such that:

$$(\mathcal{Q}; \Psi; \emptyset) \xrightarrow{\mathsf{tr}_s}_c (\mathcal{Q}_s; \Psi_s; \mathcal{S}_B), \theta \in \mathsf{Sol}(\Psi_s; \mathcal{S}_B) \text{ and } (\mathcal{Q}_s\lambda^B_\theta; \Psi_s\lambda^B_\theta) = (\mathcal{Q}'; \Psi')$$

where λ^B_θ is the substitution associated to θ w.r.t. $(\Psi_s; \mathcal{S}_B)$. The fact that we get the same symbolic trace tr_s and same solution θ comes from the third point of Proposition 4.11 and the flexibility of the symbolic semantics $\dot{\mapsto}_c$ that allows us to choose second order variables of our choice (as long as they are fresh).

Lemma 5.11 tells us that $\mathsf{tr}\theta$ is $\Phi_s\lambda^A_\theta$-minimal. Since $\Phi_s\lambda^A_\theta \sim \Psi_s\lambda^B_\theta$, we easily deduce that $\mathsf{tr}_s\theta$ is also $\Psi_s\lambda^B_\theta$-minimal, and thus Lemma 5.11 tells us that $\theta \models_{(\Psi_s;\mathcal{S}_B)} \mathsf{Deps}(\mathsf{tr})$. This allows us to conclude.

($\Leftarrow$) Consider an execution of the form $(\mathcal{P}; \Phi) \xrightarrow{\mathsf{tr}}_c (\mathcal{P}'; \Phi')$. We prove the result by induction on the number of blocks involved in tr, and we distinguish two cases depending on whether tr ends with an improper block or not.

Case where tr is made of proper blocks. Let tr_m be a Φ'-minimal trace in the equivalence class of tr. Lemma 5.10 tells us that $(\mathcal{P}; \Phi) \xrightarrow{\mathsf{tr}_m}_c (\mathcal{P}'; \Phi')$. Thanks to Proposition 4.11, we know that there exist $(\mathcal{P}_s; \Phi_s; \mathcal{S}_A)$, tr^s_m, and θ such that:

$$(\mathcal{P}; \Phi; \emptyset) \xrightarrow{\mathsf{tr}^s_m}_c (\mathcal{P}_s; \Phi_s; \mathcal{S}_A), \theta \in \mathsf{Sol}(\Phi_s; \mathcal{S}_A), (\mathcal{P}_s\lambda^A_\theta; \Phi_s\lambda^A_\theta) = (\mathcal{P}'; \Phi'), \text{ and } \mathsf{tr}^s_m\theta = \mathsf{tr}_m.$$

Using Lemma 5.11, we deduce that $\theta \models_{(\Phi_s;\mathcal{S}_A)} \mathrm{Deps}(\mathsf{tr}^s_m)$. By hypothesis, $A \sqsubseteq^s_r B$, hence:
- $(\mathcal{Q};\Psi;\emptyset) \xmapsto{\mathsf{tr}^s_m}_c (\mathcal{Q}_s;\Psi_s;\mathcal{S}_B)$ with $\theta \in \mathsf{Sol}(\Psi_s;\mathcal{S}_B)$, and $\theta \models_{(\Psi_s;\mathcal{S}_B)} \mathrm{Deps}(\mathsf{tr}^s_m)$;
- $\Phi_s\lambda^A_\theta \sim \Psi_s\lambda^B_\theta$ where λ^B_θ is the substitution associated to θ w.r.t. $(\Phi_s;\mathcal{S}_B)$

Thanks to Proposition 4.10, we deduce that

$$(\mathcal{Q};\Psi) \xrightarrow{\mathsf{tr}^s_m\theta}_c (\mathcal{Q}_s\lambda^B_\theta;\Psi_s\lambda^B_\theta).$$

Moreover, since $\Phi' = \Phi_s\lambda^A_\theta \sim \Psi_s\lambda^B_\theta$, we get $\mathsf{tr}_m \equiv_{\Psi_s\lambda^B_\theta} \mathsf{tr}$ from the fact that $\mathsf{tr}_m \equiv_{\Phi'} \mathsf{tr}$. Applying Lemma 5.10, we conclude that

$$(\mathcal{Q};\Psi) \xrightarrow{\mathsf{tr}}_c (\mathcal{Q}_s\lambda^B_\theta;\Psi_s\lambda^B_\theta) \text{ with } \Phi' \sim \Psi_s\lambda^B_\theta.$$

Case where tr is of the form $\mathsf{tr}_0 \cdot b$ where b is an improper block. We have that:

$$(\mathcal{P};\Phi) \xrightarrow{\mathsf{tr}_0}_c (\mathcal{P}';\Phi') \xrightarrow{b}_c (\emptyset;\Phi')$$

Let tr_m be a Φ'-minimal trace in the equivalence class of tr. By definition of the relation $\equiv$, block b must have a counterpart in tr_m. We thus have that tr_m is of the form $\mathsf{tr}_m = \mathsf{tr}_1 \cdot b_m \cdot \mathsf{tr}_2$ where b_m is the improper block corresponding to b. We do not necessarily have that $b = b_m$ but we know that $(b =_{\mathsf{E}} b_m)\Phi'$. If tr_2 is an empty trace, *i.e.* b_m is at the end of tr_m, the reasoning from the previous case applies.

Otherwise, we have that $\mathsf{tr}_1 \cdot \mathsf{tr}_2 \equiv_{\Phi'} \mathsf{tr}_0$, $\mathsf{tr}_1 \cdot \mathsf{tr}_2$ is Φ'-minimal, and tr_2 is non empty. Thus, thanks to Lemma 5.10, we have that:

$$(\mathcal{P};\Phi) \xrightarrow{\mathsf{tr}_1}_c (\mathcal{P}_1;\Phi_1) \xrightarrow{\mathsf{tr}_2}_c (\mathcal{P}';\Phi')$$

Since $\mathsf{tr}_1 \cdot \mathsf{tr}_2$ is made of proper blocks, we can apply our previous reasoning, and conclude that there exist $(\mathcal{Q}_1;\Psi_1)$ and $(\mathcal{Q}';\Psi')$ such that:

$$(\mathcal{Q};\Psi) \xrightarrow{\mathsf{tr}_1}_c (\mathcal{Q}_1;\Psi_1) \xrightarrow{\mathsf{tr}_2}_c (\mathcal{Q}';\Psi') \text{ and } \Phi' \sim \Psi' \text{ (and thus } (b =_{\mathsf{E}} b_m)\Psi').$$

Since we know that $(\mathcal{P}';\Phi') \xrightarrow{b}_c (\emptyset;\Phi')$, and $\mathsf{tr}_0 \cdot b \equiv_{\Phi'} \mathsf{tr}_1 \cdot b_m \cdot \mathsf{tr}_2$, we deduce that $(\mathcal{P}_1;\Phi_1) \xrightarrow{b_m}_c (\emptyset;\Phi_1)$. We have that $(\mathcal{P};\Phi) \xrightarrow{\mathsf{tr}_1 \cdot b_m}_c (\emptyset;\Phi_1)$, and $\mathsf{tr}_1 \cdot b_m$ is a Φ_1-minimal trace (note that the improper block is at the end). Thus, applying our induction hypothesis, we have that:

$$(\mathcal{Q};\Psi) \xrightarrow{\mathsf{tr}_1}_c (\mathcal{Q}_1;\Psi_1) \xrightarrow{b_m}_c (\emptyset;\Psi_1).$$

Since the channel used in b_m does not occur in tr_2, we deduce that

$$(\mathcal{Q};\Psi) \xrightarrow{\mathsf{tr}_1}_c (\mathcal{Q}_1;\Psi_1) \xrightarrow{\mathsf{tr}_2}_c (\mathcal{Q}';\Psi') \xrightarrow{b_m}_c (\emptyset;\Psi')$$

Relying on Lemma 5.10 and the fact that $(b_m =_{\mathsf{E}} b)\Psi'$, we deduce that

$$(\mathcal{Q};\Psi) \xrightarrow{\mathsf{tr}_0}_c (\mathcal{Q}';\Psi') \xrightarrow{b}_c (\emptyset;\Psi'). \qquad \square$$

Putting together Theorem 3.11 and Theorem 5.12, we are now able to state our main result: our notion of reduced trace equivalence actually coincides with the usual notion of trace equivalence. This result is generic and holds for an arbitrary equational theory, as well as for an arbitrary notion of validity (as defined in Section 2.1).

Corollary 5.13. *For any initial simple processes A and B, we have that:*

$$A \approx B \text{ if and only if } A \approx^s_r B.$$

6. Integration in Apte

We validate our approach by integrating our refined semantics in the Apte tool. As we shall see, the compressed semantics can easily be used as a replacement for the usual semantics in verification algorithms. However, exploiting the reduced semantics is not trivial, and requires to adapt the constraint resolution procedure.

It is beyond the scope of this paper to provide a detailed summary of how the verification tool Apte actually works. A 50 pages paper describing solely the constraint resolution procedure of Apte is available [21]. This procedure manipulates matrices of constraint systems, with additional kinds of constraints necessary for its inner workings. Proofs of the soundness, completeness and termination of the algorithm are available in a long and technical appendix (more than 100 pages).

In order to show how our reduced semantics have been integrated in the constraint solving procedure of Apte, we choose to provide a high-level axiomatic presentation of Apte's algorithm. This allows us to prove that our integration is correct without having to enter into complex, unnecessary details of Apte's algorithm. Our axioms are consequences of results stated and proved in [18] and have been written in concertation with Vincent Cheval. However, due to some changes in the presentation, proving them will require to adapt most of the proofs. It is therefore beyond the scope of this paper to formally prove that our axioms are satisfied by the concrete procedure.

We start this section with a high-level axiomatic presentation of Apte's algorithm, following the original procedure [20] but assuming public channels only (sections 6.1, 6.2). The purpose of this presentation is to provide enough details about Apte to explain how our optimisations have been integrated, leaving out unimportant details. Next, we show that this axiomatization is sufficient to prove soundness and completeness of Apte w.r.t. trace equivalence (Section 6.3). Then we explain the simplifications induced by the restriction to simple processes, and how compressed semantics can be used to enhance the procedure and prove the correctness of this integration (Section 6.4). We finally describe how our reduction technique can be integrated, and prove the correctness of this integration (Section 6.5). We present some benchmarks in Section 6.6, showing that our integration allows to effectively benefit from both of our partial order reduction techniques.

6.1. Apte in a nutshell.

Apte has been designed for a fixed equational theory $\mathsf{E}_{\mathsf{Apte}}$ (formally defined in Example 2.2) containing standard cryptographic primitives. It relies on a notion of message which requires that only constructors are used, and a semantics in which actions are blocked unless they are performed on such messages. This fits in our framework, described in Section 2, by taking $\mathcal{M} = \mathcal{T}(\Sigma_c, \mathcal{N})$.

We now give a high-level description of the algorithm that is implemented in Apte. The main idea is to perform all possible symbolic executions of the processes, keeping together the processes that can be reached using the same sequence of symbolic action. Then, at each step of this symbolic execution, the procedure checks that for every solution of every process on one side, there is a corresponding solution for some process on the other side so that the resulting frames are in static equivalence. This check for *symbolic equivalence* is not obviously decidable. To achieve it, Apte's procedure relies on a set of rules for simplifying sets of constraint systems. These rules are used to put constraint systems in a *solved form* that enables the efficient verification of symbolic equivalence.

The symbolic execution used in Apte is the same as described in Section 4. However, Apte's constraint resolution procedure introduces new kinds of constraints. Fortunately, we do not need to enter into the details of those constraints and how they are manipulated. Instead, we treat them axiomatically.

Definition 6.1 (extended constraint system/symbolic process)**.** *An* extended constraint system $\mathcal{C}^+ = (\Phi; \mathcal{S}; \mathcal{S}^+)$ *consists of a constraint system* $\mathcal{C} = (\Phi; \mathcal{S})$ *together with an additional set* $\mathcal{S}^+$ *of* extended constraints*. We treat this latter set abstractly, only assuming an associated satisfaction relation, written* $\theta \models \mathcal{S}^+$*, such that* $\theta \models \emptyset$ *always holds, and* $\theta \models \mathcal{S}_1^+$ *implies* $\theta \models \mathcal{S}_2^+$ *when* $\mathcal{S}_2^+ \subseteq \mathcal{S}_1^+$*. We define the set of solutions of* $\mathcal{C}^+$ *as* $\mathsf{Sol}^+(\mathcal{C}^+) = \{\, \theta \in \mathsf{Sol}(\mathcal{C}) \mid \theta \models \mathcal{S}^+ \,\}$.

An extended symbolic process $(\mathcal{P}; \Phi; \mathcal{S}; \mathcal{S}^+)$ *is a symbolic process with an additional set of extended constraints* $\mathcal{S}^+$*.*

We shall denote extended constraint systems by $\mathcal{S}^+$, $\mathcal{S}_1^+$, etc. Extended symbolic processes will be denoted by A^+, B^+, etc. Sets of extended symbolic processes will simply be denoted by $\mathbf{A}$, $\mathbf{B}$, etc. For convenience, we extend Sol and Sol^+ to symbolic processes and extended symbolic processes in the natural way:

$$\mathsf{Sol}(\mathcal{P}; \Phi; \mathcal{S}) = \mathsf{Sol}(\Phi; \mathcal{S}) \quad \text{and} \quad \mathsf{Sol}^+\left(\mathcal{P}; \Phi; \mathcal{S}; \mathcal{S}^+\right) = \mathsf{Sol}^+(\Phi; \mathcal{S}; \mathcal{S}^+).$$

We may also use the following notation to translate back and forth between symbolic processes and extended symbolic processes:

$$\lceil (\mathcal{P}; \Phi; \mathcal{S}) \rceil = (\mathcal{P}; \Phi; \mathcal{S}; \emptyset) \quad \text{and} \quad \lfloor \left(\mathcal{P}; \Phi; \mathcal{S}; \mathcal{S}^+\right) \rfloor = (\mathcal{P}; \Phi; \mathcal{S}).$$

We can now introduce the key notion of symbolic equivalence between sets of extended symbolic processes, or more precisely between their underlying extended constraint systems.

Definition 6.2 (symbolic equivalence)**.** *Given two sets of extended symbolic processes* $\mathbf{A}$ *and* $\mathbf{B}$*, we have that* $\mathbf{A} \prec^+ \mathbf{B}$ *if for every* $A^+ = \left(\mathcal{P}_A; \Phi_A; \mathcal{S}_A; \mathcal{S}_A^+\right) \in \mathbf{A}$*, for every* $\theta \in \mathsf{Sol}^+(A^+)$*, there exists* $B^+ = \left(\mathcal{P}_B; \Phi_B; \mathcal{S}_B; \mathcal{S}_B^+\right) \in \mathbf{B}$ *such that* $\theta \in \mathsf{Sol}^+(B^+)$ *and* $\Phi_A \lambda_\theta^A \sim \Phi_B \lambda_\theta^B$ *where* λ_θ^A *(resp.* λ_θ^B*) is the substitution associated to* θ *w.r.t.* $(\Phi_A; \mathcal{S}_A)$ *(resp.* $(\Phi_B; \mathcal{S}_B)$*). We say that* $\mathbf{A}$ *and* $\mathbf{B}$ *are in* symbolic equivalence*, denoted by* $\mathbf{A} \sim^+ \mathbf{B}$*, if* $\mathbf{A} \prec^+ \mathbf{B}$ *and* $\mathbf{B} \prec^+ \mathbf{A}$*.*

The whole trace equivalence procedure can finally be abstractly described by means of a transition system $\mapsto^{\mathsf{A}}$ on pairs of sets of extended symbolic processes, labelled by observable symbolic actions. Informally, the intent is that a pair of processes is in trace equivalence iff only symbolically equivalent pairs may be reached from the initial pair using $\mapsto^{\mathsf{A}}$.

We now define $\mapsto^{\mathsf{A}}$ formally. A transition $(\mathbf{A}; \mathbf{B}) \mapsto^{\mathsf{A}}$ can take place iff $\mathbf{A}$ and $\mathbf{B}$ are in symbolic equivalence[2].

