A Reduced Semantics for Deciding Trace Equivalence Using Constraint Systems*

David Baelde[1], Stéphanie Delaune[1], and Lucca Hirschi[1,2]

[1] LSV, ENS Cachan & CNRS & Inria Saclay Île-de-France
[2] ENS Lyon, France

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.* [18] 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 optimization in the form of a reduced symbolic semantics that eliminates redundant interleavings on the fly.

1 Introduction

Security protocols are widely used today to secure transactions that rely on public channels like the Internet, where dishonest users may listen to communications and interfere with them. A secure communication 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.

Formal methods have proved their usefulness for precisely analyzing the security of protocols. In particular, a wide variety of model-checking approaches have been developed to analyse protocols against an attacker who entirely controls the communication network, and several tools are now available to automatically verify cryptographic protocols [8,15,5]. 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 [17,20]. This has lead to tools for verifying reachability-based security properties such as confidentiality [17] or, more recently, equivalence-based properties such as privacy [22,12,10].

* This work has been partially supported by the project JCJC VIP ANR-11-JS02-006, and the Inria large scale initiative CAPPRIS.

In both cases, the practical impact of most of these tools is limited by a typical state explosion problem caused by the exploration of the large number of interleavings in the protocol's behaviour. In standard model-checking approaches for concurrent systems, the interleaving problem is handled using partial order reduction techniques [19]. For instance, the order of execution of two independent (parallel) actions is typically irrelevant for checking reachability. Things become more complex when working with a symbolic semantics: the states obtained from the interleaving of parallel actions will differ, but the sets of concrete states that they represent will have a significant overlap. Earlier work has shown how to limit this overlap [18] in the context of reachability properties for security protocols, leading to high efficiency gains in the OFMC tool of the AVISPA platform [5].

In this paper, we revisit the work of [18] to obtain a partial order reduction technique for the verification of equivalence properties. Specifically, we focus on trace equivalence, requiring that two processes have the same sets of observable traces and perform indistinguishable sequences of outputs. This notion is well-studied and several algorithms and tools support it [9,14,22,12,10]. 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 optimized 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 [18] that would apply equally well for reachability checking. Detailed proofs of our results can be found in [6].

Outline. In Section 2, we introduce our model for security processes. We consider the class of simple processes introduced in [13], with else branches and no replication. Then we present two successive optimizations 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, this is lifted to a symbolic semantics in Section 4. Finally, Section 5 presents the *reduced* semantics which makes use of dependency constraints to remove more interleavings. We conclude in Section 6, mentioning a preliminary implementation that shows efficiency gains in practice and some directions for future work.

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].

2.1 Messages

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, 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 $\mathcal{T}(S)$. Terms in $\mathcal{T}(\mathcal{N} \cup \mathcal{X})$ represent messages and are denoted by u, v, etc. while terms in $\mathcal{T}(\mathcal{W})$ 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 $\mathcal{T}(\mathcal{N})$. We may rely on a sort system for terms, but its details are unimportant for this paper.

To model algebraic properties of cryptographic primitives, we consider an equational theory E. The theory will usually be generated for finite axioms and enjoy nice properties, but these aspects are irrelevant for the present work.

Example 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:

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

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

2.2 Processes

We do not need the full applied pi calculus 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 [13]. 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 1 (basic/simple process). *The set of* basic processes *on $c \in \mathcal{C}$ is defined using the following grammar (below $u, v \in \mathcal{T}(\mathcal{N} \cup \mathcal{X})$ and $x \in \mathcal{X}$):*

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

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.*

For conciseness, we often omit brackets, null processes, and even "`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 (frame). *A* frame Φ *is a substitution whose domain is included in* $\mathcal{W}$ *and image is included in* $\mathcal{T}(\mathcal{N} \cup \mathcal{X})$. *It is written* $\{w \rhd u, \ldots\}$. *A frame is* closed *when its image only contains ground terms.*

An *extended simple proces* (denoted A or B) is a pair made of a simple process and a frame. Similarly, we define *extended basic processes*. Note that we do not have an explicit set of restricted names. Actually, all names are restricted and public ones are explicitly given to the attacker through a frame.

Example 2. We consider the protocol given in [2] designed for transmitting a secret without revealing its identity to other participants. In this protocol, A is willing to engage in communication with B and wants to reveal its identity to B. However, A does not want to compromise its privacy by revealing its identity or the identity of B more broadly. The participants A and B proceed as follows:

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

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 1, a session of role A played by agent a (with private key ska) with b (whose public key is pkb) can be modeled as follows:

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

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 proces where $N = \mathsf{adec}(y, skb)$.

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

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:

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

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.

2.3 Semantics

We first define a standard concrete semantics. Thus, in this section, we work only with closed extended processes, *i.e.,* processes $(\mathcal{P};\Phi)$ where $fv(\mathcal{P}) = \emptyset$.

$$\begin{array}{lll}
\textsc{Then} & (\{\texttt{if}\ u = v\ \texttt{then}\ Q_1\ \texttt{else}\ Q_2\} \uplus \mathcal{P};\Phi) \xrightarrow{\tau} (\{Q_1\} \uplus \mathcal{P};\Phi) & \text{if } u =_{\mathsf{E}} v \\
\textsc{Else} & (\{\texttt{if}\ u = v\ \texttt{then}\ Q_1\ \texttt{else}\ Q_2\} \uplus \mathcal{P};\Phi) \xrightarrow{\tau} (\{Q_2\} \uplus \mathcal{P};\Phi) & \text{if } u \neq_{\mathsf{E}} v \\
\textsc{In} & (\{\texttt{in}(c,x).Q\} \uplus \mathcal{P};\Phi) \xrightarrow{\texttt{in}(c,M)} (\{Q\{x \mapsto u\}\} \uplus \mathcal{P};\Phi) & \\
 & \qquad \text{if } M \in \mathcal{T}(\mathrm{dom}(\Phi)) \text{ and } M\Phi = u & \\
\textsc{Out} & (\{\texttt{out}(c,u).Q\} \uplus \mathcal{P};\Phi) \xrightarrow{\texttt{out}(c,w)} (\{Q\} \uplus \mathcal{P};\Phi \cup \{w \rhd u\}) & \\
 & \qquad \text{if } w \text{ is a fresh variable} &
\end{array}$$

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

A process may input any term that an attacker can build (rule In): $\{x \mapsto u\}$ is a substitution that replaces any occurrence of x with u. 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.

