SAT-Equiv: an efficient tool for equivalence properties
Véronique Cortier, Antoine Dallon, Stéphanie Delaune

To cite this version:
Véronique Cortier, Antoine Dallon, Stéphanie Delaune. SAT-Equiv: an efficient tool for equivalence properties. [Research Report] LSV, ENS Cachan, CNRS, INRIA, Université Paris-Saclay, Cachan (France); IRISA, Inria Rennes; LORIA - Université de Lorraine; CNRS. 2017. hal-01529966

HAL Id: hal-01529966
https://hal.archives-ouvertes.fr/hal-01529966
Submitted on 31 May 2017

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

Véronique Cortier*, Antoine Dallon*,†, Stéphanie Delaune†
* CNRS, LORIA, France
† LSV & ENS Paris-Saclay, Université Paris-Saclay, France
‡ CNRS, IRISA, France

Abstract—Automatic tools based on symbolic models have been successful in analyzing security protocols. Such tools are particularly adapted for trace properties (e.g. secrecy or authentication), while they often fail to analyze equivalence properties.

Equivalence properties can express a variety of security properties, including in particular privacy properties (vote privacy, anonymity, untraceability). Several decision procedures have already been proposed but the resulting tools are rather inefficient.

In this paper, we propose a novel algorithm, based on graph planning and SAT-solving, which significantly improves the efficiency of the analysis of equivalence properties. The resulting implementation, SAT-Equiv, can analyze several sessions where most tools have to stop after one or two sessions.

I. INTRODUCTION

Formal methods have produced several successful tools for the automatic analysis of security protocols. Examples of such tools are ProVerif [1], Avantssar [2], Maude-NPA [3], Scyther [4], or Tamarin [5]. They have been applied to many protocols of the literature including e.g. Kerberos and TLS. However, one type of properties still resists to these tools, namely privacy properties. Privacy properties include ballot privacy (no one should know how I voted), privacy (no one should know I am here), or unlinkability (no one should be able to relate two of my transactions). Such properties are typically expressed as equivalences: an attacker should not be able to distinguish a session from Alice from one from Bob.

Equivalence properties are harder to analyze than the more standard authentication or confidentiality properties (expressed as trace properties). Among the tools mentioned earlier, only ProVerif, Maude-NPA and Tamarin may handle equivalences. Maude-NPA [6] often does not terminate when used for equivalence properties. Since checking equivalence properties for an unbounded number of sessions is undecidable [7], ProVerif may of course also fail. This is in particular the case when the order of the protocol rules matters or when some step may be executed at most once.

The alternative is to decide equivalence, for a bounded number of sessions. Several procedures have been proposed [8]–[10], often with a companion tool: Akiss [10], Spec [9], Apte [11]. Unfortunately, these tools have a very limited practical impact because they scale badly. Analyzing one session typically requires several seconds and the analysis of two sessions is often unreachable, although the tools Apte and Akiss have recently improved their efficiency through the use of Partial Order Reduction (POR) techniques [12], [13]. It is interesting to note that considering one or two sessions is not sufficient to explore all standard attack scenarios (where each participant may engage a session with an honest or a dishonest agent and may be involved in any role). For example, in the case of a three-party protocol, with a trusted server, 6 sessions have to be considered to cover all possible scenarios with two honest agents A, B and a dishonest one C (A talking to B, A talking to C, and C talking to B, and three additional sessions where the role of the agents A and B are swapped). Assuming that dishonest roles do not need to be modelled, this leads to a scenario with 14 roles in parallel. In practice, an attack does not require 14 sessions. 3-4 sessions are typically sufficient. However, it is impossible to predict which scenario is required for the attack. Moreover, since the problem of deciding equivalence is actually undecidable, an attack may require an arbitrary number of sessions. Therefore, the more sessions we can check, the more confidence we obtain.

Our contribution. In this paper, we propose a different procedure for deciding equivalence. Instead of designing a crafted algorithm for equivalence, we use more general verification techniques, namely Graph planning [14], [15] and SAT-solving. The idea of using Graph planning and SAT-solvers for analyzing protocols has already been explored in [16], yielding the tool SATMC [17] for trace properties. Moving from trace to equivalence properties is far from being straightforward as exemplified by the research effort on equivalence these past 10 years (see e.g. [18] for a survey).

Let us first sketch how SATMC works. The tool focuses at secrecy and encodes accessibility of a (secret) term into a SAT formula. For efficiency reasons, the main step of SATMC actually consists in applying Graph planning techniques in order to compute an over-approximation of reachable messages. If no secret has been found, the protocol is deemed secure. Otherwise, actual accessibility of the potentially leaked secret is encoded into a SAT formula.

In order to benefit from Graph planning and SAT-solvers, the size of messages has to be bounded and this bound needs to be practical. In [16], [17], the authors simply assume protocols to be given with a (finite) format for the messages. Here, we do not bound a priori the format of the messages. Instead, we rely on a recent result [19] that shows that if there is an attack, that is a witness of non equivalence between two
protocols \( P \) and \( Q \), then there is a “small” attack, where messages comply to a certain format (induced by a type). This result holds for deterministic protocols that use symmetric keys and pairing. Note that this result only controls the format of the messages exchanged in \( P \), not in \( Q \) (or conversely). The fact that the messages in \( Q \) are a priori unbounded forbids any direct encoding of (non) equivalence into a SAT formula. Planning graphs are particularly helpful here: while computing an over-approximation of the messages reachable in \( P \), we simultaneously obtained an over-approximation of the messages that need to be considered in \( Q \) for checking equivalence w.r.t. \( P \). This requires of course to characterize (non) equivalence as a reachability property, which is made possible thanks to the protocols’ determinism.