Each transition for some observable action α consists of two steps, *i.e.* $(\mathbf{A}; \mathbf{B}) \overset{\alpha}{\mapsto}^{\mathsf{A}} (\mathbf{A}''; \mathbf{B}'')$ iff $(\mathbf{A}; \mathbf{B}) \overset{\alpha}{\mapsto}^{\mathsf{A1}} (\mathbf{A}'; \mathbf{B}')$ and $(\mathbf{A}'; \mathbf{B}') \mapsto^{\mathsf{A2}} (\mathbf{A}''; \mathbf{B}'')$, where the latter transitions are described below:

(1) The first part of the transition consists in performing an observable symbolic action α (either $\mathtt{in}(c, X)$ or $\mathtt{out}(c, w)$) followed by all available unobservable (τ) actions. This is done for each extended symbolic process that occurs in the pair of sets, and each possible

[2] This definition yields infinite executions for $\mapsto^{\mathsf{A}}$ if no inequivalent pair is met. Each such execution eventually reaches $(\emptyset; \emptyset)$ while, in practice, executions are obviously not explored past empty pairs. We chose to introduce this minor gap to make the theory more uniform.

transition of one such process generates a new element in the target set. Formally, we have $(\mathbf{A};\mathbf{B}) \overset{\alpha}{\mapsto}^{\mathsf{A1}} (\mathbf{A}';\mathbf{B}')$ if

$$\mathbf{A}' = \bigcup_{(\mathcal{P};\Phi;\mathcal{S};\mathcal{S}^+)\in\mathbf{A}} \left\{ (\mathcal{P}';\Phi';\mathcal{S}';\mathcal{S}^+) \mid (\mathcal{P};\Phi;\mathcal{S}) \overset{\alpha\cdot\tau^*}{\longmapsto} (\mathcal{P}';\Phi';\mathcal{S}') \not\overset{\tau}{\mapsto} \right\},$$

and correspondingly for $\mathbf{B}'$. Note that elements of $(\mathbf{A};\mathbf{B})$ that cannot perform α are simply discarded, and that the constraint systems of individual processes are enriched according to their own transitions whereas the extended part of constraint systems are left unchanged. For a fixed symbolic action α, the $\overset{\alpha}{\mapsto}^{\mathsf{A1}}$ transition is deterministic. The choice of names for handles and second-order variables does not matter, and therefore the relation $\mapsto^{\mathsf{A1}}$ is also finitely branching.

(2) The second part consists in simplifying the constraint systems of $(\mathbf{A}';\mathbf{B}')$ until reaching *solved forms*. This part of the transition is non-deterministic, *i.e.* several different $(\mathbf{A}'';\mathbf{B}'')$ may be reached depending on various choices, *e.g.* whether a message is derived by using a function symbol or one of the available handles. Although branching, this part of the transition is finitely branching. Moreover, only extended constraints may change: for any $(\mathcal{P};\Phi;\mathcal{S};\mathcal{S}_1^+) \in \mathbf{A}''$ there must be a $\mathcal{S}_0^+$ such that $(\mathcal{P};\Phi;\mathcal{S};\mathcal{S}_0^+) \in \mathbf{A}'$, and similarly for $\mathbf{B}''$.

An important invariant of this construction is that all the processes occurring in any of the two sets of processes have constraint systems that share a common structure. More precisely the transitions maintain that for any $(\mathcal{P}_1;\Phi_1;\mathcal{S}_1;\mathcal{S}_1^+), (\mathcal{P}_2;\Phi_2;\mathcal{S}_2;\mathcal{S}_2^+) \in \mathbf{A}\cup\mathbf{B}$, $fv^2(\mathcal{S}_1) = fv^2(\mathcal{S}_2)$ and $D \vdash^?_X x$ occurs in $\mathcal{S}_1$ iff it occurs in $\mathcal{S}_2$.

Example 6.3. *Consider the simple basic processes* $R_i = \mathtt{in}(c_i, x_i).\mathtt{if}\ x_i = \mathsf{ok}\ \mathtt{then}\ \mathtt{out}(c_i, n_i)$ *for* $i \in \mathbb{N}, x_i \in \mathcal{X}, n_i \in \mathcal{N}$, ok *a public constant. We illustrate the roles of* $\mapsto^{\mathsf{A1}}$ *and* $\mapsto^{\mathsf{A2}}$ *on the pair* $(\{Q_0\};\{Q_0\})$ *where* $Q_0 = (\{R_1, R_2\};\emptyset;\emptyset;\emptyset)$. *We have that*

$$(\{Q_0\};\{Q_0\}) \overset{\mathtt{in}(c_2,X_2)}{\longmapsto}^{\mathsf{A1}} (\{Q_0^t, Q_0^e\};\{Q_0^t, Q_0^e\})$$

where Q_0^t *and* Q_0^e *are the two symbolic processes one may obtain by executing the observable action* $\mathtt{in}(c_2, X_2)$*, depending on the conditional after that input. Specifically, we have:*

- $Q_0^t = \big(\{R_1, \mathtt{out}(c_2, n_2)\};\emptyset;\{X_2 \vdash^?_\emptyset x_2, x_2 =^? \mathsf{ok}\};\emptyset\big)$
- $Q_0^e = \big(\{R_1\};\emptyset;\{X_2 \vdash^?_\emptyset x_2, x_2 \neq^? \mathsf{ok}\};\emptyset\big)$

After this first step, $\mapsto^{\mathsf{A2}}$ *is going to non-deterministically solve the constraint systems. From the latter pair, it will produce only two alternatives. Indeed, if* $x_2 =^? \mathsf{ok}$ *holds then* Apte *infers that the only recipe that it needs to consider is the recipe* $R = \mathsf{ok}$*. In that case, the only considered solution is* $\{X_2 \mapsto \mathsf{ok}\}$*. Otherwise,* $x_2 \neq^? \mathsf{ok}$ *holds but, at this point, no more information is inferred on* X_2*. Formally,*

$$(\{Q_0^t, Q_0^e\};\{Q_0^t, Q_0^e\}) \mapsto^{\mathsf{A2}} (\{Q_1^t\};\{Q_1^t\})$$
$$(\{Q_0^t, Q_0^e\};\{Q_0^t, Q_0^e\}) \mapsto^{\mathsf{A2}} (\{Q_1^e\};\{Q_1^e\})$$

where

- $Q_1^t = \big(\{R_1, \mathtt{out}(c_2, n_2)\};\emptyset;\{X_2 \vdash^?_\emptyset x_2, x_2 =^? \mathsf{ok}\};\mathcal{S}_1^t\big)$ *and* $\mathsf{Sol}^+(Q_1^t) = \{\Theta_1^t\}$ *where* $\Theta_1^t = \{X_2 \mapsto \mathsf{ok}\}$*;*
- $Q_1^e = \big(\{R_1\};\emptyset;\{X_2 \vdash^?_\emptyset x_2, x_2 \neq^? \mathsf{ok}\};\mathcal{S}_1^e\big)$.

The content of $\mathcal{S}_1^t$ *and* $\mathcal{S}_1^e$ *is not important. Note that after* $\mapsto^{\mathsf{A2}}$*, only one alternative remains (*i.e. *there is only one extended symbolic process on each side of the resulting pair) because only one of the two processes* Q_0^t, Q_0^e *complies with the choices made in each branch.*

Definition 6.4 ($\approx^{\mathsf{A}}$)**.** *Let $A = (\mathcal{P}_A; \Phi_A)$ and $B = (\mathcal{P}_B; \Phi_B)$ be two processes. We say that $A \approx^{\mathsf{A}} B$ when $\mathbf{A} \sim^{+} \mathbf{B}$ for any pair $(\mathbf{A}; \mathbf{B})$ such that $((\mathcal{P}_A; \Phi_A; \emptyset; \emptyset)\,; (\mathcal{P}_B; \Phi_B; \emptyset; \emptyset)) \overset{\mathsf{tr}}{\mapsto}^{\mathsf{A}} (\mathbf{A}; \mathbf{B})$.*

As announced above, we expect $\approx^{\mathsf{A}}$ to coincide with trace equivalence. We shall actually prove it (see Section 6.3), after having introduced a few axioms (Section 6.2). We note, however, that this can only hold under some minor assumptions on processes. In practice, Apte does not need those assumptions but they allow for a more concise presentation.

Definition 6.5. *A simple process (resp. symbolic process) A is said to be* quiescent *when $A \not\xrightarrow{\tau}$ (resp. $A \not\xmapsto{\tau}$). An extended symbolic process A^+ is* quiescent *when $\lfloor A^+ \rfloor \not\xmapsto{\tau}$.*

In $\overset{\alpha}{\mapsto}^{\mathsf{A1}}$ transitions, processes must start by executing an observable action α and possibly some τ actions after that. Hence, it does not make sense to consider $\mapsto^{\mathsf{A}}$ transitions on processes that can still perform τ actions. We shall thus establish that $\approx^{\mathsf{A}}$ and $\approx^{s}$ coincide only on quiescent processes, which is not a significant restriction since it is always possible to pre-execute all available τ-actions before testing equivalences.

6.2. **Specification of the procedure.** We now list and comment the specification satisfied by the exploration performed by Apte. These statements are consequences of results stated and proved in [18] but it is beyond the scope of this paper to prove them.

Soundness and completeness of constraint resolution. The $\mapsto^{\mathsf{A2}}$ step, corresponding to Apte's constraint resolution procedure, only makes sense under some assumptions on the (common) structure of the processes that are part of the pairs of sets under consideration. Rather than precisely formulating these conditions (which would be at odds with the abstract treatment of extended constraint systems) we start by defining an under-approximation of the set of pairs on which we may apply $\mapsto^{\mathsf{A2}}$ at some point. We choose this under-approximation sufficiently large to cover pairs produced by the compressed semantics, and we then formulate our specifications in that domain. More precisely, the under-approximation has to cover two things:

(1) we have to consider additional disequalities of the form $u \neq^{?} u$ in constraint systems since they are eventually added by our compressed symbolic semantics (see Figure 6);
(2) we have to allow the removal of some extended symbolic process from the original sets since they are eventually discarded by our compressed (resp. reduced) symbolic semantics.

Given an extended symbolic process $A^+ = (\mathcal{P}; \Phi; \mathcal{S}; \mathcal{S}^+)$, we denote $\mathrm{add}(A^+)$ the set of extended symbolic processes obtained from A^+ by adding into $\mathcal{S}$ a number of disequalities of the form $u \neq^{?} u$ with $fv^1(u) \subseteq fv^1(\mathcal{S})$. This is then extended to sets of extended symbolic processes as follows: $\mathrm{add}(\{A_1^+, \ldots, A_n^+\}) = \{\{B_1^+, \ldots, B_n^+\} \mid B_i^+ \in \mathrm{add}(A_i^+)\}$.

Definition 6.6 (valid and intermediate valid pairs)**.** *The set of* valid pairs *is the least set such that:*

- *For all quiescent, symbolic processes $A = (\mathcal{P}; \Phi; \emptyset)$ and $B = (\mathcal{Q}; \Psi; \emptyset)$, $(\{\lceil A \rceil\}; \{\lceil B \rceil\})$ is valid.*
- *If $(\mathbf{A}; \mathbf{B})$ is valid and $\mathbf{A} \sim^{+} \mathbf{B}$, $(\mathbf{A}; \mathbf{B}) \overset{\alpha}{\mapsto}^{\mathsf{A1}} (\mathbf{A}_1; \mathbf{B}_1)$, $\mathbf{A}_2 \subseteq \mathbf{A}_1$, $\mathbf{B}_2 \subseteq \mathbf{B}_1$, $\mathbf{A}_3 \in \mathrm{add}(\mathbf{A}_2)$, $\mathbf{B}_3 \in \mathrm{add}(\mathbf{B}_2)$, and $(\mathbf{A}_3; \mathbf{B}_3) \mapsto^{\mathsf{A2}} (\mathbf{A}'; \mathbf{B}')$ then $(\mathbf{A}'; \mathbf{B}')$ is valid. In that case, the pair $(\mathbf{A}_3; \mathbf{B}_3)$ is called an* intermediate valid pair*.*

It immediately follows that $(\{\lceil A\rceil\};\{\lceil B\rceil\}) \overset{\mathsf{tr}}{\mapsto}^{\mathsf{A}} (\mathbf{A};\mathbf{B})$ implies that $(\mathbf{A};\mathbf{B})$ is valid and only made of quiescent, extended symbolic processes. But the notion of validity accomodates more pairs: it will cover pairs accessible under refinements of $\mapsto^{\mathsf{A}}$ based on subset restrictions of $\mapsto^{\mathsf{A1}}$. We may note that these pairs are actually pairs that would have been explored by Apte when starting with another pair of processes (*e.g.* a process that makes explicit the use of trivial conditionals of the form `if` $u = u$ `then` P `else` Q). Therefore, those pairs do not cause any trouble when they have to be handled by Apte.

Axiom 1 (soundness of constraint resolution)**.** Let $(\mathbf{A}';\mathbf{B}')$ be an intermediate valid pair such that $(\mathbf{A}';\mathbf{B}') \mapsto^{\mathsf{A2}} (\mathbf{A}'';\mathbf{B}'')$. Then, for all $A'' \in \mathbf{A}''$ (resp. $B'' \in \mathbf{B}''$) there exists some $A' \in \mathbf{A}'$ (resp. $B' \in \mathbf{B}'$) such that $\lfloor A'\rfloor = \lfloor A''\rfloor$ (resp. $\lfloor B'\rfloor = \lfloor B''\rfloor$) and $\mathsf{Sol}^+(A'') \subseteq \mathsf{Sol}^+(A')$ (resp. $\mathsf{Sol}^+(B'') \subseteq \mathsf{Sol}^+(B')$).

Apte treats almost symmetrically the two components of the pair of sets on which transitions take place. This is reflected by the fact that axioms concern both sides and are completely symmetric, like **Axiom 1**. In order to make the following specifications more concise and readable, we state properties only for one of the two sets and consider the other "symmetrically" as well.

The completeness specification is in two parts: it first states that no first-order solution is lost in the constraint resolution process, and then that the branching of $\mapsto^{\mathsf{A2}}$ corresponds to different second-order solutions.

Axiom 2 (first-order completeness of constraint resolution)**.** Let $(\mathbf{A};\mathbf{B})$ be an intermediate valid pair. For all $A^+ \in \mathbf{A}$ and $\theta \in \mathsf{Sol}(A^+)$ there exists $(\mathbf{A};\mathbf{B}) \mapsto^{\mathsf{A2}} (\mathbf{A}_2;\mathbf{B}_2)$, $A_2^+ \in \mathbf{A}_2$ and $\theta^+ \in \mathsf{Sol}^+(A_2^+)$ such that $\lfloor A_2^+\rfloor = \lfloor A^+\rfloor$ and $\lambda_\theta^A =_{\mathsf{E}} \lambda_{\theta^+}^A$, where λ_θ^A (resp. $\lambda_{\theta^+}^A$) is the substitution associated to θ (resp. to θ^+) w.r.t. $\lfloor A^+\rfloor$. Symmetrically for $B^+ \in \mathbf{B}$.

Axiom 3 (second-order consistency of constraint resolution)**.** Let $(\mathbf{A};\mathbf{B})$ be an intermediate valid pair such that $(\mathbf{A};\mathbf{B}) \mapsto^{\mathsf{A2}} (\mathbf{A}_2;\mathbf{B}_2)$, $\theta \in \mathsf{Sol}^+(A^+)$ for some $A^+ \in \mathbf{A}$ and $\theta \in \mathsf{Sol}^+(C_2^+)$ for some $C_2^+ \in \mathbf{A}_2 \cup \mathbf{B}_2$. Then there exists some $A_2^+ \in \mathbf{A}_2$ such that $\lfloor A^+\rfloor = \lfloor A_2^+\rfloor$ and $\theta \in \mathsf{Sol}^+(A_2^+)$. Symmetrically for $B^+ \in \mathbf{B}$.