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

Example 3. Consider the process $(\mathcal{P};\Phi_0)$ introduced in Example 2. We have:

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

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:

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

2.4 Trace Equivalence

Many interesting security properties, such as privacy-type properties studied *e.g.,* in [4], are formalized using the notion of *trace equivalence.* We first introduce the notion of *static equivalence* that compares sequences of messages.

Definition 3 (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:*

$$M\Phi =_{\mathsf{E}} N\Phi \;\Leftrightarrow\; M\Phi' =_{\mathsf{E}} N\Phi' \textit{ for any terms } M, N \in \mathcal{T}(\mathrm{dom}(\Phi)).$$

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.

Example 4. Consider the frame Φ given in Example 3 and the frame Φ' below:

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

Actually, we have that $\Phi \sim \Phi'$. Intuitively, the equivalence 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 $\Phi_+ = \Phi \uplus \{w_5 \triangleright n_a\}$ and $\Phi'_+ = \Phi' \uplus \{w_5 \triangleright n_a\}$, then the two frames Φ_+ and Φ'_+ are not in static equivalence anymore. Let $M = \mathsf{aenc}(\langle w_5, w_1\rangle, w_2)$ and $N = w_3$. We have that $M\Phi_+ =_{\mathsf{E_{aenc}}} N\Phi_+$ whereas $M\Phi'_+ \neq_{\mathsf{E_{aenc}}} N\Phi'_+$.

Definition 4 (trace equivalence). *Let A and B be two simple processes. We have that $A \sqsubseteq B$ if, for every sequence of actions* tr *such that $A \overset{\mathsf{tr}}{\Longrightarrow} (\mathcal{P};\Phi)$, there exists $(\mathcal{P}';\Phi')$ such that $B \overset{\mathsf{tr}}{\Longrightarrow} (\mathcal{P}';\Phi')$ 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 5. Intuitively, the private authentication protocol presented in Example 2 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:

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

For illustration purposes, we also consider a variant of the process Q, denoted Q_0, where its `else` branch has been replaced by `else 0`. 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 $\Phi = \Phi_0 \uplus \{w_3 \triangleright \mathsf{aenc}(\langle \mathsf{pk}(ska), \langle n_b, \mathsf{pk}(skb)\rangle\rangle, \mathsf{pk}(ska))\}$.

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 $(Q_0(skb, \mathsf{pk}(ska)); \Phi_0) \not\approx (Q_0(skb, \mathsf{pk}(ska')); \Phi_0)$. Actually, it can been shown that $(Q(skb, \mathsf{pk}(ska)); \Phi_0) \approx (Q(skb, \mathsf{pk}(ska')); \Phi_0)$. This is a non trivial equivalence that can be checked using the tool APTE [11] 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 Reduction Based on Grouping Actions

A large number of possible interleavings results into multiple occurrences of identical states. The compression step lifts a common optimization that partly tackles this issue in the case of reachability properties to trace equivalence. 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 6. 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 optimization, 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 optimization.

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 $P = \mathtt{in}(c,x).P'$ for some channel c, *i.e.,* each basic process composing A starts with an input action.

Example 7. Continuing Example 2, $(\{P(ska,\mathsf{pk}(skb)),Q(skb,\mathsf{pk}(ska))\};\Phi_0)$ is not initial. Instead, we may consider $(\{P_{\mathsf{init}},Q(skb,\mathsf{pk}(ska))\};\Phi_0)$ where:

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

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

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 1 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 8. Going back to Example 5, we have that:

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

where Φ is as given in Example 5. As illustrated by the prooftree below, we have also $(Q_0(skb,\mathsf{pk}(ska));\Phi_0) \xrightarrow{\mathsf{tr}}\!\!\twoheadrightarrow_{i^+} (\bot;\Phi_0)$ with $\mathsf{tr} = \mathtt{in}(c_B,\mathsf{aenc}(\langle w_1,w_1\rangle,w_2))$.

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

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

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

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

$$\textsc{Improper}\quad \frac{}{(0;\Phi) \xrightarrow{\epsilon}\!\!\!\!\rightarrow_{i^*} (\bot;\Phi)}$$

Fig. 1. Focused semantics on extended basic processes

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

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

Then we define the compressed reduction $\rightarrow_c$ between extended simple processes as the least reflexive transitive relation satisfying the following rules:

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

A basic process is allowed to *properly* end a block execution when it has performed outputs and it cannot perform any more. 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 improper termination of a block, when the basic process that is currently executing is not able to perform any visible action (input or output) and it has not yet performed an output.

Example 9. Continuing Example 8, using the rule BLOCK, we can derive that:

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

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 IMPROPER). 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 IMPROPER may seem too strong. Actually, 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: it is in fact sufficient to consider such improper blocks at the end of traces.

Example 10. 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 define the notion of *compressed trace equivalence* (resp. *inclusion*) accordingly relying on $\rightarrow_c$ instead of $\Rightarrow$, and we denote them $\approx_c$ (resp. $\sqsubseteq_c$).

3.2 Soundness and Completeness

The purpose of this section is to establish the soundness and completeness of the compressed semantics. More precisely, we show that the two relations $\approx$ and $\approx_c$ coincide on initial simple processes.