Partial solution. In order to avoid performing some explorations when dependency constraints of our reduced semantics are not satisfied, we shall be interested in knowing when all solutions of a given constraint system assign a given recipe to some variable. Such information is generally available in the solved forms computed by Apte, but not always in a complete fashion. We reflect this by introducing an abstract function that represents the information that can effectively be inferred by the procedure.

Definition 6.7 (partial solution)**.** *We assume a* partial solution[3] *function* ps *which maps sets of extended constraints $\mathcal{S}^+$ to a substitution, such that for any $\theta \in \mathsf{Sol}^+(\mathcal{P};\Phi;\mathcal{S};\mathcal{S}^+)$, there exists θ' such that $\theta = \mathsf{ps}(\mathcal{S}^+) \sqcup \theta'$. We extend* ps *to extended symbolic processes: $\mathsf{ps}(\mathcal{P};\Phi;\mathcal{S};\mathcal{S}^+) = \mathsf{ps}(\mathcal{S}^+)$.*

[3]We use the notation $\sigma_1 \sqcup \sigma_2$ to emphasize the fact that the two substitutions do not interact together. They have disjoint domain, i.e. $\mathrm{dom}(\sigma_1) \cap \mathrm{dom}(\sigma_2) = \emptyset$, and no variable of $\mathrm{dom}(\sigma_i)$ occurs in $\mathrm{img}(\sigma_j)$ with $\{i,j\} = \{1,2\}$.

Intuitively, given an extended constraint system, the function ps returns the value of some of its second-order variables (those for which their instantiation is already completely determined). Our specification of the partial solution shall postulate that the partial solution returned by Apte is the same for each extended symbolic process occurring in a pair $(\mathbf{A};\mathbf{B})$ reached during the exploration. Moreover, there is a monotonicity property that ensures that this partial solution becomes more precise along the exploration.

Axiom 4. We assume the following about the partial solution:

(1) For any valid pair $(\mathbf{A};\mathbf{B})$, we have that $\mathsf{ps}(A) = \mathsf{ps}(B)$ for any $A, B \in \mathbf{A} \cup \mathbf{B}$. This allows us to simply write $\mathsf{ps}(\mathbf{A};\mathbf{B})$ when $\mathbf{A} \cup \mathbf{B} \neq \emptyset$.

(2) For any intermediate valid pair $(\mathbf{A};\mathbf{B})$ such that $(\mathbf{A};\mathbf{B}) \mapsto^{\mathsf{A2}} (\mathbf{A}';\mathbf{B}')$ and $\mathbf{A}' \cup \mathbf{B}' \neq \emptyset$, we have $\mathsf{ps}(\mathbf{A}';\mathbf{B}') = \mathsf{ps}(\mathbf{A};\mathbf{B}) \sqcup \theta$ for some θ.

Example 6.8. *Continuing Example 6.3, we first note that* $(\{Q_0\};\{Q_0\})$ *is a valid pair. Second, the exploration* $(\{Q_0\};\{Q_0\}) \overset{\mathtt{in}(c_2,X_2)}{\mapsto}{}^{\mathsf{A}} (\{Q_1^t\};\{Q_1^t\})$ *covers all executions of the form* $\lfloor Q_0 \rfloor \overset{\mathtt{in}(c_2,X_2).\tau}{\longmapsto} \lfloor Q_1^t \rfloor$ *going to the* `then` *branch even though the only solution of* Q_1^t *is* Θ_1^t*. Indeed, if* $\Theta \in \mathsf{Sol}(\lfloor Q_1^t \rfloor)$ *then the message computed by* $X_2\Theta$ *should be equal to* ok *and thus no first-order solution is lost as stated by* Axiom 2*. Moreover, because the value of* X_2 *is already known in* Q_1^t*, we may have* $\mathsf{ps}(Q_1^t) = \mathsf{ps}(\mathcal{S}_1^+) = \{X_2 \mapsto \mathsf{ok}\}$*.*

6.3. **Proof of the original procedure.** The procedure, axiomatized as above, can be proved correct w.r.t the regular symbolic semantics $\mapsto$ and its induced trace equivalence $\approx^s$ as defined in Section 4.2. Of course, Axiom 4 is unused in this first result. It will be used later on when implementing our reduced semantics. We first start by establishing that all the explorations performed by Apte correspond to symbolic executions.

This result is not new and has been established from scratch (*i.e.* without relying on the axioms stated in the previous section) in [18]. Nevertheless, we found it useful to establish that our axioms are sufficient to prove correctness of the original Apte procedure. The proofs provided in the following sections to establish correctness of our optimised procedure follow the same lines as the ones presented below.

Lemma 6.9. *Let* $(\mathbf{A};\mathbf{B})$ *be a valid pair such that* $(\mathbf{A};\mathbf{B}) \overset{\mathsf{tr}}{\mapsto}{}^{\mathsf{A}} (\mathbf{A}';\mathbf{B}')$*. Then, for all* $A' \in \mathbf{A}'$ *there is some* $A \in \mathbf{A}$ *such that* $\lfloor A \rfloor \overset{\mathsf{tr}'}{\longmapsto} \lfloor A' \rfloor$ *for some* tr' *with* $\mathrm{obs}(\mathsf{tr}') = \mathsf{tr}$*. Symmetrically for* $B' \in \mathbf{B}'$*.*

Proof. We proceed by induction on tr. When tr is empty, we have that $(\mathbf{A};\mathbf{B}) = (\mathbf{A}';\mathbf{B}')$, and the result trivially holds. Otherwise we have that:

$$(\mathbf{A};\mathbf{B}) \overset{\alpha}{\mapsto}{}^{\mathsf{A1}} (\mathbf{A}_1;\mathbf{B}_1) \mapsto^{\mathsf{A2}} (\mathbf{A}_2;\mathbf{B}_2) \overset{\mathsf{tr}_0}{\mapsto}{}^{\mathsf{A}} (\mathbf{A}';\mathbf{B}') \text{ with } \mathsf{tr} = \alpha \cdot \mathsf{tr}_0.$$

Let A' be a process of $\mathbf{A}'$. By induction hypothesis we have some $A_2 \in \mathbf{A}_2$ such that $\lfloor A_2 \rfloor \overset{\mathsf{tr}'_0}{\longmapsto} \lfloor A' \rfloor$ with $\mathrm{obs}(\mathsf{tr}'_0) = \mathsf{tr}_0$. By Axiom 1 there is some $A_1 \in \mathbf{A}_1$ such that $\lfloor A_1 \rfloor = \lfloor A_2 \rfloor$, and by definition of $\mapsto^{\mathsf{A1}}$ we finally find some $A \in \mathbf{A}$ such that $\lfloor A \rfloor \overset{\alpha \cdot \tau^k}{\longmapsto} \lfloor A_1 \rfloor$. To sum up, we have $A \in \mathbf{A}$ such that $\lfloor A \rfloor \overset{\mathsf{tr}}{\mapsto} \lfloor A' \rfloor$ with $\mathrm{obs}(\mathsf{tr}') = \mathsf{tr}$. □

We now turn to completeness results. Assuming that processes under study are in equivalence $\approx^{\mathsf{A}}$ (so that Apte will not stop its exploration prematurely), we are able to show that any valid symbolic execution (*i.e.* a symbolic execution with a solution in its resulting constraint system) is captured by an exploration performed by Apte. Actually, since Apte discards some second-order solution during its exploration, we can only assume that another second-order solution with the same associated first-order solution will be found.

Lemma 6.10. *Let $A = (\mathcal{P}; \Phi; \emptyset)$, $B = (\mathcal{Q}; \Psi; \emptyset)$ and $A' = (\mathcal{P}'; \Phi'; \mathcal{S}')$ be three quiescent, symbolic processes such that $(\mathcal{P}; \Phi) \approx^{\mathsf{A}} (\mathcal{Q}; \Psi)$, $A \overset{\mathsf{tr}}{\mapsto} A'$, and $\theta \in \mathsf{Sol}(A')$. Then there exists an* Apte *exploration $(\{\lceil A \rceil\}; \{\lceil B \rceil\}) \overset{\mathsf{tr}_o}{\longmapsto}^{\mathsf{A}} (\mathbf{A}'; \mathbf{B}')$ and some $A^+ \in \mathbf{A}'$, $\theta^+ \in \mathsf{Sol}^+(A^+)$ such that $\mathrm{obs}(\mathsf{tr}) = \mathsf{tr}_o$, $\lfloor A^+ \rfloor = A'$ and $\lambda_\theta =_{\mathsf{E}} \lambda_{\theta^+}$, where λ_θ (resp. λ_{θ^+}) is the substitution associated to θ (resp. to θ^+) with respect to $(\Phi'; \mathcal{S}')$. Symmetrically for $B \overset{\mathsf{tr}}{\mapsto} B'$.*

Proof. By hypothesis, we have that $A \overset{\mathsf{tr}}{\mapsto} A'$. We will first reorganize this derivation to ensure that τ actions are always performed as soon as possible. Then, we proceed by induction on $\mathrm{obs}(\mathsf{tr})$. When $\mathrm{obs}(\mathsf{tr})$ is empty, we have that $A' = A$ since A is quiescent. Let $(\mathbf{A}'; \mathbf{B}') = (\{\lceil A \rceil\}; \{\lceil B \rceil\})$, $A^+ = \lceil A \rceil$, $\theta^+ = \theta$. We have that $\theta \in \mathsf{Sol}(A)$ and therefore $\theta \in \mathsf{Sol}^+(\lceil A \rceil)$, *i.e.* $\theta^+ = \theta \in \mathsf{Sol}^+(A^+)$. We easily conclude.

Otherwise, consider $A \overset{\mathsf{tr}_0}{\longmapsto} A_1 \overset{\alpha \cdot \tau^k}{\longmapsto} A'$ with $\theta \in \mathsf{Sol}(A')$. Let $A' = (\mathcal{P}'; \Phi'; \mathcal{S}')$ and $A_1 = (\mathcal{P}_1; \Phi_1; \mathcal{S}_1)$. We have that $\mathcal{S}_1 \subseteq \mathcal{S}'$. Since $\theta \in \mathsf{Sol}(A')$, we also have $\theta|_V \in \mathsf{Sol}(A_1)$ where $V = fv^2(\mathcal{S}_1)$. Therefore, we apply our induction hypothesis and we obtain that there exists an Apte exploration $(\{\lceil A \rceil\}; \{\lceil B \rceil\}) \overset{\mathsf{tr}'_0}{\longmapsto}^{\mathsf{A}} (\mathbf{A}_1; \mathbf{B}_1)$ and some $A_1^+ \in \mathbf{A}_1$, $\theta_1^+ \in \mathsf{Sol}^+(A_1^+)$ such that $\mathrm{obs}(\mathsf{tr}_0) = \mathsf{tr}'_0$, $\lfloor A_1^+ \rfloor = A_1$, and the first-order substitutions associated to $\theta|_V$ and θ_1^+ with respect to $(\Phi_1; \mathcal{S}_1)$ are identical. By hypothesis we have $(\mathcal{P}; \Phi) \approx^{\mathsf{A}} (\mathcal{Q}; \Psi)$, thus $\mathbf{A}_1 \sim^+ \mathbf{B}_1$. Hence a $\mapsto^{\mathsf{A1}}$ transition can take place on that pair. By definition of $\mapsto^{\mathsf{A1}}$ and since $\lfloor A_1^+ \rfloor = A_1 \overset{\alpha \cdot \tau^k}{\longmapsto} A'$ with A' quiescent, there must be some $(\mathbf{A}_1; \mathbf{B}_1) \overset{\alpha}{\mapsto}^{\mathsf{A1}} (\mathbf{A}_2; \mathbf{B}_2)$ with $A_2^+ \in \mathbf{A}_2$, $\lfloor A_2^+ \rfloor = A'$. Thus $\theta \in \mathsf{Sol}(A_2^+)$ and we can apply Axiom 2. There exists $(\mathbf{A}_2; \mathbf{B}_2) \mapsto^{\mathsf{A2}} (\mathbf{A}'; \mathbf{B}')$, $A^+ \in \mathbf{A}'$, $\lfloor A^+ \rfloor = \lfloor A_2^+ \rfloor$ and $\theta^+ \in \mathsf{Sol}^+(A^+)$ such that $\lfloor A_2^+ \rfloor = \lfloor A^+ \rfloor$, and the substitutions associated to θ (resp. θ^+) w.r.t. $(\Phi'; \mathcal{S}')$ coincide. To sum up, the exploration

$$(\{\lceil A \rceil\}; \{\lceil B \rceil\}) \overset{\mathsf{tr}'_0}{\longmapsto}^{\mathsf{A}} (\mathbf{A}_1; \mathbf{B}_1) \overset{\alpha}{\mapsto}^{\mathsf{A1}} (\mathbf{A}_2; \mathbf{B}_2) \mapsto^{\mathsf{A2}} (\mathbf{A}'; \mathbf{B}')$$

together with $A^+ \in \mathbf{A}'$, and $\theta^+ \in \mathsf{Sol}^+(A^+)$ satisfy all the hypotheses. □

Lemma 6.11. *Let A, B, A' be quiescent symbolic processes such that $A \overset{\mathsf{tr}}{\mapsto} A' = (\mathcal{P}'; \Phi'; \mathcal{S}')$, $\theta \in \mathsf{Sol}(A')$ and $(\{\lceil A \rceil\}; \{\lceil B \rceil\}) \overset{\mathsf{tr}_o}{\longmapsto}^{\mathsf{A}} (\mathbf{A}'; \mathbf{B}')$ with $\mathrm{obs}(\mathsf{tr}) = \mathsf{tr}_o$ and $\theta \in \mathsf{Sol}^+(C)$ for some $C \in \mathbf{A}' \cup \mathbf{B}'$. Then there exists some $A^+ \in \mathbf{A}'$ such that $\lfloor A^+ \rfloor = A'$ and $\theta \in \mathsf{Sol}^+(A^+)$. Symmetrically for $B \overset{\mathsf{tr}}{\mapsto} B'$.*

Proof. We proceed by induction on tr_o. When tr_o is empty, we have that $A' = A$ (because A is quiescent), $\mathbf{A}' = \{\lceil A \rceil\}$, and $\mathbf{B}' = \{\lceil B \rceil\}$. Let A^+ be $\lceil A \rceil = \lceil A' \rceil$. We deduce that $\theta \in \mathsf{Sol}^+(A^+)$ from the fact that $\theta \in \mathsf{Sol}(A)$ and $A^+ = \lceil A \rceil$.

We consider now the case of a non-empty execution:

$$(\{\lceil A \rceil\}; \{\lceil B \rceil\}) \overset{\mathsf{tr}_o}{\longmapsto}^{\mathsf{A}} (\mathbf{A}_1; \mathbf{B}_1) \overset{\alpha}{\mapsto}^{\mathsf{A1}} (\mathbf{A}_2; \mathbf{B}_2) \mapsto^{\mathsf{A2}} (\mathbf{A}_3; \mathbf{B}_3) \text{ and } A \overset{\mathsf{tr}}{\mapsto} A_1 \overset{\alpha \cdot \tau^k}{\longmapsto} A_3.$$

Note that, by reordering τ actions, we can assume A_1 to be quiescent. By assumption we have $\theta \in \mathsf{Sol}(A_3)$, $\mathrm{obs}(\mathsf{tr}) = \mathsf{tr}_o$ and $\theta \in \mathsf{Sol}^+(C_3)$ for some $C_3 \in \mathbf{A}_3 \cup \mathbf{B}_3$. By Axiom 1, there exists some $C_2 \in \mathbf{A}_2 \cup \mathbf{B}_2$ such that $\theta \in \mathsf{Sol}^+(C_2)$. By definition of $\mapsto^{\mathsf{A1}}$ we obtain $C_1 \in \mathbf{A}_1 \cup \mathbf{B}_1$ such that $\lfloor C_1 \rfloor \overset{\alpha \cdot \tau^k}{\longmapsto} \lfloor C_2 \rfloor$ and $\mathcal{S}^+(C_1) = \mathcal{S}^+(C_2)$ (*i.e.* the sets of extended

constraints of C_1 and C_2 coincide). The first fact implies $\theta|_V \in \mathsf{Sol}(C_1)$ by monotonicity (where $V = fv^2(\mathcal{S}(C_1))$, *i.e.* second-order variables that occur in the set of non-extended constraints of C_1), and the second allows us to conclude more strongly that $\theta|_V \in \mathsf{Sol}^+(C_1)$. Since we also have $\theta|_V \in \mathsf{Sol}(A_1)$ by monotonicity, the induction hypothesis applies and we obtain some $A_1^+ \in \mathbf{A}_1$ with $\lfloor A_1^+ \rfloor = A_1$ and $\theta|_V \in \mathsf{Sol}^+(A_1^+)$.

By definition of $\mapsto^{\mathsf{A1}}$, and since $\lfloor A_1^+ \rfloor \xmapsto{\alpha \cdot \tau^k} A_3 \not\xrightarrow{\mathcal{T}}$ (A_3 is quiescent by hypothesis), we have $A_2^+ \in \mathbf{A}_2$ such that $\lfloor A_2^+ \rfloor = A_3$ and $\mathcal{S}^+(A_1^+) = \mathcal{S}^+(A_2^+)$. Therefore, we have that $\theta \in \mathsf{Sol}(\lfloor A_2^+ \rfloor)$, and the fact that $\mathcal{S}^+(A_1^+) = \mathcal{S}^+(A_2^+)$ allows us to say that $\theta \in \mathsf{Sol}^+(A_2^+)$ We can finally apply Axiom 3 to obtain some A_3^+ such that $\lfloor A_3^+ \rfloor = \lfloor A_2^+ \rfloor = A_3$ and $\theta \in \mathsf{Sol}^+(A_3^+)$. □

Theorem 6.12. *For any quiescent extended simple processes, we have that:*

$$A \approx^s B \text{ if, and only if, } A \approx^{\mathsf{A}} B.$$

Proof. Let $A_0 = (\mathcal{P}; \Phi)$, $B_0 = (\mathcal{P}'; \Phi')$, $\mathbf{A}_0 = \{(\mathcal{P}; \Phi; \emptyset; \emptyset)\}$ and $\mathbf{B}_0 = \{(\mathcal{P}'; \Phi'; \emptyset; \emptyset)\}$. We prove the two directions separately.

($\Rightarrow$) Assume $A_0 \approx^s B_0$ and consider some exploration $(\mathbf{A}_0; \mathbf{B}_0) \xmapsto{\mathsf{tr}_o}{}^{\mathsf{A}} (\mathbf{A}; \mathbf{B})$. We shall establish that $\mathbf{A} \prec^+ \mathbf{B}$. Let $A = (\mathcal{P}_A; \Phi_A; \mathcal{S}_A)$ be in $\mathbf{A}$ and $\theta \in \mathsf{Sol}^+(A)$. By Lemma 6.9, we have $(\mathcal{P}; \Phi; \emptyset) \xmapsto{\mathsf{tr}} \lfloor A \rfloor$ such that $\mathsf{obs}(\mathsf{tr}) = \mathsf{tr}_o$. By hypothesis, there exists $B = (\mathcal{P}_B; \Phi_B; \mathcal{S}_B)$ such that $(\mathcal{P}'; \Phi'; \emptyset) \xmapsto{\mathsf{tr}'} B$, $\mathsf{obs}(\mathsf{tr}') = \mathsf{obs}(\mathsf{tr}) = \mathsf{tr}_o$, $\theta \in \mathsf{Sol}(B)$ and $\Phi_B \lambda_\theta^B \sim \Phi_A \lambda_\theta^A$. We can finally apply Lemma 6.11, which tells us that there must be some $B^+ \in \mathbf{B}$ such that $\lfloor B^+ \rfloor = B$ and $\theta \in \mathsf{Sol}^+(B^+)$.

($\Leftarrow$) We now establish $A_0 \sqsubseteq^s B_0$ assuming $A_0 \approx^{\mathsf{A}} B_0$. Consider $(\mathcal{P}; \Phi; \emptyset) \xmapsto{\mathsf{tr}} A$ and $\theta \in \mathsf{Sol}(A)$. If A is not quiescent, it is easy to complete the latter execution into $(\mathcal{P}; \Phi; \emptyset) \xmapsto{\mathsf{tr} \cdot \tau^k} A' = (\mathcal{P}_A; \Phi_A; \mathcal{S}_A)$ and $\theta \in \mathsf{Sol}(A')$ such that A' is quiescent. By Lemma 6.10 we know that $(\mathbf{A}_0; \mathbf{B}_0) \xmapsto{\mathsf{tr}_o}{}^{\mathsf{A}} (\mathbf{A}; \mathbf{B})$ with $\mathsf{obs}(\mathsf{tr}) = \mathsf{tr}_o$, $A^+ \in \mathbf{A}$, $\theta^+ \in \mathsf{Sol}^+(A^+)$ with $A' = \lfloor A^+ \rfloor$ and $\lambda_\theta =_{\mathsf{E}} \lambda_{\theta^+}$ where λ_θ (resp. λ_{θ^+}) is the substitution associated to θ (resp. θ^+) w.r.t. $(\Phi_A; \mathcal{S}_A)$. By assumption we have $\mathbf{A} \prec^+ \mathbf{B}$ and thus there exists some $B = (\mathcal{P}_B; \Phi_B; \mathcal{S}_B; \mathcal{S}_B^+) \in \mathbf{B}$ with $\theta^+ \in \mathsf{Sol}^+(B)$, and $\Phi_B \lambda_{\theta^+}^B \sim \Phi_A \lambda_{\theta^+}$ where $\lambda_{\theta^+}^B$ is the substitution associated to θ^+ w.r.t. $(\Phi_B; \mathcal{S}_B)$. By Lemma 6.9 we have $(\mathcal{P}'; \Phi'; \emptyset) \xmapsto{\mathsf{tr}'} \lfloor B \rfloor$ with $\mathsf{obs}(\mathsf{tr}') = \mathsf{tr}_o = \mathsf{obs}(\mathsf{tr})$. To conclude the proof, it remains to show that $\theta \in \mathsf{Sol}(\lfloor B \rfloor)$ and that $\Phi_A \lambda_\theta \sim \Phi_B \lambda_\theta^B$ where λ_θ^B is the substitution associated to θ w.r.t. $(\Phi_B; \mathcal{S}_B)$.

For any $X \in fv^2(\mathcal{S}_B) = fv^2(\mathcal{S}_A)$, we have $valid((X\theta)(\Phi_A \lambda_{\theta^+}))$, $valid((X\theta^+)(\Phi_A \lambda_{\theta^+}))$, and

$$(X\theta)(\Phi_A \lambda_{\theta^+}) =_{\mathsf{E}} (X\theta)(\Phi_A \lambda_\theta) =_{\mathsf{E}} x_A \lambda_\theta =_{\mathsf{E}} x_A \lambda_{\theta^+} =_{\mathsf{E}} (X\theta^+)(\Phi_A \lambda_{\theta^+})$$

where x_A is the first-order variable associated to X in $\mathcal{S}_A$. Since $\Phi_A \lambda_{\theta^+} \sim \Phi_B \lambda_{\theta^+}^B$, we deduce that $(X\theta)(\Phi_B \lambda_{\theta^+}^B) =_{\mathsf{E}} (X\theta^+)(\Phi_B \lambda_{\theta^+}^B)$, $valid((X\theta)(\Phi_B \lambda_{\theta^+}^B))$ and therefore $\theta \in \mathsf{Sol}(\lfloor B \rfloor)$, and its associated substitution λ_θ^B w.r.t. $(\Phi_B; \mathcal{S}_B)$ coincides with $\lambda_{\theta^+}^B$, and therefore $\Phi_A \lambda_\theta \sim \Phi_B \lambda_\theta^B$ is a direct consequence of $\Phi_B \lambda_{\theta^+}^B \sim \Phi_A \lambda_{\theta^+}$ and $\lambda_\theta =_{\mathsf{E}} \lambda_{\theta^+}$. □

6.4. **Integrating compression.** We now discuss the integration of the compressed semantics of Section 4 as a replacement for the regular symbolic semantics in Apte.

Although our compressed semantics $\mapsto_c$ has been defined as executing blocks rather than elementary actions, we allow ourselves to view it in a slightly different way in this section: we shall assume that the symbolic compressed semantics deals with elementary actions and enforces that those actions, when put together, form a prefix of a sequence of

blocks that can actually be executed (for the process under consideration) in the compressed semantics of Section 4. This can easily be obtained by means of extra annotations at the level of processes, and we will not detail that modification. This slight change makes it simpler to integrate compression into Apte, both in the theory presented here and in the implementation.

Definition 6.13. *Given two sets of extended symbolic processes* $\mathbf{A}, \mathbf{B}$*, and an observable action* α*, we write* $(\mathbf{A};\mathbf{B}) \overset{\alpha}{\mapsto}{}^{\mathsf{A1}}_{c} (\mathbf{A}';\mathbf{B}')$ *when*

$$\mathbf{A}' = \bigcup_{(\mathcal{P};\Phi;\mathcal{S};\mathcal{S}^+)\in\mathbf{A}} \left\{ (\mathcal{P}';\Phi';\mathcal{S}';\mathcal{S}^+) \mid (\mathcal{P};\Phi;\mathcal{S}) \overset{\alpha}{\mapsto}_c (\mathcal{P}';\Phi';\mathcal{S}') \not\Rightarrow \right\},$$

and similarly for $\mathbf{B}'$*. We say that* $(\mathbf{A};\mathbf{B}) \overset{\alpha}{\mapsto}{}^{\mathsf{A}}_{c} (\mathbf{A}'';\mathbf{B}'')$ *when* $(\mathbf{A};\mathbf{B}) \overset{\alpha}{\mapsto}{}^{\mathsf{A1}}_{c} (\mathbf{A}';\mathbf{B}')$ *and* $(\mathbf{A}';\mathbf{B}') \mapsto^{\mathsf{A2}} (\mathbf{A}'';\mathbf{B}'')$*.*

Finally, given two simple extended processes $A = (\mathcal{P}_A;\Phi_A)$ *and* $B = (\mathcal{P}_B;\Phi_B)$*, we say that* $A \approx^{\mathsf{A}}_{c} B$ *when* $\mathbf{A} \approx^{+} \mathbf{B}$ *for any* $(\{\lceil(\mathcal{P}_A;\Phi_A;\emptyset)\rceil\};\{\lceil(\mathcal{P}_B;\Phi_B;\emptyset)\rceil\}) \overset{\mathsf{tr}}{\mapsto}{}^{\mathsf{A}}_{c} (\mathbf{A};\mathbf{B})$*.*

As expected, $\mapsto^{\mathsf{A1}}_{c}$ allows to consider much fewer explorations than with the original $\mapsto^{\mathsf{A1}}$. It inherits the features of compression, prioritizing outputs, not considering interleavings of outputs, executing inputs only under focus, and preventing executions beyond improper blocks. These constraints apply to individual processes in $\mathbf{A} \cup \mathbf{B}$, but we remark that they also have a global effect in $\mapsto^{\mathsf{A1}}_{c}$, *e.g.* all processes of $\mathbf{A} \cup \mathbf{B}$ must start a new block simultaneously: recall that the beginning of a block corresponds to an input after some outputs, and such inputs can only be executed if no more outputs are available.

Example 6.14. *Continuing Example 6.8, there is only one non-trivial*[4] *compressed exploration of one action from the valid pair* $(\{Q_1^t\};\{Q_1^t\})$*. It corresponds to the output on channel* c_2*:* $(\{Q_1^t\};\{Q_1^t\}) \overset{\mathtt{out}(c_2,w_2)}{\longmapsto}{}^{\mathsf{A}}_{c} (\{Q_2\},\{Q_2\})$ *for* $Q_2 = \big(\{R_1\};\{w_2 \triangleright n_2\};\{X_2 \vdash^{?}_{\emptyset} x_2, x_2 =^{?} \mathsf{ok}\};\mathcal{S}_2^{+}\big)$*. In particular, for any* $i \in \{1,2\}$*, we have* $(\{Q_1^t\};\{Q_1^t\}) \overset{\mathtt{in}(c_i,X_i)}{\longmapsto}{}^{\mathsf{A}}_{c} (\emptyset;\emptyset)$*.*

Observe that, because $\mapsto^{\mathsf{A}}_{c}$ is obtained from $\mapsto^{\mathsf{A}}$ by a subset restriction in $\mapsto^{\mathsf{A1}}$ up to some disequalities, we have that $(\mathbf{A}';\mathbf{B}')$ is a valid pair when $(\{\lceil A\rceil\};\{\lceil B\rceil\}) \overset{\mathsf{tr}}{\mapsto}{}^{\mathsf{A}}_{c} (\mathbf{A}';\mathbf{B}')$ for some quiescent, symbolic processes A, B having empty sets of constraints. Following the same reasoning as the one performed in Section 6.3, we can establish that $\approx^{s}_{c}$ coincides with $\approx^{\mathsf{A}}_{c}$. The main difference is that $\mapsto_c$ already ignores τ-actions, and therefore we do not need to apply the obs$(\cdot)$ operator.

Lemma 6.15. *Let* $(\mathbf{A};\mathbf{B})$ *be a valid pair such that* $(\mathbf{A};\mathbf{B}) \overset{\mathsf{tr}}{\mapsto}{}^{\mathsf{A}}_{c} (\mathbf{A}';\mathbf{B}')$*. Then, for all* $A' \in \mathbf{A}'$ *there is some* $A \in \mathbf{A}$ *such that* $\lfloor A\rfloor \overset{\mathsf{tr}}{\mapsto}_c \lfloor A'\rfloor$*. Symmetrically for* $B' \in \mathbf{B}'$*.*

Lemma 6.16. *Let* $A = (\mathcal{P};\Phi;\emptyset)$*,* $B = (\mathcal{Q};\Psi;\emptyset)$*, and* $A' = (\mathcal{P}';\Phi';\mathcal{S}')$ *be three quiescent, symbolic processes such that* $(\mathcal{P};\Phi) \approx^{\mathsf{A}}_{c} (\mathcal{Q};\Psi)$*,* $A \overset{\mathsf{tr}}{\mapsto}_c A'$ *and* $\theta \in \mathsf{Sol}(A')$*. Then there exists an exploration* $(\{\lceil A\rceil\};\{\lceil B\rceil\}) \overset{\mathsf{tr}}{\mapsto}{}^{\mathsf{A}}_{c} (\mathbf{A}';\mathbf{B}')$ *and some* $A^+ \in \mathbf{A}'$*,* $\theta^+ \in \mathsf{Sol}^+(A^+)$ *such that* $\lfloor A^+\rfloor = A'$ *and* $\lambda_\theta =_{\mathsf{E}} \lambda_{\theta^+}$*, where* λ_θ *(resp.* λ_{θ^+}*) is the substitution associated to* θ *(resp. to* θ^+*) with respect to* $(\Phi';\mathcal{S}')$*. Symmetrically for* $B \overset{\mathsf{tr}}{\mapsto}_c B'$*.*

Lemma 6.17. *Let* A, B *and* A' *be quiescent, simple symbolic processes such that* $A \overset{\mathsf{tr}}{\mapsto}_c A' = (\mathcal{P}';\Phi';\mathcal{S}')$*,* $\theta \in \mathsf{Sol}(A')$*, and* $(\{\lceil A\rceil\};\{\lceil B\rceil\}) \overset{\mathsf{tr}}{\mapsto}{}^{\mathsf{A}}_{c} (\mathbf{A}';\mathbf{B}')$ *with* $\theta \in \mathsf{Sol}^+(C)$ *for some* $C \in \mathbf{A}' \cup \mathbf{B}'$*. Then there exists some* $A^+ \in \mathbf{A}'$ *such that* $\lfloor A^+\rfloor = A'$ *and* $\theta \in \mathsf{Sol}^+(A^+)$*. Symmetrically for* $B \overset{\mathsf{tr}}{\mapsto}_c B'$*.*

[4] We dismiss here the (infinitely many) transitions obtained for infeasible actions, which yield $(\emptyset;\emptyset)$.