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 $\mathsf{act}\cdot\mathsf{act}' =_{\mathcal{I}_a} \mathsf{act}'\cdot\mathsf{act}$ for all act and act' 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 1. *Let A, A' be two simple extended 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 1. *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 $A \xRightarrow{\mathsf{tr}} A'$.*

Proposition 2. *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 $A \xrightarrow{\mathsf{tr}}_c A'$.*

Theorem 1. *Let A and B be two initial simple processes. We have that:*

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

Proof. (Sketch) The main difficulty is that Proposition 2 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 $\mathtt{tr}$ of a process A that is an interleaving of some prefix of proper and improper blocks. We will first complete it to obtain an interleaving of complete blocks and improper blocks. We then reorganize the actions providing an equivalent trace $\mathtt{tr}'$ w.r.t. $=_{\mathcal{I}_a}$ such that $\mathtt{tr}' = \mathtt{tr}_{\text{io}} \cdot \mathtt{tr}_{\text{in}}$ where $\mathtt{tr}_{\text{io}}$ is made of proper blocks and $\mathtt{tr}_{\text{in}}$ is made of improper blocks. For each improper block b of $\mathtt{tr}_{\text{in}}$, we show by applying Lemma 1 and Proposition 2 that A is able to perform $\mathtt{tr}_{\text{io}} \cdot b$ 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 $\mathtt{tr}'$. $\square$

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

Example 11. Let $P = \mathtt{in}(c, x)$ and $Q = \mathtt{in}(c, x).\mathtt{out}(c, n)$. Actually, we have that $(P; \emptyset) \sqsubseteq (Q; \emptyset)$ whereas $(P; \emptyset) \not\sqsubseteq_c (Q; \emptyset)$ since in the compressed semantics Q 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.,* [17,7]. Such a semantics avoids potentially infinite branching of our reduction 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 [7], 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 5 (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 $u, v \in \mathcal{T}(\mathcal{N} \cup \mathcal{X})$.

The first kind of constraint expresses that a recipe X has to use 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 $\mathcal{C}$, we define a dependency order on

$fv^1(\mathcal{C}) \cap \mathcal{X}$ by declaring that x depends on y if, and only if, $\mathcal{S}$ contains a deduction constraint $D \vdash^?_X x$ with $y \in fv^1(\Phi(D))$. For $\mathcal{C}$ to be a well-formed constraint system, we require that the dependency relationship is acyclic and that 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 12. 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 [17,7]. We use a simpler and (slightly) more permissive variant because we are not concerned with constraint solving procedures in this work.

Definition 6 (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 $X\theta \in \mathcal{T}(D_{\mathcal{C}}(X))$ 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 that $(X\theta)(\Phi\lambda) =_{\mathsf{E}} x\lambda$;*
- *for every $u =^? v$ in $\mathcal{S}$, we have that $u\lambda =_{\mathsf{E}} v\lambda$; and*
- *for every $u \neq^? v$ in $\mathcal{S}$, we have that $u\lambda \neq_{\mathsf{E}} v\lambda$.*

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.*

Example 13. Consider again the constraint system $\mathcal{C}$ given in Example 12. 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

From a 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 rely on free variables.

Definition 7 (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 below a standard symbolic semantics for our symbolic processes.

In	$(\mathtt{in}(c,y).P;\Phi;\mathcal{S}) \xmapsto{\mathtt{in}(c,X)} (P\{y \mapsto x\};\Phi;\mathcal{S} \cup \{\mathrm{dom}(\Phi) \vdash^?_X x\})$ where X (resp. x) is a fresh second-order (resp. first-order) variable
Out	$(\mathtt{out}(c,u).P;\Phi;\mathcal{S}) \xmapsto{\mathtt{out}(c,w)} (P;\Phi \cup \{w \rhd u\};\mathcal{S})$ where w is a fresh first-order variable
Then	$(\mathtt{if}\ u = v\ \mathtt{then}\ P\ \mathtt{else}\ Q;\Phi;\mathcal{S}) \xmapsto{\tau} (P;\Phi;\mathcal{S} \cup \{u =^? v\})$
Else	$(\mathtt{if}\ u = v\ \mathtt{then}\ P\ \mathtt{else}\ Q;\Phi;\mathcal{S}) \xmapsto{\tau} (Q;\Phi;\mathcal{S} \cup \{u \neq^? v\})$

From this semantics, we derive our compressed symbolic semantics $\xmapsto{\mathsf{tr}}_c$ following the same pattern as for the concrete semantics. We consider interleavings that execute maximal blocks of actions, and we allow improper termination of a block only at the end of a trace.

Example 14. We have that $(Q_0(b,a);\Phi_0;\emptyset) \xmapsto{\mathsf{tr}}_c (\emptyset;\Phi;\mathcal{S})$ 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 12.

We are now able to define our notion of (symbolic) trace equivalence.

Definition 8 (trace equivalence w.r.t. $\xmapsto{\mathsf{tr}}_c$). *Let $A = (\mathcal{P};\Phi)$ and $B = (\mathcal{Q};\Psi)$ be two simple processes. We have that $A \sqsubseteq_s B$ when, for every sequence* tr *such that $(\mathcal{P};\Phi;\emptyset) \xmapsto{\mathsf{tr}}_c (\mathcal{P}';\Phi';\mathcal{S}_A)$, for every $\theta \in \mathsf{Sol}(\Phi';\mathcal{S}_A)$, 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*
- *$\Phi\lambda^A_\theta \sim \Psi\lambda^B_\theta$ 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. $\xmapsto{\mathsf{tr}}_c$ *if $A \sqsubseteq_s B$ and $B \sqsubseteq_s A$.*

Example 15. We have that $(Q_0(b,a);\Phi_0) \not\sqsubseteq_s (Q_0(b,a');\Phi_0)$. Continuing Example 14, we have seen that $(Q_0(b,a);\Phi_0;\emptyset) \xmapsto{\mathsf{tr}}_c (\emptyset;\Phi;\mathcal{S})$, and $\theta \in \mathsf{Sol}(\Phi;\mathcal{S})$ (see Example 12). The only symbolic process that is reachable from $(Q_0(b,a');\Phi_0;\emptyset)$ using tr is $(\emptyset;\Phi';\mathcal{S}')$ with:

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

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

For processes without replication, the symbolic transition system is finite. 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 [7,14,12].