Theorem 6.18. *For any quiescent extended simple processes, we have that:*

$$A \approx_c^s B \text{ if, and only if, } A \approx_c^{\mathsf{A}} B.$$

6.5. **Integrating dependency constraints in** Apte**.** We now define a final variant of Apte explorations, which integrates the ideas of Section 5 to further reduce redundant explorations. We can obviously generate dependency constraints in Apte, just like we did in Section 5, but the real difficulty is to exploit them in constraint resolution to prune some branches of the exploration performed by Apte. Roughly, we shall simply stop the exploration when reaching a state for which we know that all of its solutions violate dependency constraints. To do that, we rely on the notion of partial solution introduced in Section 6.1. In other words, we do not modify Apte's constraint resolution, but simply rely on information that it already provides to know when dependency constraints become unsatisfiable. As we shall see, this simple strategy is very satisfying in practice.

Definition 6.19. *We define* $\mapsto_r^{\mathsf{A}}$ *as the greatest relation contained in* $\mapsto_c^{\mathsf{A}}$ *and such that, for any symbolic processes* A *and* B *with empty constraint sets,* $(\{\lceil A\rceil\};\{\lceil B\rceil\}) \overset{\mathsf{tr}}{\mapsto}_r^{\mathsf{A}} (\mathbf{A}';\mathbf{B}')$ *implies that there is no* $\overrightarrow{X} \ltimes \overrightarrow{w} \in \mathrm{Deps}(\mathsf{tr})$ *such that for all* $X_i \in \overrightarrow{X}$ *we have* $X_i \in \mathrm{dom}(\mathsf{ps}(\mathbf{A}';\mathbf{B}'))$*, and* $\overrightarrow{w} \cap fv^1(X_i\mathsf{ps}(\mathbf{A}';\mathbf{B}')) = \emptyset$*.*

Finally, given two simple extended process $A = (\mathcal{P}_A;\Phi_A)$ *and* $B = (\mathcal{P}_B;\Phi_B)$*, we say that* $A \approx_r^{\mathsf{A}} B$ *when* $\mathbf{A} \sim^+ \mathbf{B}$ *for any pair* $(\mathbf{A};\mathbf{B})$ *such that* $((\mathcal{P}_A;\Phi_A;\emptyset;\emptyset);(\mathcal{P}_B;\Phi_B;\emptyset;\emptyset)) \overset{\mathsf{tr}}{\mapsto}_r^{\mathsf{A}} (\mathbf{A};\mathbf{B})$*.*

Example 6.20. *Continuing Example 6.14, consider the following compressed exploration, where* Q_3 *contains the constraints* $X_2 \vdash^?_{\emptyset} x_2$*,* $X_1 \vdash^?_{\{w_2 \triangleright n_2\}} x_1$*,* $x_2 =^? \mathsf{ok}$ *and* $x_1 =^? \mathsf{ok}$*:*

$$(\{Q_0\};\{Q_0\}) \overset{\mathtt{in}(c_2,X_2)}{\mapsto}{}_c^{\mathsf{A}} \overset{\mathtt{out}(c_2,w_2)}{\mapsto}{}_c^{\mathsf{A}} (\{Q_2\};\{Q_2\}) \overset{\mathtt{in}(c_1,X_1)}{\mapsto}{}_c^{\mathsf{A}} \dots \overset{\mathtt{out}(c_1,w_1)}{\mapsto}{}_c^{\mathsf{A}} (\{Q_3\};\{Q_3\}).$$

Assuming that $\mathsf{ps}(Q_3) = \{X_2 \mapsto \mathsf{ok}, X_1 \mapsto \mathsf{ok}\}$ *(which is the case in the actual* Apte *procedure) this compressed exploration is not explored by* $\mapsto_r^{\mathsf{A}}$ *because*

$$X_1 \ltimes w_2 \in \mathrm{Deps}\left(\mathtt{io}_{c_2}(X_2,w_2)\cdot \mathtt{io}_{c_1}(X_1,w_1)\right),\ X_1\mathsf{ps}(Q_3) = \mathsf{ok} \ \text{ and } \ \{w_2\} \cap fv^1(\mathsf{ok}) = \emptyset.$$

Lemma 6.21. *Let* $A = (\mathcal{P};\Phi;\emptyset)$*,* $B = (\mathcal{Q};\Psi;\emptyset)$ *and* $A' = (\mathcal{P}';\Phi';\mathcal{S}')$ *be quiescent, simple symbolic processes such that* $(\mathcal{P};\Phi) \approx_r^{\mathsf{A}} (\mathcal{Q};\Psi)$*,* $A \overset{\mathsf{tr}}{\mapsto}_c A'$*,* $\theta \in \mathsf{Sol}(A')$ *and* $\theta \models_{(\Phi';\mathcal{S}')} \mathrm{Deps}(\mathsf{tr})$*. Then there exists an exploration* $(\{\lceil A\rceil\};\{\lceil B\rceil\}) \overset{\mathsf{tr}}{\mapsto}_r^{\mathsf{A}} (\mathbf{A}';\mathbf{B}')$ *and some* $A^+ \in \mathbf{A}'$*,* $\theta^+ \in \mathsf{Sol}^+(A^+)$ *such that* $\lfloor A^+\rfloor = A'$ *and* $\lambda_\theta =_{\mathsf{E}} \lambda_{\theta^+}$*, where* λ_θ *(resp.* λ_{θ^+}*) is the substitution associated to* θ *(resp. to* θ^+*) with respect to* $(\Phi';\mathcal{S}')$*. Symmetrically for* $B \overset{\mathsf{tr}}{\mapsto}_c B'$*.*

Proof. We proceed by induction on tr. The empty case is easy. Otherwise, consider $A \overset{\mathsf{tr}}{\mapsto}_c A_1 \overset{\alpha}{\mapsto}_c A_3 = (\mathcal{P}_3;\Phi_3;\mathcal{S}_3)$ with $\theta \in \mathsf{Sol}(A_3)$, A_1, A_3 quiescent, and $\theta \models_{(\Phi_3;\mathcal{S}_3)} \mathrm{Deps}(\mathsf{tr}\cdot\alpha)$. Let $A_1 = (\mathcal{P}_1;\Phi_1;\mathcal{S}_1)$ and $V_1 = fv^2(\mathcal{S}_1)$. We also have $\theta|_{V_1} \in \mathsf{Sol}(A_1)$ and $\theta|_{V_1} \models_{(\Phi_1;\mathcal{S}_1)} \mathrm{Deps}(\mathsf{tr})$, so the induction hypothesis applies and we obtain $(\{\lceil A\rceil\};\{\lceil B\rceil\}) \overset{\mathsf{tr}}{\mapsto}_r^{\mathsf{A}} (\mathbf{A}_1;\mathbf{B}_1)$ with $A_1^+ \in \mathbf{A}_1$, $\lfloor A_1^+\rfloor = A_1$ and $\theta_1^+ \in \mathsf{Sol}^+(A_1^+)$ such that the first-order substitutions associated to $\theta|_{V_1}$ and θ_1^+ w.r.t. $(\Phi_1;\mathcal{S}_1)$ coincide.

By hypothesis we have $A \approx_r^{\mathsf{A}} B$, thus $\mathbf{A}_1 \sim^+ \mathbf{B}_1$. Hence a $\mapsto_c^{\mathsf{A1}}$ transition can take place on that pair. By definition of $\mapsto_c^{\mathsf{A1}}$ and since $\lfloor A_1^+\rfloor = A_1 \overset{\alpha}{\mapsto}_c A_3$, there must be some $(\mathbf{A}_1;\mathbf{B}_1) \overset{\alpha}{\mapsto}_c^{\mathsf{A1}} (\mathbf{A}_2;\mathbf{B}_2)$ with $A_2^+ \in \mathbf{A}_2$, $\lfloor A_2^+\rfloor = A_3$. Thus $\theta \in \mathsf{Sol}(A_2^+)$ and we can apply Axiom 2 to obtain $(\mathbf{A}_2;\mathbf{B}_2) \mapsto^{\mathsf{A2}} (\mathbf{A}_3;\mathbf{B}_3)$ with $A_3^+ \in \mathbf{A}_3$, $\lfloor A_3^+\rfloor = \lfloor A_2^+\rfloor$ and $\theta_3^+ \in \mathsf{Sol}^+(A_3^+)$ such that the subsitutions associated to θ and θ_3^+ w.r.t. $(\Phi_3;\mathcal{S}_3)$ coincide.

It only remains to show that this extra execution step in $\mapsto_c^{\mathsf{A}}$ is also present in $\mapsto_r^{\mathsf{A}}$, *i.e.* that $\mathsf{ps}(\mathbf{A}_3;\mathbf{B}_3)$ does not violate $\mathrm{Deps}(\mathsf{tr}\cdot\alpha)$ in the sense of Definition 6.19. This is because, by definition of the partial solution, we have that $\theta_3^+ = \mathsf{ps}(\mathbf{A}_3;\mathbf{B}_3)\sqcup\tau$ for some τ, so that if $\mathsf{ps}(\mathbf{A}_3;\mathbf{B}_3)$ violated $\mathrm{Deps}(\mathsf{tr}\cdot\alpha)$ then we would have $\theta_3^+ \not\models_{(\Phi_3;\mathcal{S}_3)} \mathrm{Deps}(\mathsf{tr}\cdot\alpha)$. Since θ_3^+ and θ induce the same first-order substitutions with respect to $(\Phi_3;\mathcal{S}_3)$, we would finally have $\theta \not\models_{(\Phi_3;\mathcal{S}_3)} \mathrm{Deps}(\mathsf{tr}\cdot\alpha)$, contradicting the hypothesis on θ. □

Theorem 6.22. *For any quiescent initial simple processes A and B, we have that:*

$$A \approx B \text{ if, and only if, } A \approx_r^{\mathsf{A}} B.$$

Proof. Let $A = (\mathcal{P};\Phi)$ and $B = (\mathcal{Q};\Psi)$ be two quiescent, initial simple processes. Thanks to our previous results, we have that $A \approx B$ implies $A \approx_c^{\mathsf{A}} B$. Then, we obviously have $A \approx_r^{\mathsf{A}} B$: for any $(\{\lceil(\mathcal{P};\Phi;\emptyset)\rceil\};\{\lceil(\mathcal{Q};\Psi;\emptyset)\rceil\}) \overset{\mathsf{tr}}{\mapsto}_r^{\mathsf{A}} (\mathbf{A}';\mathbf{B}')$ we have $(\{\lceil(\mathcal{P};\Phi;\emptyset)\rceil\};\{\lceil(\mathcal{Q};\Psi;\emptyset)\rceil\}) \overset{\mathsf{tr}}{\mapsto}_c^{\mathsf{A}} (\mathbf{A}';\mathbf{B}')$ by definition of $\mapsto_r^{\mathsf{A}}$, and thus $\mathbf{A}' \sim^+ \mathbf{B}'$ by hypothesis.

For the other direction, it suffices to show that $A \approx_r^{\mathsf{A}} B$ implies $A \sqsubseteq_r^s B$. Let $(\mathcal{P};\Phi;\emptyset) \overset{\mathsf{tr}}{\mapsto}_c A' = (\mathcal{P}';\Phi';\mathcal{S}')$ with $\theta \in \mathsf{Sol}(A')$ and $\theta \models_{(\Phi';\mathcal{S}')} \mathrm{Deps}(\mathsf{tr})$. By Lemma 6.21 we have $(\{\lceil(\mathcal{P};\Phi;\emptyset)\rceil\};\{\lceil(\mathcal{Q};\Psi;\emptyset)\rceil\}) \overset{\mathsf{tr}}{\mapsto}_r^{\mathsf{A}} (\mathbf{A}';\mathbf{B}')$ with $A^+ \in \mathbf{A}'$, $\theta^+ \in \mathsf{Sol}^+(A^+)$ such that $\lfloor A^+\rfloor = A'$ and $\lambda_\theta^{A'} =_{\mathsf{E}} \lambda_{\theta^+}^{A'}$ where $\lambda_\theta^{A'}$ (resp. $\lambda_{\theta^+}^{A'}$) is the substitution associated to θ (resp. θ^+) w.r.t. $(\Phi';\mathcal{S}')$.

Since $A \approx_r^{\mathsf{A}} B$, we have $\mathbf{A}' \sim^+ \mathbf{B}'$: there must be some $B^+ = (\mathcal{P}_{B'};\Phi_{B'};\mathcal{S}_{B'};\mathcal{S}_B^+) \in \mathbf{B}'$ such that $\theta^+ \in \mathsf{Sol}^+(B^+)$ and $\Phi'\lambda_{\theta^+}^{A'} \sim \Phi_{B'}\lambda_{\theta^+}^{B'}$ where $\lambda_{\theta^+}^{B'}$ is the substitution associated to θ^+ w.r.t. $(\Phi_{B'};\mathcal{S}_{B'})$. By Lemma 6.15 we have $(\mathcal{Q};\Psi;\emptyset) \overset{\mathsf{tr}}{\mapsto}_c \lfloor B^+\rfloor$. Furthermore, we can show as before (see the end of the proof of Theorem 6.12) that $\theta \in \mathsf{Sol}(B^+)$ and $\Phi'\lambda_\theta^{A'} \sim \Phi_{B'}\lambda_\theta^{B'}$, where $\lambda_\theta^{B'}$ is the substitution associated to θ w.r.t. $(\Phi_{B'};\mathcal{S}_{B'})$. Finally, by $\theta \models_{(\Phi';\mathcal{S}')} \mathrm{Deps}(\mathsf{tr})$, $D_{(\Phi';\mathcal{S}')} = D_{(\Phi_{B'};\mathcal{S}_{B'})}$ (*i.e.* sets of handles that second-order variables may use coincide), and $\Phi'\lambda_\theta^{A'} \sim \Phi_{B'}\lambda_\theta^{B'}$, we obtain that $\theta \models_{(\Phi_{B'};\mathcal{S}_{B'})} \mathrm{Deps}(\mathsf{tr})$. □

6.6. **Benchmarks.** The optimisations developed in the present paper have been implemented, following the above approach, in the official version of Apte [23].

In practice, many processes enjoy a nice property that allows one to ensure that non-blocking outputs will never occur: it is often the case that enough tests have been performed before outputting a term to ensure its validity.

Example 6.23. *Consider the following process, where k' is assumed to be valid (*e.g. *because it is a pure constructor term):*

$$\mathtt{in}(c,x).\mathtt{if}\ \mathsf{dec}(x,k) = \mathsf{hash}(u)\ \mathtt{then}\ \mathtt{out}(c,\mathsf{enc}(\mathsf{dec}(x,k),k'))$$

The term outputted during an execution is necessarily valid thanks to the test that is performed just before this output.

We exploit this property in order to avoid adding additional disequalities when integrating compression in Apte. Therefore, in this section, we will restrict ourselves to simple processes that are *non-blocking* as defined below.

Definition 6.24. *Let $(\mathcal{P};\Phi)$ be a simple process. We say that $(\mathcal{P};\Phi)$ is* non-blocking *if u is valid for any tr, c, u, Q', $\mathcal{Q}$, Ψ such that $(\mathcal{P};\Phi) \overset{\mathsf{tr}}{\to} (\{\mathtt{out}(c,u).Q'\}\cup\mathcal{Q};\Psi)$.*

This condition may be hard to check in general, but it is actually quite easy to see that it is satisfied on all of our examples. Roughly, enough tests are performed before any output action, and this ensures the validity of the term when the output action becomes reachable, as in Example 6.23.

Our modified version of Apte can verify our optimised equivalences in addition to the original trace equivalence. It has been integrated into the main development line of the tool. The modifications of the code ($\approx$ 2kloc) are summarized at

`https://github.com/lutcheti/APTE/compare/ref...APTE:POR`

For reference, the version of Apte that we are using in the benchmarks below is available at `https://github.com/APTE/APTE/releases/tag/bench-POR-LMCS` together with all benchmark files, in subdirectory `bench/protocols`. More details, including instructions for reproducing our benchmarks are available at `http://www.lsv.fr/~hirschi/apte_por`.