4.3 Soundness and Completeness

Using the usual approach, such as the one developed in [7,13], 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.

Finally, relying on these two results, we can establish that symbolic trace equivalence ($\approx_s$) exactly captures compressed trace equivalence ($\approx_c$).

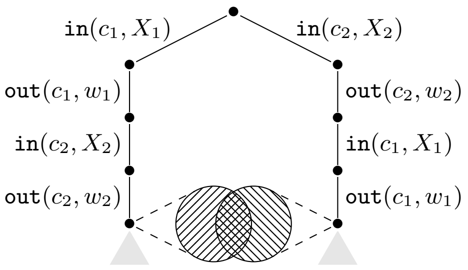
Theorem 2. *For any extended simple processes A and B, we have that:*

$$A \sqsubseteq_c B \iff A \sqsubseteq_s B.$$

5 Reduction Using Dependency Constraints

Unlike compression, which is based only on the input/output nature of actions, our second optimization takes into account the exchanged messages.

Let us first illustrate one simple instance of our optimization and how dependency constraints [18] may be used to incorporate it in symbolic semantics. Let $P_i = \mathtt{in}(c_i, x_i).\mathtt{out}(c_i, u_i).P'_i$ with $i \in \{1, 2\}$, and $\Phi_0 = \{w_0 \rhd n\}$ be a closed frame. We consider the simple process $A = (\{P_1, P_2\}; \Phi_0)$, and the two symbolic interleavings depicted below.



The two resulting symbolic processes are of the form $(\{P'_1, P'_2\}; \Phi; \mathcal{S}_i)$ where:

- $\Phi = \Phi_0 \uplus \{w_1 \rhd u_1, w_2 \rhd u_2\}$,
- $\mathcal{S}_1 = \left\{w_0 \vdash^?_{X_1} x_1;\ w_0, w_1 \vdash^?_{X_2} x_2\right\}$,
- $\mathcal{S}_2 = \left\{w_0 \vdash^?_{X_2} x_2;\ w_0, w_2 \vdash^?_{X_1} x_1\right\}$.

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 $\mathcal{C}_2$, whose purpose is to discard all solutions θ such that the message $x_1\lambda_\theta$ can be derived without using $w_2 \rhd u_2\lambda_\theta$. 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.

The idea of [18] 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 optimizing 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 optimization 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 diamond shape from the previous example.

There are at least two natural semantics for dependency constraints. The simplest semantics focuses on the second-order notion of recipe. In the above example, it would require that recipe $X_1\theta$ contains the variable w_2. That is weaker than a first-order semantics requiring that *any* recipe derivinig $x_1\lambda_\theta$ would involve w_2 since spurious dependencies may easily be introduced. Our ultimate goal in this section is to show that trace equivalence w.r.t. the first-order reduced semantics coincides with the regular symbolic semantics. However, we first establish this result for the second-order semantics, which is more easily analysed and provides a useful stepping stone.

5.1 Second-Order Reduced Semantics

We start by introducing *dependency constraints*, in a more general form than the one used above, and give them a second-order semantics.

Definition 9 (dependency constraint). *A* dependency constraint *is a constraint of the form* $\overrightarrow{X} \ltimes \overrightarrow{w}$ *where* $\overrightarrow{X}$ *is a vector of variables in* $\mathcal{X}^2$*, and* $\overrightarrow{w}$ *is a vector of handles, i.e. variables in* $\mathcal{W}$*.*

Given a substitution θ *with* $\mathrm{dom}(\theta) \subseteq \mathcal{X}^2$*, and* $X\theta \in \mathcal{T}(\mathcal{W})$ *for any* $X \in \mathrm{dom}(\theta)$*. We say that* θ *satisfies* $\overrightarrow{X} \ltimes \overrightarrow{w}$*, denoted* $\theta \models \overrightarrow{X} \ltimes \overrightarrow{w}$*, if either* $\overrightarrow{w} = \emptyset$ *or there exist* $X_i \in \overrightarrow{X}$ *and* $w_j \in \overrightarrow{w}$ *such that* $w_j \in fv^1(X_i\theta)$*.*

A constraint system with dependency constraints is called a *dependency constraint system*. We denote by $\mathcal{C}^\circ$ the regular constraint system obtained by removing all dependency constraints from $\mathcal{C}$. We only consider *well-formed* dependency constraint systems, that is those $\mathcal{C}$ such that $\mathcal{C}^\circ$ is well-formed. A solution of $\mathcal{C}$ is a substitution θ such that $\theta \in \mathsf{Sol}(\mathcal{C}^\circ)$ and $\theta \models \overrightarrow{X} \ltimes \overrightarrow{w}$ for each dependency constraint $\overrightarrow{X} \ltimes \overrightarrow{w} \in \mathcal{C}$. We denote this set $\mathsf{Sol}^2(\mathcal{C})$.

We shall now define 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 $\mathtt{in}(c, X_1) \cdot \ldots \cdot \mathtt{in}(c, X_\ell).\mathtt{out}(c, w_1) \cdot \ldots \cdot \mathtt{out}(c, w_k)$ 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 10 (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* $c_i \prec c \prec c_k$ *for all* $k < i \leq n$*, then we define*

$$\mathrm{dep}\,(\mathsf{tr}, c) = \{\ w \mid w \in \overrightarrow{w_i} \text{ with } k \leq i \leq n\}$$

Otherwise, we have that $\mathrm{dep}(\mathsf{tr}, c) = \emptyset$.

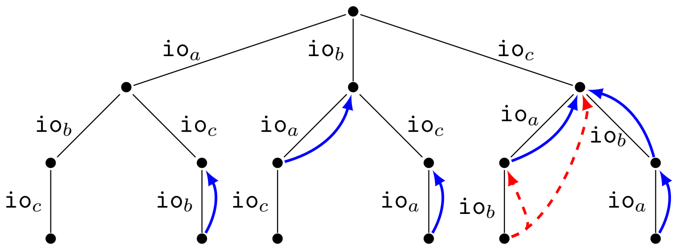
We obtain our reduced semantics by integrating those dependency constraints into the symbolic compressed semantics. We define $\mapsto_d$ as the least reflexive relation satisying the following rule:

$$\frac{(\mathcal{P};\Phi;\emptyset) \xmapsto{\mathsf{tr}}_d (\mathcal{P}';\Phi';\mathcal{S}') \quad (\mathcal{P}';\Phi';\mathcal{S}') \xmapsto{\mathtt{io}_c(\overrightarrow{X},\overrightarrow{w})}_c (\mathcal{P}'';\Phi'';\mathcal{S}'')}{(\mathcal{P};\Phi;\emptyset) \xmapsto{\mathsf{tr}\cdot\mathtt{io}_c(\overrightarrow{X},\overrightarrow{w})}_d (\mathcal{P}'';\Phi'';\mathcal{S}'' \cup \{\overrightarrow{X} \ltimes \mathrm{dep}(\mathsf{tr}, c)\})}$$

Given a proper trace, we define $\mathrm{Deps}(\mathsf{tr})$ to be the accumulation of the generated constraints as defined above for all prefixes of tr (where each proper block is considered as an atomic action). We may observe that:

– if $A \xmapsto{\mathsf{tr}}_d (\mathcal{P};\Phi;\mathcal{S})$ then $\mathcal{S} = \mathcal{S}^\circ \cup \mathrm{Deps}(\mathsf{tr})$ and $A \xmapsto{\mathsf{tr}}_c (\mathcal{P};\Phi;\mathcal{S}^\circ)$;
– if $A \xmapsto{\mathsf{tr}}_c (\mathcal{P};\Phi;\mathcal{S})$ then $\mathcal{S} = \mathcal{S}'^\circ$ and $A \xmapsto{\mathsf{tr}}_d (\mathcal{P};\Phi;\mathcal{S}')$.

Example 16. 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.

This reduced semantics gives rise to a notion of trace equivalence. It is defined as in Definition 8, relying on $\mapsto_d$ instead of $\mapsto_c$ and on Sol^2 instead of Sol. We denote it $\approx^2_d$, and the associated notion of inclusion is denoted $\sqsubseteq^2_d$

5.2 Soundness and Completeness

In order to establish that $\approx_s$ and $\approx^2_d$ coincide, we are going to study more carefully concrete traces made of (not necessarily proper) blocks. We denote by $\mathcal{B}$

the set of blocks $\mathtt{io}_c(\overrightarrow{M}, \overrightarrow{w})$ such that $c \in \mathcal{C}$, $M_i \in \mathcal{T}(\mathcal{W})$ 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 necessarily made of blocks, *i.e.,* it belongs to $\mathcal{B}^*$. Note that all traces from executions in the compressed semantics are concrete traces in this sense. We show that we can view $\mathcal{B}^*$ as a partially commutative monoid in a meaningful way. This allows us to lift a classic result in which we ground our reduced semantics.

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. We define similarly $\prec$ on symbolic traces.

Partially commutative monoid. We define an *independence relation* $\mathcal{I}_b$ over $\mathcal{B}$: we say that $\mathtt{io}_c(\overrightarrow{M}, \overrightarrow{w})\ \mathcal{I}_b\ \mathtt{io}_{c'}(\overrightarrow{M'}, \overrightarrow{w'})$ when $c \neq c'$, none of the variables of $\overrightarrow{w}'$ occurs in $\overrightarrow{M}$, and none of the variables of $\overrightarrow{w}$ occurs in $\overrightarrow{M}'$. Then we define $=_{\mathcal{I}_b}$ as the least congruence satisfying

$$\mathtt{io}_c(\overrightarrow{M}, \overrightarrow{w}) \cdot \mathtt{io}_{c'}(\overrightarrow{M'}, \overrightarrow{w'}) =_{\mathcal{I}_b} \mathtt{io}_{c'}(\overrightarrow{M'}, \overrightarrow{w'}) \cdot \mathtt{io}_c(\overrightarrow{M}, \overrightarrow{w})$$

for all $\mathtt{io}_c(\overrightarrow{M}, \overrightarrow{w})$ and $\mathtt{io}_{c'}(\overrightarrow{M'}, \overrightarrow{w'})$ with $\mathtt{io}_c(\overrightarrow{M}, \overrightarrow{w})\ \mathcal{I}_b\ \mathtt{io}_{c'}(\overrightarrow{M'}, \overrightarrow{w'})$. The set of concrete traces, quotiented by this equivalence relation, is the *partially commutative monoid* obtained from $\mathcal{I}_b$. Given a concrete trace tr, we denote by $\mathsf{min}(\mathsf{tr})$ the minimum for $\prec$ among all the traces that are equal to tr modulo $=_{\mathcal{I}_b}$.

First, we prove that the symbolic semantics is equally able to execute equivalent (w.r.t. $=_{\mathcal{I}_b}$) traces. Second we prove that the reduced semantics generates dependency constraints that are (only) satisfied by minimal traces.

Lemma 2. *Let* $(\mathcal{P}_0; \Phi_0; \emptyset) \xmapsto{\mathsf{tr}}_c (\mathcal{P}; \Phi; \mathcal{S})$ *with* tr *made of proper blocks, and* $\theta \in \mathsf{Sol}(\Phi; \mathcal{S})$. *For any concrete trace* $\mathsf{tr}_c =_{\mathcal{I}_b} \mathsf{tr}\theta$ *there exists a symbolic trace* tr' *such that* $\mathsf{tr}_c = \mathsf{tr}'\theta$, $(\mathcal{P}_0; \Phi_0; \emptyset) \xmapsto{\mathsf{tr}'}_c (\mathcal{P}; \Phi; \mathcal{S}')$ *and* $\theta \in \mathsf{Sol}(\Phi; \mathcal{S}')$.