We ran the tool (compiled with OCaml 3.12.1) on a single 2.67GHz Xeon core (memory is not relevant) and compared three different versions:

- *reference:* the reference version without our optimisations (*i.e.* $\approx^{\mathsf{A}}$);
- *compression:* using only the compression optimisation (*i.e.* $\approx^{\mathsf{A}}_{c}$);
- *reduction:* using both compression and reduction (*i.e.* $\approx^{\mathsf{A}}_{r}$).

We first show examples in which equivalence holds. They are the most significant, because the time spent on inequivalent processes is too sensitive to the order in which the (depth-first) exploration is performed.

Toy example. We consider a parallel composition of n roles R_i as defined in Example 6.3: $P_n := \Pi_{i=1}^{n} R_i$. When executed in the regular symbolic semantics $\mapsto$, the $2n$ actions of P_n may be interleaved in $(2n)!/2^n$ ways in a trace containing all actions. In the compressed symbolic semantics $\mapsto_c$, the actions of individual R_i processes must be bundled in blocks, so there are only $n!$ interleavings containing all actions. In the reduced symbolic semantics $\mapsto_r$, only one interleaving of that length remains: the trace cannot deviate from the priority order, since the only way to satisfy a dependency constraint would be to feed an input with a message that cannot be derived without some previously output nonce n_i, but in that case the message will not be ok and the trace won't be explored further. Note that there is still an exponential number of symbolic traces in the reduced semantics when one takes into account traces with less than $2n$ actions.

We show in Figure 8 the time needed to verify $P_n \approx P_n$ for $n = 1$ to 22 in the three versions of Apte described above: *reference*, *compression* and *reduction*. The results, in logarithmic scale, show that each of our optimisations brings an exponential speedup, as predicted by our theoretical analysis. Similar improvements are observed if one compares the numbers of explored pairs rather than execution times.

Denning-Sacco protocol. We ran a similar benchmark, checking that Denning-Sacco ensures strong secrecy in various scenarios. The protocol has three roles and we added processes playing those roles in turn, starting with three processes in parallel. Srong secrecy is expressed by considering, after one of the roles B, the output of a message encrypted with the established key on one side of the equivalence, and with a fresh key on the other side. The results are plotted in Figure 9. The fact that we add one role out of three at each step explains the irregular growth in verification time. We still observe an exponential speedup for each optimisation.

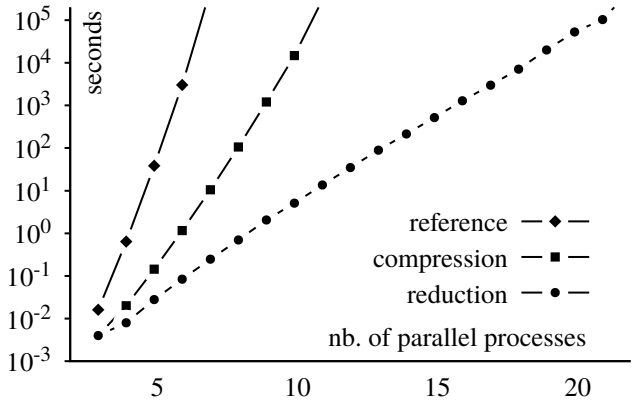


Figure 8: Impact of optimisations on verification time on toy example.

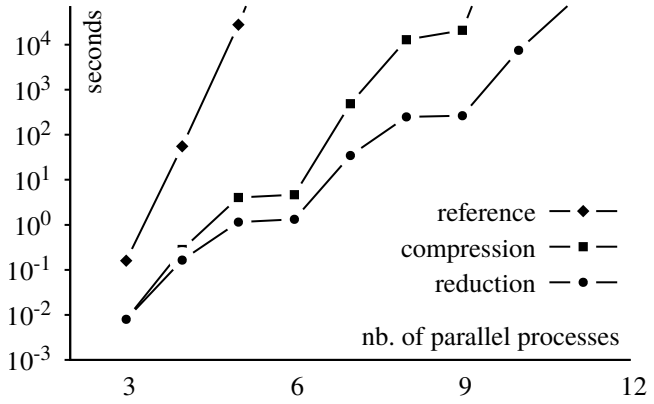


Figure 9: Impact of optimisations on verification time on Denning-Sacco.

Practical impact. Finally, we illustrate how our optimisations make Apte much more useful in practice for investigating interesting scenarios. Verifying a single session of a protocol brings little assurance into its security. In order to detect replay attacks and to allow the attacker to compare messages that are exchanged, at least two sessions should be considered. This means having at least four parallel processes for two-party protocols, and six when a trusted third party is involved. This is actually beyond what the unoptimised Apte can handle in a reasonable amount of time. We show in Figure 10 how many parallel processes could be handled in 20 hours by Apte on various use cases of protocols, for the same three variants of Apte as before, *i.e. reference*, *compression* and *reduction*. We verify an anonymity property for the Passive Authentication protocol of e-passports. For other protocols, we analyse strong secrecy of established keys: for one of the roles we add, on one side of the equivalence, an output encrypted by the established key and, on the other side, an output encrypted by a fresh key.

Protocol	reference	compression	reduction
Needham Schroeder (3-party)	4	6	7
Private Authentication (2-party)	4	7	7
Yahalom (3-party)	4	5	5
E-Passport PA (2-party)	4	7	9
Denning-Sacco (3-party)	5	9	10
Wide Mouth Frog (3-party)	6	12	13

Figure 10: Maximum number of parallel processes verifiable in 20 hours.

We finally present the benefits of our optimisations for discovering attacks. We performed some experiments on flawed variants of protocols, shown in Figure 11, corresponding to example files in subdirectory `bench/protocols/attacks/` of the above mentioned release. The scenario *Denning-Sacco A* expresses strong secrecy of the (3-party) Denning-Sacco protocol, but this time on two instances of roles at the same time (instead of one as in Figure 10). In *Denning-Sacco B*, we consider again a form of strong secrecy expressed by outputting encrypted messages but this time at the end of role B. The *Needham-Schroeder pub* scenario corresponds to strong secrecy of the public-key Needham-Schroder protocol. The *E-Passport PA exposed* experiments show that anonymity is (obviously) lost with the Passive Authentication protocol when the secret key is made public. Similarly, the *Yahalom exposed* experiment shows that strong secrecy of Yahalom is lost when secrets keys are revealed. Since Apte stops its exploration as soon as an attack is found, the time needed for Apte to find the attack highly depends on the order in which the depth-first exploration is performed. However, as shown in Figure 11, we always observe in practice dramatic improvements brought by our optimisations compared to the reference version of Apte. In some cases, our optimisations are even mandatory for Apte to find the attack using reasonable resources.

Protocol	reference	compression	reduction
Denning-Sacco A (6 par. proc.)	OoM	0.07s	0.02s
Denning-Sacco B (6 par. proc.)	5.83s	0.04s	0.04s
Needham-Shroeder pub (7 par. proc.)	TO	0.77s	0.67s
Needham-Shroeder pub (5 par. proc.)	0.79s	0.21s	0.13s
E-Passport PA exposed (8 par. proc.)	TO	0.02s	0.02s
E-Passport PA exposed (6 par. proc.)	4.37s	0.03s	0.02s
Yahalom exposed (4 par. proc.)	7.24s	0.02s	0.02s

Figure 11: Impact of optimisations for finding attacks (OoM denotes a consumption of >32Go of RAM and TO denotes a running time of >20 hours).

7. Related Work

The techniques we have presented borrow from standard ideas from concurrency theory and trace theory. Blending all these ingredients, and adapting them to the demanding framework of security protocols, we have come up with partial order reduction techniques that can effectively be used in symbolic verification algorithms for equivalence properties of security

protocols. We now discuss related work, and there is a lot of it given the huge success of POR techniques in various application areas. We shall focus on the novel aspects of our approach, and explain why such techniques have not been needed outside of security protocol analysis. These observations are not new: as pointed out by Baier and Katoen [11], "[POR] is mainly appropriate to control-intensive applications and less suited for data-intensive applications"; Clarke *et al.* [25] also remark that "In the domain of model checking of reactive systems, there are numerous techniques for reducing the state space of the system. One such technique is partial-order reduction. This technique does not directly apply to [security protocol analysis] because we explicitly keep track of knowledge of various agents, and our logic can refer to this knowledge in a meaningful way." We first compare our work with classical POR techniques, and then comment on previous work in the domain of security protocol analysis.

7.1. **Classical POR.** Partial order reduction techniques have proved very useful in the domain of model checking concurrent programs. Given a Labelled Transition System (LTS) and some property to check (*e.g.* a Linear Temporal Logic formula), the basic idea of POR [41, 34, 11] is to only consider a reduced version of the given LTS whose transitions of some states might be not exhaustive but are such that this transformation does not affect the property. POR techniques can be categorized in two groups [34]. First, the *persistent set* techniques (*e.g. stubborn sets, ample sets*) where only a sufficiently representative subset of available transitions is explored. Second, *sleep set* techniques memoize past exploration and use this information along with available transitions to disable some provably redundant transitions. Note that these two kinds of techniques are compatible, and are indeed often combined to obtain better reductions. Theoretical POR techniques apply to transition systems which may not be explicitly available in practice, or whose explicit computation may be too costly. In such cases, POR is often applied to an approximation of the LTS that is obtained through static analysis. Another, more recent approach is to use *dynamic POR* [31, 45, 3] where the POR arguments are applied based on information that is obtained during the execution of the system.

Clearly, classical POR techniques would apply to our concrete LTS, but that would not be practically useful since this LTS is wildly infinite, taking into account all recipes that the attacker could build. Applying most classical POR techniques to the LTS from which data would have been abstracted away would be ineffective: any input would be dependent on any output (since the attacker's knowledge, increased by the output, may enable new input messages). Our compression technique lies between these two extremes. It exploits a semi-commutation property: outputs can be permuted before inputs, but not the converse in general. Further, it exploits the fact that inputs do not increase the attacker's knowledge, and can thus be executed in a chained fashion, under focus. The semi-commutation is reminiscent of the asymmetrical dependency analysis enabled by the *conditional* stubborn set technique [34], and the execution of inputs under focus may be explained by means of sleep sets. While it may be possible to formally derive our compressed semantics by instantiating abstract POR techniques to our setting, we have not explored this possibility in detail[5]. Concerning our reduced semantics, it may be seen as an application of the sleep set technique

[5] Although this would be an interesting question, we do not expect that any improvement of compression would come out of it. Indeed, compression can be argued to be maximal in terms of eliminating redundant traces without analysing data: for any compressed trace there is a way to choose messages and modify tests to obtain a concrete execution which does not belong to the equivalence class of any other compressed trace.

(or even as a reformulation of Anisimov's and Knuth's characterization of lexicographic normal forms) but the real contribution with this technique is to have formulated it in such a way (see Definition 5.4) that it can be implemented without requiring an *a priori* knowledge of data dependencies: it allows us to eliminate redundant traces on-the-fly as data (in)dependency is discovered by the constraint resolution procedure (as explained in Section 6.5)—in this sense, it may be viewed as a case of dynamic POR.

Narrowing the discussion a bit more, we now focus on the fact that our techniques are designed for the verification of equivalence properties. This requirement turns several seemingly trivial observations into subtle technical problems. For instance, ideas akin to compression are often applied without justification (*e.g.* in [44, 45, 40]) because they are obvious when one does reachability rather than equivalence checking. To understand this, it is important to distinguish between two very different ways of applying POR to equivalence checking (independently of the precise equivalence under consideration). The first approach is to reduce a system such that the reduced system and the original systems are equivalent. In the second approach, one only requires that two reduced systems are equivalent iff the original systems are equivalent. The first approach seems to be more common in the POR literature (where one finds, *e.g.* reductions that preserve LTL-satisfiability [11] or bisimilarity [37]) though there are instances of the second approach (*e.g.* for Petri nets [33]). In the present work, we follow the second approach: neither of our two reduction techniques preserves trace equivalence. This allows stronger reductions but requires extra care: one has to ensure that the independencies used in the reduction of one process are also meaningful for the other processes; in other words, reduction has to be symmetrical. We come back to these two different approaches later, when discussing specific POR techniques for security.

7.2. **Security applications.** The idea of applying POR to the verification of security protocols dates back, at least, to the work of Clarke *et al.* [25, 26]. In this work, the authors remark that traditional POR techniques cannot be directly applied to security mainly because "[they] must keep track of knowledge of various agents" and "[their] logic can refer to this knowledge in a meaningful way". This led them to define a notion of *semi-invisible actions* (output actions, that cannot be swapped after inputs but only before them) and design a reduction that prioritizes outputs and performs them in a fixed order. Compared to our work, this reduction is much weaker (even weaker than compression only), only handles a finite set of messages, and only focuses on reachability properties checking.

In [30], the authors develop "state space reduction" techniques for the Maude-NRL Protocol Analyzer (Maude-NPA). This tool proceeds by backwards reachability analysis and treats at the same level the exploration of protocol executions and attacker's deductions. Several reductions techniques are specific to this setting, and most are unrelated to partial order reduction in general, and to our work in particular. We note that the lazy intruder techniques from [30] should be compared to what is done in constraint resolution procedures (*e.g.* the one used in Apte) rather than to our work. A simple POR technique used in Maude-NPA is based on the observation that inputs can be executed in priority in the backwards exploration, which corresponds to the fact that we can execute outputs first in forward explorations. We note again that this is only one aspect of the focused strategy, and that it is not trivial to lift this observation from reachability to trace equivalence. Finally, a "transition subsumption" technique is described for Maude-NPA. While highly non-trivial due to the technicalities of the model, this is essentially a tabling technique rather than a partial order reduction. Though it does yield a significant state space reduction (as shown in

the experiments [30]) it falls short of exploiting independencies fully, and has a potentially high computational cost (which is not evaluated in the benchmarks of [30]).

In [32], Fokkink *et al.* model security protocols as labeled transition systems whose states contain the control points of different agents as well as previously outputted messages. They devise some POR technique for these transition systems, where output actions are prioritized and performed in a fixed order. In their work, the original and reduced systems are trace equivalent *modulo* outputs (the same traces can be found after removing output actions). The justification for their reduction would fail in our setting, where we consider standard trace equivalence with observable outputs. More importantly, their requirement that a reduced system should be equivalent to the original one makes it impossible to swap input actions, and thus reductions such as the execution under focus of our compressed semantics cannot be used. The authors leave as future work the problem of combining their algorithm with symbolic executions, in order to be able to lift the restriction to a finite number of messages.

Cremers and Mauw proposed [29] a reduction technique for checking secrecy in security protocols. Their method allows to perform outputs eagerly, as in our compressed semantics. It also uses a form of *sleep set* technique to avoid redundant interleavings of input actions. In addition to being applicable only for reachability property, the algorithm of [29] works under the assumption that for each input only finitely many input messages need to be considered. The authors identify as important future work the need to lift their method to the symbolic setting.

Earlier work by Mödersheim *et al.* has shown how to combine POR techniques with symbolic semantics [40] in the context of reachability properties for security protocols. This has led to high efficiency gains in the OFMC tool of the AVISPA platform [7]. While their reduction is very limited, it brings some key insight on how POR may be combined with symbolic execution. For instance, their reduction imposes a dependency constraint (called *differentiation* constraint in their work) on the interleavings of

$$\{\mathtt{in}(c, x).\mathtt{out}(c, m),\ \mathtt{in}(d, y).\mathtt{out}(d, m')\}.$$

Assuming that priority is given to the process working on channel c, this constraint enforces that any symbolic interleaving of the form $\mathtt{in}(d, M').\mathtt{out}(d, w').\mathtt{in}(c, M).\mathtt{out}(c, w)$ would only be explored for instances of M that depend on w'. Our reduced semantics constrains patterns of arbitrary size (instead of just size 2 diamond patterns as above) by means of dependency constraints. Going back to Example 5.5, their technique will only be able (at most) to exploit the dependencies depicted in plain blue arrows, and they will not consider the one represented by the dashed 2-arrow. Moreover, while we generate dependency constraints on the fly, they implement their technique by looking for such a pattern afterwards. This causes a tradeoff between reduction and the cost of redundancy detection: their technique fails to detect all patterns of this kind. Besides these differences, we note that Mödersheim et al. use dependency constraints to guide a dedicated constraint resolution procedure, while we chose to treat constraint resolution (almost) as a black box, and leave it unchanged. Finally, we recall that our POR technique has been designed to be sound and complete for trace equivalence checking as well as reachability checking.