Lemma 3. *Let* $A \xmapsto{\mathsf{tr}}_c (\mathcal{P}; \Phi; \mathcal{S})$ *and* $\theta \in \mathsf{Sol}(\Phi; \mathcal{S})$. *We have that* $\theta \models \mathrm{Deps}\,(\mathsf{tr})$ *if, and only if,* $\mathsf{tr}\theta = \mathsf{min}(\mathsf{tr}\theta)$.

Proof (Sketch). Let $A \xmapsto{\mathsf{tr}}_c (\mathcal{P}; \Phi; \mathcal{S})$ and $\theta \in \mathsf{Sol}(\Phi; \mathcal{S})$. We need a characterization of minimal traces. We exploit the following one, which is equivalent to the characterization of Anisimov and Knuth [3]:

> The trace t is minimal if, and only if, for all factors aub of t such that (1) $a, b \in \mathcal{B}$, $u \in \mathcal{B}^*$ and $d \prec b \prec a$ for all $d \in u$, we have (2) some $c \in au$ such that $c\,\mathcal{I}_b\,b$ does not hold.

We remark that condition (1) characterizes the factors of (symbolic) traces for which we generate a dependency constraint. Here, that constraint would be

$$\overrightarrow{X}_b \ltimes \cup_{d \in au} \overrightarrow{w}_d$$

where $\alpha \in \mathcal{B}$ is also written $\mathsf{io}_{c_\alpha}(\overrightarrow{X}_\alpha, \overrightarrow{w}_\alpha)$ to have an access to its components.

Then we note that (2) corresponds to the satisfaction of that dependency constraint in a concrete instance of the trace. $\square$

Finally, relying on these results, we can establish that trace equivalence ($\approx_d$) w.r.t. the reduced semantics exactly captures symbolic trace equivalence ($\approx_s$).

Theorem 3. *For any extended simple processes A and B, we have that:*

$$A \sqsubseteq_s B \iff A \sqsubseteq_d^2 B.$$

Proof (Sketch). Implication ($\Rightarrow$) is straightforward and only relies on the fact that dependency constraints generated by the reduced semantics only depend on the trace that is executed. The other direction ($\Leftarrow$) is more interesting. Here, we only outline the main idea, in the case of a trace made of proper blocks. We show that a concrete trace $\mathsf{tr}\theta$ which is not captured when considering $\mapsto_d$ (*i.e.,* a trace $\mathsf{tr}\theta$ that does not satisfy the generated dependency constraints) can be mapped to another trace, namely $\mathsf{min}(\mathsf{tr}\theta)$, which manipulates the same recipes/messages but where blocks are executed in a different order. Lemma 2 is used to obtain an execution of the minimal trace, and Lemma 3 ensures that dependency constraints are satisfied in that execution. Thus the minimal trace can also be executed by the other process. We go back to $\mathsf{tr}\theta$ using Lemma 2. $\square$

5.3 First-Order Reduced Semantics

We finally introduce the stronger, first-order semantics for dependency constraints, and we prove soundness and completeness for the corresponding equivalence property by building on the previous theorem.

Definition 11. *Let $\mathcal{C} = (\Phi; \mathcal{S})$ be a dependency constraint system. We define $\mathsf{Sol}^1(\mathcal{C})$ to be the set of substitutions $\theta \in \mathsf{Sol}(\mathcal{C}^\circ)$ such that, for each $\overrightarrow{X} \ltimes \overrightarrow{w}$ in $\mathcal{C}$ with non-empty $\overrightarrow{w}$ there is some $X_i \in \overrightarrow{X}$ such that for all recipes $M \in \mathcal{T}(D_\mathcal{C}(X))$ satisfying $M(\Phi\lambda_\theta) =_\mathsf{E} (X\theta)(\Phi\lambda_\theta)$, we have $fv^1(M) \cap \overrightarrow{w} \neq \emptyset$.*

We define the notion of trace equivalence accordingly, as it has been done at the end of Section 5.1, relying on Sol^1 instead of Sol^2. We denote it $\approx_d^1$, and the associated notion of inclusion is denoted $\sqsubseteq_d^1$.

Theorem 4. *For any extended simple processes A and B, we have that:*

$$A \sqsubseteq_d^2 B \iff A \sqsubseteq_d^1 B.$$

Proof (Sketch). ($\Rightarrow$) This implication is relatively easy to establish. It actually relies on the fact that $\mathsf{Sol}^1(\mathcal{C}) \subseteq \mathsf{Sol}^2(\mathcal{C})$ for any dependency constraint system $\mathcal{C}$. This allows us to use our hypothesis $A \sqsubseteq_d^2 B$. Then, in order to come back to our more constrainted first-order reduced semantics, we may notice that as soon as θ is a solution of $\mathcal{C}$ and $\mathcal{C}'$ (w.r.t. Sol^2) with static equivalence of their associated frames, we have that: $\theta \in \mathsf{Sol}^1(\mathcal{C})$ if, and only if, $\theta \in \mathsf{Sol}^1(\mathcal{C}')$. ($\Leftarrow$) In order to exploit our hypothesis $A \sqsubseteq_d^1 B$, given a trace

$$A \overset{\mathsf{tr}}{\mapsto}_d (\mathcal{P}; \Phi; \mathcal{S}) \text{ with } \theta \in \mathsf{Sol}^2(\Phi; \mathcal{S}),$$

we build tr' and θ' such that $A \overset{\mathsf{tr}'}{\mapsto}_d (\mathcal{P}; \Phi; \mathcal{S}')$ with "$\mathsf{tr} =_{\mathcal{I}_b} \mathsf{tr}'$", and $\theta' \in \mathsf{Sol}^1(\Phi; \mathcal{S}')$. Actually, we do this without changing the underlying first-order substitution, *i.e.,* $\lambda_\theta = \lambda_{\theta'}$. This is done by a sub-induction; iteratively modifying θ and tr. Whenever θ is not already a first-order solution, we slightly modify it. We obtain a new substitution θ' that is not a second-order solution anymore w.r.t. tr, and we use Lemmas 2 and 3 to obtain a new trace $\mathsf{tr}' \prec \mathsf{tr}$ for which θ' is a second-order solution. By induction hypothesis on tr' we obtain a first-order solution. We finally go back to the original trace tr, using an argument similar to the one in the first direction to handle static equivalence. □

Example 17. We illustrate the construction of tr', which is at the core of the above proof. Consider $A = (\{P_1, P_2, P_3\}; \Phi)$ where $P_i = \mathtt{in}(c_i, x_i).\mathtt{out}(c_i, n_i)$, and $\Phi_0 = \{w_0 \rhd n_0\}$, and $n_i \in \mathcal{N}$ for $0 \leq i \leq 3$. We assume that $c_1 \prec c_2 \prec c_3$, and we consider the situation where the nonces n_0 and n_2 (resp. n_1 and n_3) are the same.

Let $\mathsf{tr} = \mathtt{io}_{\mathsf{c}_3}(X_3, w_3).\mathtt{io}_{\mathsf{c}_2}(X_2, w_2).\mathtt{io}_{\mathsf{c}_1}(X_1, w_1)$ and $(\Phi; \mathcal{S})$ the dependency constraint system such that $A \overset{\mathsf{tr}}{\mapsto}_d (\emptyset; \Phi; \mathcal{S})$. We consider the substitution $\theta = \{X_3 \mapsto \mathsf{start},\ X_2 \mapsto w_3,\ X_1 \mapsto w_2\}$. We note that $\theta \in \mathsf{Sol}^2(\Phi; \mathcal{S})$ but we have that $\theta \notin \mathsf{Sol}^1(\Phi; \mathcal{S})$ due to the presence of $X_1 \ltimes w_2$ in $\mathcal{S}$. We could try to fix this problem by building a "better" solution θ' that yields the same first-order solution: $\theta' = \{X_3 \mapsto \mathsf{start},\ X_2 \mapsto w_3,\ X_1 \mapsto w_0\}$ is such a candidate. Applying Lemmas 2 and 3, we obtain a smaller symbolic trace:

$$\mathsf{tr}' = \mathtt{io}_{\mathsf{c}_1}(X_1, w_1) \cdot \mathtt{io}_{\mathsf{c}_3}(X_3, w_3) \cdot \mathtt{io}_{\mathsf{c}_2}(X_2, w_2).$$

Let $(\Phi; \mathcal{S}')$ be the constraint system obtained from the execution of tr'. We have that $\theta' \in \mathsf{Sol}^2(\Phi; \mathcal{S}')$ but again $\theta' \notin \mathsf{Sol}^1(\Phi; \mathcal{S}')$. This is due to the presence of $X_2 \ltimes w_3$ in $\mathcal{S}'$ — which was initially satisfied by θ in the first-order sense. With one more iteration of this transformation, we obtain a third candidate: $\theta'' = \{X_3 \mapsto \mathsf{start},\ X_2 \mapsto w_1,\ X_1 \mapsto w_0\}$ and

$$\mathsf{tr}'' = \mathtt{io}_{\mathsf{c}_1}(X_1, w_1) \cdot \mathtt{io}_{\mathsf{c}_2}(X_2, w_2) \cdot \mathtt{io}_{\mathsf{c}_3}(X_3, w_3).$$

The associated constraint system does not contain any dependency constraint, and thus θ'' is trivially a first-order solution.

5.4 Applications

We first describe two situations showing that our reduced semantics can yield an exponential benefit. Then, we illustrate the effect of our reduced semantics on our running example, *i.e.,* the private authentication protocol.

Consider first the simple process $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ where each P_i denotes the basic process $\mathtt{in}(c_i, x).\mathtt{if}\ x = \mathsf{ok}\ \mathtt{then}\ \mathtt{out}(c_i, n_i)$ with $n_i \in \mathcal{N}$. There are $(2n)!/2^n$ different traces of size $2n$ (*i.e.,* containing $2n$ visible actions) in the concrete semantics. This number is actually the same in the standard symbolic semantics. In the compressed semantics (as well as the symbolic compressed semantics) this number goes down to $n!$. Finally, in the reduced semantics, there is only one trace such that the resulting constraint system admits a solution. Assuming that $c_1 \prec \ldots \prec c_n$, that trace is simply:

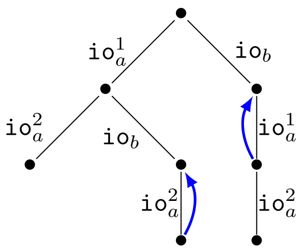
$$\mathsf{tr} = \mathtt{io}_{c_1}(\overrightarrow{X_1}, \overrightarrow{w_1}) \cdot \ldots \cdot \mathtt{io}_{c_n}(\overrightarrow{X_n}, \overrightarrow{w_n}).$$

Next, we consider the simple process $\mathcal{P} = \{P_1^n, P_2^n\}$ where $P_i^0 = 0$, and P_i^{n+1} denotes the basic process $\mathtt{in}(c_i, x_j).\mathtt{if}\ x_j = \mathsf{ok}\ \mathtt{then}\ \mathtt{out}(c_i, n_j).P_i^n$. We consider traces of size $4n$. In the concrete semantics, there are $\binom{4n}{2n}$ different traces, whereas the number of such traces is reduced to $\binom{2n}{n}$ in the compressed semantics. Again, there is only one trace left in the reduced semantics.

Going back to our running example (see Examples 2 and 7), we represent some symbolic traces obtained using our reduced semantics. We consider:

$$(\{P_{\mathsf{init}}, Q_0(skb, \mathsf{pk}(ska))\}; \Phi_0; \emptyset)$$

and we assume that $c_A \prec c_B$. We consider all symbolic traces obtained without considering the ELSE rule.



Those executions are represented in the diagram on the left, where

– $\mathtt{io}_a^1$ to denote $\mathtt{io}_{c_A}(X_a^1, w_a)$,
– $\mathtt{io}_a^2$ to denote $\mathtt{io}_{c_A}(X_a^2, \emptyset)$, and
– $\mathtt{io}_b$ to denote $\mathtt{io}_{c_B}(X_b, w_b)$.