Finally, in [10], the authors of the present paper extend some of the results presented here. Instead of considering the syntactic fragment of simple processes, we work under the more general semantical assumption of *action-determinism*. We show that compression and reduction can be extended to that case, preserving the main result: the induced equivalences

coincide. However, that work is completely carried out in concrete rather than symbolic semantics. Thus, this development should be viewed as being orthogonal to the one carried out in the present paper. The main ideas behind the integration in symbolic semantics and Apte would apply to the action-deterministic case as well. The line of research followed in [10], that consists in extending the supported fragment for our POR techniques, is still open: it would be interesting to support processes that are not action-deterministic, which are commonplace when analysing anonymity or unlinkability scenarios.

8. Conclusion

We have developed two POR techniques that are adequate for verifying trace equivalence properties between simple processes. The first refinement groups actions in blocks, while the second one uses dependency constraints to restrict to minimal interleavings among a class of permutations. In both cases, the refined semantics has less traces, yet we show that the associated trace equivalence coincides with the standard one. We have effectively implemented these refinements in Apte, and shown that they yield the expected, significant benefit.

We claim that our POR techniques – at least compression – and the significant optimisations they allow are generic enough to be applicable to other verification methods as long as they perform forward symbolic executions. In addition to the integration in Apte we have extensively discussed, we also have successfully done so in Spec [35]. Furthermore, parts of our POR techniques have been independently integrated and implemented in the distributed version of Akiss[6].

We are considering several directions for future work. Regarding the theoretical results presented here, it is actually possible to slightly relax the syntactic condition we imposed on processes by an action-determinism hypothesis and apply our reduction techniques on replicated processes [10]. The question of whether the action-determinism condition can be removed without degrading the reductions too much is left open. Another interesting direction would be to adapt our techniques for verification methods based on *backward search* instead of *forward search* as is the case in this paper. We also believe that stronger reductions can be achieved: for instance, exploiting symmetries should be very useful for dealing with multiple sessions. Regarding the practical application of our results, we can certainly go further. We could investigate the role of the particular choice of the order $\prec$, to determine heuristics for maximising the practical impact of reduction.

Acknowledgements. We would like to thank Vincent Cheval for interesting discussions and comments, especially on Section 6.

References

[1] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proc. 28th Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115. ACM Press, 2001.
[2] M. Abadi and C. Fournet. Private authentication. *Theoretical Computer Science*, 322(3):427–476, 2004.
[3] P. Abdulla, S. Aronis, B. Jonsson, and K. Sagonas. Optimal dynamic partial order reduction. *ACM SIGPLAN Notices*, 49(1):373–384, 2014.
[4] J.-M. Andreoli. Logic programming with focusing proofs in linear logic. *J. Log. Comput.*, 2(3), 1992.

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

[5] M. Arapinis, T. Chothia, E. Ritter, and M. Ryan. Analysing unlinkability and anonymity using the applied pi calculus. In *Proc. 23rd Computer Security Foundations Symposium (CSF'10)*, pages 107–121. IEEE Comp. Soc. Press, 2010.

[6] A. Armando, R. Carbone, L. Compagna, J. Cuellar, and L. T. Abad. Formal analysis of saml 2.0 web browser single sign-on: Breaking the saml-based single sign-on for google apps. In *Proc. 6th ACM Workshop on Formal Methods in Security Engineering (FMSE 2008)*, pages 1–10, 2008.

[7] A. Armando et al. The AVISPA Tool for the automated validation of internet security protocols and applications. In *Proc. 17th Int. Conference on Computer Aided Verification (CAV'05)*, LNCS. Springer, 2005.

[8] A. Armando et al. The AVANTSSAR platform for the automated validation of trust and security of service-oriented architectures. In *Tools and Algorithms for the Construction and Analysis of Systems - 18th International Conference, TACAS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, pages 267–282, 2012.

[9] D. Baelde, S. Delaune, and L. Hirschi. A reduced semantics for deciding trace equivalence using constraint systems. In *Proc. of POST'14*. Springer, 2014.

[10] D. Baelde, S. Delaune, and L. Hirschi. Partial order reduction for security protocols. In L. Aceto and D. de Frutos-Escrig, editors, *Proceedings of the 26th International Conference on Concurrency Theory (CONCUR'15)*, volume 42 of *Leibniz International Proceedings in Informatics*, pages 497–510, Madrid, Spain, Sept. 2015. Leibniz-Zentrum für Informatik.

[11] C. Baier and J.-P. Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[12] M. Baudet. Deciding security of protocols against off-line guessing attacks. In *Proc. 12th Conference on Computer and Communications Security*. ACM, 2005.

[13] B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *Proc. 14th Computer Security Foundations Workshop (CSFW'01)*, pages 82–96. IEEE Comp. Soc. Press, 2001.

[14] B. Blanchet, M. Abadi, and C. Fournet. Automated verification of selected equivalences for security protocols. *Journal of Logic and Algebraic Programming*, 2008.

[15] R. Chadha, Ş. Ciobâcă, and S. Kremer. Automated verification of equivalence properties of cryptographic protocols. In *Proc. 21th European Symposium on Programming Languages and Systems (ESOP'12)*, LNCS. Springer, 2012.

[16] R. Chadha, Ş. Ciobâcă, and S. Kremer. Automated verification of equivalence properties of cryptographic protocols. In *Programming Languages and Systems*, pages 108–127. Springer, 2012.

[17] V. Cheval. APTE: `http://projects.lsv.ens-cachan.fr/APTE/`, 2011.

[18] V. Cheval. *Automatic verification of cryptographic protocols: privacy-type properties*. Thèse de doctorat, Laboratoire Spécification et Vérification, ENS Cachan, France, Dec. 2012.

[19] V. Cheval. Apte: an algorithm for proving trace equivalence. In *Proc. TACAS'14*, 2014.

[20] V. Cheval, H. Comon-Lundh, and S. Delaune. Trace equivalence decision: Negative tests and non-determinism. In *Proc. 18th Conference on Computer and Communications Security (CCS'11)*. ACM Press, 2011.

[21] V. Cheval, H. Comon-Lundh, and S. Delaune. A procedure for deciding symbolic equivalence between sets of constraint systems. *Information and Computation*, 2016. To appear.

[22] V. Cheval, V. Cortier, and S. Delaune. Deciding equivalence-based properties using constraint solving. *Theoretical Computer Science*, 492:1–39, June 2013.

[23] V. Cheval and L. Hirschi. sources of APTE, 2015. `https://github.com/APTE/APTE`.

[24] Y. Chevalier and M. Rusinowitch. Decidability of symbolic equivalence of derivations. *Journal of Automated Reasoning*, 48(2), 2012.

[25] E. Clarke, S. Jha, and W. Marrero. Partial order reductions for security protocol verification. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 503–518. Springer, 2000.

[26] E. M. Clarke, S. Jha, and W. R. Marrero. Efficient verification of security protocols using partial-order reductions. *International Journal on Software Tools for Technology Transfer*, 4(2):173–188, 2003.

[27] V. Cortier and B. Smyth. Attacking and fixing Helios: An analysis of ballot secrecy. *Journal of Computer Security*, 21(1):89–148, 2013.

[28] C. J. F. Cremers. The Scyther Tool: Verification, falsification, and analysis of security protocols. In *Proc. 20th International Conference on Computer Aided Verification (CAV'08)*, LNCS. Springer, 2008.

[29] C. J. F. Cremers and S. Mauw. Checking secrecy by means of partial order reduction. In *System Analysis and Modeling*. Springer, 2005.
[30] S. Escobar, C. Meadows, J. Meseguer, and S. Santiago. State space reduction in the maude-nrl protocol analyzer. *Inf. Comput.*, 238:157–186, 2014.
[31] C. Flanagan and P. Godefroid. Dynamic partial-order reduction for model checking software. In *ACM Sigplan Notices*, volume 40, pages 110–121. ACM, 2005.
[32] W. Fokkink, M. T. Dashti, and A. Wijs. Partial order reduction for branching security protocols. In *Proceedings of ACSD'10*. IEEE, 2010.
[33] P. Godefroid. Using partial orders to improve automatic verification methods. In *Computer-Aided Verification*, pages 176–185. Springer Berlin Heidelberg, 1991.
[34] P. Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems - An Approach to the State-Explosion Problem*, volume 1032 of *Lecture Notes in Computer Science*. Springer, 1996.
[35] L. Hirschi. SPEC with dependency constraints. `http://www.lsv.ens-cachan.fr/~hirschi/spec.php`. Accessed: 2017-04-04.
[36] L. Hirschi and S. Delaune. A survey of symbolic methods for establishing equivalence-based properties in cryptographic protocols. *Journal of Logical and Algebraic Methods in Programming*, 2016. To appear.
[37] M. Huhn, P. Niebert, and H. Wehrheim. Partial order reductions for bisimulation checking. In V. Arvind and R. Ramanujam, editors, *Foundations of Software Technology and Theoretical Computer Science, 18th Conference, Chennai, India, December 17-19, 1998, Proceedings*, volume 1530 of *Lecture Notes in Computer Science*, pages 271–282. Springer, 1998.
[38] S. Meier, B. Schmidt, C. J. F. Cremers, and D. Basin. The tamarin prover for the symbolic analysis of security protocols. In *Proc. International Conference on Computer Aided Verification (CAV'13)*, pages 696–701. Springer, 2013.
[39] J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proc. 8th ACM Conference on Computer and Communications Security (CCS'01)*. ACM Press, 2001.
[40] S. Mödersheim, L. Viganò, and D. A. Basin. Constraint differentiation: Search-space reduction for the constraint-based analysis of security protocols. *Journal of Computer Security*, 18(4):575–618, 2010.
[41] D. Peled. Ten years of partial order reduction. In *Proc. 10th International Conference on Computer Aided Verification, CAV'98*, volume 1427 of *Lecture Notes in Computer Science*. Springer, 1998.
[42] M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *Proc. 14th Computer Security Foundations Workshop (CSFW'01)*, pages 174–190. IEEE Comp. Soc. Press, 2001.
[43] S. Santiago, S. Escobar, C. Meadows, and J. Meseguer. A formal definition of protocol indistinguishability and its verification using Maude-NPA. In *Security and Trust Management*, pages 162–177. Springer, 2014.
[44] K. Sen and G. Agha. Automated systematic testing of open distributed programs. In *Fundamental Approaches to Software Engineering*, pages 339–356. Springer, 2006.
[45] S. Tasharofi, R. K. Karmani, S. Lauterburg, A. Legay, D. Marinov, and G. Agha. Transdpor: A novel dynamic partial-order reduction technique for testing actor programs. In *Formal Techniques for Distributed Systems*, pages 219–234. Springer, 2012.
[46] A. Tiu. Spec: `http://users.cecs.anu.edu.au/~tiu/spec/`, 2010.
[47] A. Tiu and J. E. Dawson. Automating open bisimulation checking for the spi calculus. In *Proc. 23rd IEEE Computer Security Foundations Symposium (CSF'10)*, pages 307–321. IEEE Computer Society Press, 2010.

Appendix A. Notations

Symbol	Description	Reference
$\rightarrow$	transition for concrete processes	Figure 2
$\Rightarrow$	$\rightarrow$ up-to non-observable actions	Section 2.2
$\sim$	static equivalence	Definition 2.7
$\sqsubseteq$	trace inclusion for concrete processes	Definition 2.9
$\approx$	trace equivalence for concrete processes	Definition 2.9
$\twoheadrightarrow$	focused semantics	Figure 3
$\rightarrow_c$	compressed semantics	Figure 4
$\sqsubseteq_c$	trace inclusion induced by $\rightarrow_c$	Definition 3.6
$\approx_c$	trace equivalence induced by $\rightarrow_c$	Definition 3.6
$\mapsto$	symbolic semantics	Figure 5
$\mapsto\!\!\!\!\twoheadrightarrow$	focused symbolic semantics	Figure 6
$\mapsto_c$	compressed symbolic semantics	Figure 6
$\sqsubseteq^s$	trace inclusion induced by $\mapsto$	Definition 4.7
$\sqsubseteq^s_c$	trace inclusion induced by $\mapsto_c$	Definition 4.8
$\approx^s$	trace equivalence induced by $\mapsto$	Definition 4.7
$\approx^s_c$	trace equivalence induced by $\mapsto_c$	Definition 4.8
$\overrightarrow{X} \ltimes \overrightarrow{w}$	dependency constraint	Definition 5.2
$\mathrm{Deps}(\mathsf{tr})$	dependency constraints induced by a trace	Definition 5.4
$\sqsubseteq^s_r$	trace inclusion up-to dependency constraints	Definition 5.6
$\approx^s_r$	trace equivalence up-to dependency constraints	Definition 5.6
$\|\|$	independence of blocks	Definition 5.7
$\equiv_\Phi$	equivalence of two traces	Definition 5.9
$\lceil(\mathcal{P};\Phi;\mathcal{S})\rceil$	associated extented symbolic process	Section 6.1
$\lfloor(\mathcal{P};\Phi;\mathcal{S};\mathcal{S}^+)\rfloor$	associated symbolic process	Section 6.1
$\prec^+$	symbolic inclusion of sets of extended symbolic processes	Definition 6.2
$\sim^+$	symbolic equivalence of sets of extended symbolic processes	Definition 6.2
$\mapsto^{\mathsf{A}}$	Apte exploration step	Section 6.1
$\mapsto^{\mathsf{A1}}$	first part of Apte exploration step	Section 6.1
$\mapsto^{\mathsf{A2}}$	second part of Apte exploration step	Section 6.1
$\mapsto^{\mathsf{A}}_c$	compressed version of $\mapsto^{\mathsf{A}}$	Section 6.4
$\mapsto^{\mathsf{A}}_r$	reduced version of $\mapsto^{\mathsf{A}}_c$	Section 6.5
$\approx^{\mathsf{A}}$	equivalence induced by $\mapsto^{\mathsf{A}}$	Definition 6.4
$\approx^{\mathsf{A}}_c$	equivalence induced by $\mapsto^{\mathsf{A}}_c$	Section 6.4
$\approx^{\mathsf{A}}_r$	equivalence induced by $\mapsto^{\mathsf{A}}_r$	Section 6.5

Appendix B. Proofs of Section 3

Lemma 3.8. *Let A, A' be two extended simple processes and tr, tr' be such that $\mathsf{tr} =_{\mathcal{I}_a} \mathsf{tr}'$. We have that $A \stackrel{\mathsf{tr}}{\Rightarrow} A'$ if, and only if, $A \stackrel{\mathsf{tr}'}{\Rightarrow} A'$.*

Proof. It suffices to establish that $A \stackrel{\alpha\cdot\alpha'}{\Longrightarrow} A'$ implies $A \stackrel{\alpha'\cdot\alpha}{\Longrightarrow} A'$ for any $\alpha\, \mathcal{I}_a\, \alpha'$.