The block $\mathtt{io}_a^2$ is an improper block since it only contains an input action. First, we may note that many interleavings are not taken into account thanks to compression. Now, consider the symbolic trace $\mathtt{io}_a^1 \cdot \mathtt{io}_b \cdot \mathtt{io}_a^2$. A dependency constraint of the form $X_a^2 \ltimes w_b$ is generated. Thus, a concrete trace that satisfies this dependency constraint must use the output of the role $Q_0(b, a)$ to build the second input of the role P_{init}.

Second, consider the rightmost branch. A dependency constraint of the form $X_a^1 \ltimes w_b$ is generated, and since X_a^1 has to be instantiated by a recipe that gives the public constant start (due of the constraint $x_a^1 =^? \mathsf{start}$ present in the system), the reduced semantics makes it possible to prune all executions starting with $\mathtt{io}_b \cdot \mathtt{io}_a^1$.

6 Conclusion

We have proposed two refinements of the symbolic semantics for 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. In theory, this yields a potentially exponential algorithmic optimization.

In order to validate our approach, an experimental implementation has been developed[1]. This tool is based on SPEC [21] (which does not support else branches) and implements our modified semantics as well as an adapted constraint resolution procedure that takes (first-order) dependency constraints into account. The latter procedure is quite preliminary and far from optimal. Yet,

[1] Available at `<http://perso.ens-lyon.fr/lucca.hirschi/spec_en.html>`.

the modified checker already shows significant improvements over the original version on various benchmarks ([16], Figure 9).

We are considering several directions for future work. Constraint solving procedures should be studied in depth: we may optimize the one we already developed [16] and we are also interested in studying the problem in other frameworks, *e.g.,* [11]. We also believe that stronger reductions can be achieved: for instance, exploiting symmetries should be very useful for dealing with multiple sessions.

References

1. Abadi, M., Fournet, C.: Mobile values, new names, and secure communication. In: Proc. 28th Symposium on Principles of Programming Languages (POPL 2001), pp. 104–115. ACM Press (2001)
2. Abadi, M., Fournet, C.: Private authentication. Theoretical Computer Science 322(3), 427–476 (2004)
3. Anisimov, A., Knuth, D.: Inhomogeneous sorting. International Journal of Computer & Information Sciences 8(4), 255–260 (1979)
4. Arapinis, M., Chothia, T., Ritter, E., Ryan, M.: Analysing unlinkability and anonymity using the applied pi calculus. In: Proc. 23rd Computer Security Foundations Symposium (CSF 2010), pp. 107–121. IEEE Comp. Soc. Press (2010)
5. Armando, A., et al.: The AVISPA tool for the automated validation of internet security protocols and applications. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 281–285. Springer, Heidelberg (2005)
6. Baelde, D., Delaune, S., Hirschi, L.: A reduced semantics for deciding trace equivalence using constraint systems. ArXiv e-prints (January 2014)
7. Baudet, M.: Deciding security of protocols against off-line guessing attacks. In: Proc. 12th Conference on Computer and Communications Security. ACM (2005)
8. Blanchet, B.: An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In: Proc. 14th Computer Security Foundations Workshop (CSFW 2001), pp. 82–96. IEEE Comp. Soc. Press (2001)
9. Blanchet, B., Abadi, M., Fournet, C.: Automated verification of selected equivalences for security protocols. Journal of Logic and Algebraic Programming (2008)
10. Chadha, R., Ciobâcă, Ş., Kremer, S.: Automated verification of equivalence properties of cryptographic protocols. In: Seidl, H. (ed.) ESOP 2012. LNCS, vol. 7211, pp. 108–127. Springer, Heidelberg (2012)
11. Cheval, V.: APTE (2011), `http://projects.lsv.ens-cachan.fr/APTE/`
12. Cheval, V., Comon-Lundh, H., Delaune, S.: Trace equivalence decision: Negative tests and non-determinism. In: Proc. 18th Conference on Computer and Communications Security (CCS 2011). ACM Press (2011)
13. Cheval, V., Cortier, V., Delaune, S.: Deciding equivalence-based properties using constraint solving. Theoretical Computer Science 492, 1–39 (2013)
14. Chevalier, Y., Rusinowitch, M.: Decidability of symbolic equivalence of derivations. Journal of Automated Reasoning 48(2) (2012)
15. Cremers, C.J.F.: The Scyther Tool: Verification, falsification, and fnalysis of security protocols. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 414–418. Springer, Heidelberg (2008)
16. Hirschi, L.: Réduction d'entrelacements pour l'équivalence de traces. RR LSV-13-13, Laboratoire Spécification et Vérification, ENS Cachan, France (September 2013)

17. Millen, J., Shmatikov, V.: Constraint solving for bounded-process cryptographic protocol analysis. In: Proc. 8th ACM Conference on Computer and Communications Security (CCS 2001). ACM Press (2001)
18. Mödersheim, S., Viganò, L., Basin, D.A.: Constraint differentiation: Search-space reduction for the constraint-based analysis of security protocols. Journal of Computer Security 18(4), 575–618 (2010)
19. Peled, D.: Ten years of partial order reduction. In: Vardi, M.Y. (ed.) CAV 1998. LNCS, vol. 1427, pp. 7–28. Springer, Heidelberg (1998)
20. Rusinowitch, M., Turuani, M.: Protocol insecurity with finite number of sessions is NP-complete. In: Proc. 14th Computer Security Foundations Workshop (CSFW 2001), pp. 174–190. IEEE Comp. Soc. Press (2001)
21. Tiu, A.: Spec (2010), `http://users.cecs.anu.edu.au/~tiu/spec/`
22. Tiu, A., Dawson, J.E.: Automating open bisimulation checking for the spi calculus. In: Proc. 23rd IEEE Computer Security Foundations Symposium (CSF 2010), pp. 307–321. IEEE Computer Society Press (2010)