- Assume that we have $A \xrightarrow{\mathtt{out}(c_i,w_i)} A_i \xrightarrow{\mathtt{out}(c_j,w_j)} A'$ with $c_i \neq c_j$. Because we are considering simple processes, the two actions must be concurrent. More specifically, our process A must be of the form $(\{P_i, P_j\} \uplus \mathcal{P}_r; \Phi)$ with P_i (resp. P_j) being a basic process on channel c_i (resp. c_j). We assume that in our sequence of reductions, τ actions pertaining to P_i are all executed before reaching A_i, and that τ actions pertaining to $\mathcal{P}_r$ are executed last. This is without loss of generality, because a τ action on a given basic process can easily be permuted with actions taking place on another basic process, since it does not depend on the context and has no effect on the frame. Thus we have that $A_i = (\{P'_i, P_j\} \uplus \mathcal{P}_r; \Phi \uplus \{w_i \triangleright m_i\})$, $A' = (\{P'_i; P'_j\} \uplus \mathcal{P}'_r; \Phi \uplus \{w_i \triangleright m_i, w_j \triangleright m_j\})$. Since the τ actions taking place on $\mathcal{P}'_r$ rely neither on the frame nor on the first two basic processes, we easily obtain the permuted execution:
$$\begin{array}{rl} A & \xrightarrow{\mathtt{out}(c_j,w_j)} \quad (\{P_i, P'_j\} \uplus \mathcal{P}_r; \Phi \uplus \{w_j \triangleright m_j\}) \\ & \xrightarrow{\mathtt{out}(c_i,w_i)} \quad (\{P'_i, P'_j\} \uplus \mathcal{P}'_r; \Phi \uplus \{w_i \triangleright m_i, w_j \triangleright m_j\}) \end{array}$$
- The permutation of two input actions on distinct channels is very similar. In this case, the frame does not change at all, and the order in which messages are derived from the frame does not matter. Moreover, the instantiation of the input variable on one basic process has no impact on the other ones.
- Assume that we have $A \xrightarrow{\mathtt{out}(c_i,w_i)} A_i \xrightarrow{\mathtt{in}(c_j,M)} A'$ with $c_i \neq c_j$ and $w_i \notin fv(M)$. Again, the two actions are concurrent, and we can assume that τ actions are organized conveniently so that A is of the form $(\{P_i, P_j\} \uplus \mathcal{P}_r; \Phi)$ with P_i (resp. P_j) a basic process on c_i (resp. c_j); A_i is of the form $(\{P'_i, P_j\} \uplus \mathcal{P}_r; \Phi \uplus \{w_i \triangleright m_i\})$; and A' is of the form $(\{P'_i, P'_j\} \uplus \mathcal{P}'_r; \Phi \uplus \{w_i \triangleright m_i\})$. As before, the τ actions from $\mathcal{P}_r$ to $\mathcal{P}'_r$ are easily moved around. Additionally, $w_i \notin fv(M)$ implies $fv(M) \subseteq \mathrm{dom}(\Phi)$ and thus we have:
$$(\{P_i, P_j\} \uplus \mathcal{P}_r; \Phi) \xrightarrow{\mathtt{in}(c_j,M)} (\{P_i, P'_j\} \uplus \mathcal{P}_r; \Phi)$$
The next step is trivial:
$$(\{P_i, P'_j\} \uplus \mathcal{P}_r; \Phi) \xrightarrow{\mathtt{out}(c_i,w_i)} (\{P'_i, P'_j\} \uplus \mathcal{P}'_r; \Phi \uplus \{w_i \triangleright m_i\})$$
- We also have to perform the reverse permutation, but we shall not detail it; this time we are delaying the derivation of M from the frame, and it only gets easier. □

Proposition 3.10. *Let A, A' be two initial simple processes, and tr be a trace made of proper blocks such that $A \overset{\mathsf{tr}}{\Rightarrow} A'$. Then, we have that $A \xrightarrow{\mathsf{tr}}_c A'$.*

Proof. We first observe that $A \overset{\mathsf{tr}}{\Rightarrow} A'$ implies $A \xrightarrow{\mathsf{tr}}_{o^*} A'$ if A' is initial and tr is a (possibly empty) sequence of output actions on the same channel. We prove this by induction on the sequence of actions. If it is empty, we can conclude using one of the Proper rules because $A = A'$ is initial. Otherwise, we have:
$$A \xrightarrow{\mathtt{out}(c,w)} A'' \overset{\mathsf{tr}}{\Rightarrow} A'.$$
We obtain $A'' \xrightarrow{\mathsf{tr}}_{o^*} A'$ by induction hypothesis, and conclude using rules Tau and Out.

The next step is to show that $A \overset{\mathsf{tr}}{\Rightarrow} A'$ implies $A \xrightarrow{\mathsf{tr}}_{i^*} A'$, if A' is initial and tr is the concatenation of a (possibly empty) sequence of inputs and a non-empty sequence of outputs, all on the same channel. This is easily shown by induction on the number of input actions. If there are none we use the previous result, otherwise we conclude by induction hypothesis

and using rules Tau and In. Otherwise, the first output action allows us to conclude from the previous result and rules Tau and Out.

We can now show that $A \stackrel{\mathsf{tr}}{\Rightarrow} A'$ implies $A \stackrel{\mathsf{tr}}{\twoheadrightarrow}_{i^+} A'$ if A' is initial and tr is a proper block. Indeed, we must have

$$A \xrightarrow{\mathtt{in}(c,M)} A'' \stackrel{\mathsf{tr}'}{\Longrightarrow} A'$$

which allows us to conclude using the previous result and rules Tau and In.

We finally obtain that $A \stackrel{\mathsf{tr}}{\Rightarrow} A'$ implies $A \stackrel{\mathsf{tr}}{\to}_c A'$ when A and A' are initial *simple* processes and tr is a sequence of proper blocks. This is done by induction on the number of blocks. The base case is trivial. Because A is initial, the execution of its basic processes can only start with observable actions, thus only one basic process is involved in the execution of the first block. Moreover, we can assume without loss of generality that the execution of this first block results in another initial process: indeed the basic process resulting from that execution is either in the final process A', which is initial, or it will perform another block, *i.e.* it can perform τ actions followed by an input, in which case we can force those τ actions to take place as early as possible. Thus we have

$$A \stackrel{\mathsf{b}}{\Rightarrow} A'' \stackrel{\mathsf{tr}'}{\Longrightarrow} A'$$

where b is a proper block, and we conclude using the previous result and the induction hypothesis. □

Before proving Theorem 3.11, we establish the following result.

Proposition B.1. *Let* tr *be a trace of observable actions such that, for any channel* c *occurring in the trace, it appears first in an input action. There exists a sequence of proper blocks* tr_{io} *and a sequence of improper blocks* tr_i *such that* $\mathsf{tr} =_{\mathcal{I}_a} \mathsf{tr}_{io} \cdot \mathsf{tr}_i$.

Proof. We proceed by induction on the length of tr, and distinguish two cases:

- If tr has no output action then, by swapping input actions on distinct channels, we reorder tr so as to obtain $\mathsf{tr}_i = \mathsf{tr}^{c_1} \cdot \ldots \cdot \mathsf{tr}^{c_n} =_{\mathcal{I}_a} \mathsf{tr}$ where the c_i's are pairwise distinct and tr^{c_i} is an improper block on channel c_i.
- Otherwise, there must be a decomposition $\mathsf{tr} = \mathsf{tr}_1 \cdot \mathtt{out}(c, w) \cdot \mathsf{tr}_2$ such that tr_1 does not contain any output. We can perform swaps involving input actions of tr_1 on all channel $c' \neq c$, so that they are delayed after the first output on c. We obtain $\mathsf{tr} =_{\mathcal{I}_a} \mathtt{in}(c, M_1) \cdot \ldots \cdot \mathtt{in}(c, M_n) \cdot \mathtt{out}(c, w) \cdot \mathsf{tr}'_1 \cdot \mathsf{tr}_2$ with $n \geq 1$. Next, we swap output actions on channel c from $\mathsf{tr}'_1 \cdot \mathsf{tr}_2$ that are not preceded by another input on c, so as to obtain

  $$\mathsf{tr} =_{\mathcal{I}_a} \mathtt{in}(c, M_1) \ldots \mathtt{in}(c, M_n) \cdot \mathtt{out}(c, w) \cdot \mathtt{out}(c, w_1) \ldots \mathtt{out}(c, w_m) \cdot \mathsf{tr}'_2$$

  such that either tr'_2 does not contain any action on channel c or the first one is an input action. We have thus isolated a first proper block, and we can conclude by induction hypothesis on tr'_2. □

Note that the above result does not exploit all the richness of $\mathcal{I}_a$. In particular, it never relies on the possibility to swap an input action before an output when the input message does not use the output handled. Indeed, the idea behind compression does not rely on messages. This is no longer the case in Section 5 where we use $\mathcal{I}_a$ more fully.

We finally prove the main result about the compressed semantics relying on Proposition B.1 stated and proved above. Given two simple process $A = (\mathcal{P}; \Phi)$ and $A' = (\mathcal{P}'; \Phi')$, we shall write $\Phi(A) \sim \Phi(A')$ (or even $A \sim A'$) instead of $\Phi \sim \Phi'$.

Theorem 3.11. *Let A and B be two initial simple processes. We have that*

$$A \approx B \iff A \approx_c B.$$

Proof. We prove the two directions separately.
($\Rightarrow$) Let A be an initial simple process such that $A \approx B$ and $A \xrightarrow{\mathsf{tr}}_c A'$. One can easily see that the trace tr must be of the form $\mathsf{tr}_{io} \cdot \mathsf{tr}_i$ where tr_{io} is made of proper blocks and tr_i is a (possibly empty) sequence of inputs on the same channel c_j. We have:

$$A \xrightarrow{\mathsf{tr}_{io}}_c A'' \xrightarrow{\mathsf{tr}_i}_c A'$$

Using Proposition 3.9, we obtain that $A \overset{\mathsf{tr}_{io}}{\Longrightarrow} A''$. We also claim that $A'' \overset{\mathsf{tr}_i}{\Rightarrow} A^+$ for some A^+ having the same frame as A' which is itself equal to the one of A''. This is obvious when tr_i is empty—in that case we can simply choose $A^+ = A' = A''$. Otherwise, the execution of the improper block tr_i results from the application of rule Improper. Except for the fact that this rule "kills" the resulting process, its subderivation simply packages a sequence of inputs, and so we have a suitable A^+. We thus have:

$$A \overset{\mathsf{tr}_{io}}{\Longrightarrow} A'' \overset{\mathsf{tr}_i}{\Rightarrow} A^+$$

By hypothesis, it implies that $B \overset{\mathsf{tr}_{io}}{\Longrightarrow} B''$ and $B \overset{\mathsf{tr}_{io} \cdot \mathsf{tr}_i}{\Longrightarrow} B^+$ with $A'' \sim B''$ and $A^+ \sim B^+$. Relying on the fact that B is a simple process, we have:

$$B \overset{\mathsf{tr}_{io}}{\Longrightarrow} B'' \overset{\mathsf{tr}_i}{\Rightarrow} B^+$$

It remains to establish that $B \xrightarrow{\mathsf{tr}}_c B'$ such that $B' \sim A'$. We can assume that B'' does not have any basic process starting with a test, without loss of generality since forcing τ actions cannot break static equivalence. Further, we observe that B'' is initial. Otherwise, it would mean that a basic process of B is not initial (absurd) or that one of the blocks of tr_{io}, which are maximal for A, is not maximal for B (absurd again, because it contradicts $A \approx B$). This allows us to apply Proposition 3.10 to obtain

$$B \xrightarrow{\mathsf{tr}_{io}}_c B''.$$

This concludes when tr_i is empty, because $B' = B'' \sim A'' = A'$. Otherwise, we note that A^+ cannot perform any action on channel c_j, because the execution of tr_i in the compressed semantics must be maximal. Since $A \approx B$, it must be that B^+ cannot perform any observable action on the channel c_j either. Thus B'' can complete an improper step:

$$B'' \xrightarrow{\mathsf{tr}_i}_c B' \text{ where } B' = (\emptyset; \Phi(B^+)).$$

We can finally conclude that $B \xrightarrow{\mathsf{tr}}_c B'$ with $\Phi(B') = \Phi(B^+) \sim \Phi(A^+) = \Phi(A')$.

($\Leftarrow$) Let A be an initial simple process such that $A \approx_c B$ and $A \overset{\mathsf{tr}}{\Rightarrow} A'$. We "complete" this execution as follows:

- We force τ actions whenever possible.
- If the last action on c in tr is an input, we trigger available inputs on c using a valid public constant as a recipe.
- We trigger all the outputs that are available and not blocked.

We obtain a trace of the form $\mathsf{tr} \cdot \mathsf{tr}^+$. Let A^+ be the process obtained from this trace:

$$A \overset{\mathsf{tr}}{\Rightarrow} A' \overset{\mathsf{tr}^+}{\Longrightarrow} A^+$$

We observe that A^+ is initial. Indeed, for each basic process that performs actions in $\mathsf{tr} \cdot \mathsf{tr}^+$, we have that:

- either the last action on its channel is an output and the basic process is of the form $\mathtt{in}(c, \_).P$ or $\mathtt{out}(c, u).P$ with $\neg \mathit{valid}(u)$,
- or the last action is an input and the basic process is reduced to 0 and disappears, or it is an output which is blocked.

Next, we apply Proposition B.1 to obtain traces tr_{sio} (resp. tr_i) made of proper (resp. improper) blocks, such that $\mathsf{tr} \cdot \mathsf{tr}^+ =_{\mathcal{I}_a} \mathsf{tr}_{io} \cdot \mathsf{tr}_i$. By Lemma 3.8 we know that this permuted trace can also lead to A^+:

$$A \stackrel{\mathsf{tr}_{io}}{\Longrightarrow} A_{io} \stackrel{\mathsf{tr}_i}{\Longrightarrow} A^+$$

As before, we can assume that A_{io} cannot perform any τ action. Under this condition, since A^+ is initial, A_{io} must also be initial.

By Proposition 3.10 we have that $A \xrightarrow{\mathsf{tr}_{io}}_c A_{io}$, and $A \approx_c B$ implies that:

$$B \xrightarrow{\mathsf{tr}_{io}}_c B_{io} \text{ with } \Phi(A_{io}) \sim \Phi(B_{io}).$$

A simple inspection of the PROPER rules shows that a basic process resulting from the execution of a proper block must be initial. Thus, since the whole simple process B is initial, B_{io} is initial too.

Thanks to Proposition 3.9, we have that $B \stackrel{\mathsf{tr}_{io}}{\Longrightarrow} B_{io}$. Our goal is now to prove that we can complete this execution with tr_i. This trace is of the form $\mathsf{tr}^{c_1} \cdot \mathsf{tr}^{c_2} \dots \mathsf{tr}^{c_n}$ where tr^{c_i} contains only inputs on channel c_i and the c_i are pairwise disjoint. Now, we easily see that for each i,

$$A_{io} \stackrel{\mathsf{tr}^{c_i}}{\Longrightarrow} A_i$$

and A_i has no more atomic process on channel c_i. Thus we have $A_{io} \xrightarrow{\mathsf{tr}^{c_i}}_c A_i^0$ with $A_i^0 = (\emptyset; \Phi(A_i))$. Since $A \approx_c B$, we must have some B_i^0 such that:

$$B \xrightarrow{\mathsf{tr}_{io}}_c B_{io} \xrightarrow{\mathsf{tr}^{c_i}}_c B_i^0$$

We can translate this back to the regular semantics, obtaining $B \stackrel{\mathsf{tr}_{io}}{\Longrightarrow} B_{io} \stackrel{\mathsf{tr}^{c_i}}{\Longrightarrow} B_i$. We can now execute all these inputs to obtain an execution of $\mathsf{tr}_{io} \cdot \mathsf{tr}_i$ towards some process B^+:

$$B \stackrel{\mathsf{tr}_{io}}{\Longrightarrow} B_{io} \stackrel{\mathsf{tr}_i}{\Longrightarrow} B^+$$

Permuting those actions, we obtain thanks to Lemma 3.8:

$$B \stackrel{\mathsf{tr}}{\Rightarrow} B' \stackrel{\mathsf{tr}^+}{\Longrightarrow} B^+$$

We observe that $\Phi(B^+) = \Phi(B_{io}) \sim \Phi(A_{io}) \sim \Phi(A^+)$, and it follows that $A' \sim B'$ because those frames have the same domain, which is a subset of that of $\Phi(A^+) \sim \Phi(B^+)$. □