In order to further reduce the traces than need to be explored, we show that we can restrict ourselves to an attacker that only decompose messages (and do not compose them), provided that protocols are flattened, that is all meaningful composition steps are pre-computed in advance. This flattening technique has been used in [16] (although we are not aware of any proof of correction). We formally prove this technique to be sound, in the more general case of equivalence properties. Handling equivalence is non trivial since it is not sufficient to preserve the set of messages that can be computed, it is also necessary to preserve cases of failure on both processes. Moreover, we had use one more ingredient to obtain an efficient bound. We significantly reduce the number of constants that need to be considered to find an attack. Namely, we show that only two constants are necessary, which is a result of independent interest.

**Implementation.** We have implemented our algorithm and our first experimentations demonstrate the good performance of our tool. For most protocols, we can easily analyse several sessions while the three other tools (Akiss, Spec, Apte) typically fail for more than one session, with the exception of the variant Apte-por [13], which can handle several sessions, in some cases. All files related to the tool implementation and case studies are available at [20].

## II. Model for security protocols

A common framework for modelling security protocols are process algebra like the applied pi-calculus [21]. We consider here a variant of the calculus provided in [19] in order to benefit from its main result, which guarantees a “small attack” property: when there is an attack, there is a well-typed attack.

### A. Syntax

**Term algebra:** As usual, messages are modelled by terms. We consider an infinite set of names \( \mathcal{N} \), an infinite set of constants \( \Sigma_0 \), and two distinct sets of variables \( \mathcal{X} \) and \( \mathcal{W} \). Names are typically used to represent keys or nonces. Variables in \( \mathcal{X} \) refer to unknown parts of messages expected by participants while variables in \( \mathcal{W} \) are used to store messages learnt by the attacker. We consider the following sets of function symbols:

\[
\Sigma_c = \{ \text{enc}, \langle \rangle \} \quad \Sigma_d = \{ \text{dec}, \text{proj}_1, \text{proj}_2 \} \quad \Sigma_{std} = \Sigma_c \cup \Sigma_d
\]

The symbol enc and dec both of arity 2 represent encryption and decryption. Concatenation of messages is modelled through the symbol \( \langle \rangle \) of arity 2, with projection functions \( \text{proj}_1 \) and \( \text{proj}_2 \) of arity 1. We distinguish between constructor symbols in \( \Sigma_c \) and destructor symbols in \( \Sigma_d \).

We consider several sets of terms. Given a set of A of atoms (i.e. names, variables, and constants), and a signature \( \mathcal{F} \in \{ \Sigma_c, \Sigma_d, \Sigma_{std} \} \), we denote by \( \mathcal{T}(\mathcal{F}, A) \) the set of terms built from \( \mathcal{F} \) and A. Constructors terms with atomic encryptions are represented by the set \( \mathcal{T}_0(\Sigma_c, A) \), which is the subset of \( \mathcal{T}(\Sigma_c, A) \) such that any subterm \( \text{enc}(m, k) \) of a term in \( \mathcal{T}_0(\Sigma_c, A) \) is such that \( k \in A \). Given \( \Sigma \subseteq \Sigma_0 \), we denote by \( \mathcal{M}_\Sigma \) the set \( \mathcal{T}_0(\Sigma_c, \Sigma \cup \mathcal{N}) \), i.e. the set of messages built using constants in \( \Sigma \). The positions of a term are defined as usual. We denote \( \text{vars}(u) \) the set of variables that occur in \( u \). The application of a substitution \( \sigma \) to a term \( u \) is written \( u\sigma \), and we denote \( \text{dom}(\sigma) \) its domain, and \( \text{img}(\sigma) \) its image. Two terms \( u_1 \) and \( u_2 \) are unifiable when there exists \( \sigma \) such that \( u_1 \sigma = u_2 \sigma \). In this case, we denote \( \text{mg} \) their most general unifier. The composition of two substitutions \( \sigma_1 \) and \( \sigma_2 \) is denoted \( \sigma_1 \circ \sigma_2 \).

**Example 1:** Let \( k_{ab} \) and \( k_{as} \) be two names in \( \mathcal{N} \), and \( a \) be a constant from \( \Sigma_0 \). We have that \( t = \text{enc}(k_{ab}, a) \) is a message from \( \mathcal{M}_{\Sigma_0} \), whereas \( \text{enc}(a, \langle k_{as}, k_{ab} \rangle) \) is not (due to the presence of a compound term in key position).

An attacker can build any term by applying function symbols. Its computation is formally modelled by terms, called recipes. Given \( \Sigma \subseteq \Sigma_0 \), we denote \( \mathcal{R}_\Sigma \) the set \( \mathcal{T}(\Sigma_{std}, \Sigma \cup \mathcal{W}) \), i.e. the set of recipes built using constants in \( \Sigma \). Note that a recipe does not contain names, since, intuitively, names are initially secret.

**Example 2:** Assume that the attacker has first intercepted the message \( t \) (stored in \( w_1 \)), and then the key \( k_{as} \) (stored in \( w_2 \)). The term \( R = \text{proj}_1(\text{dec}(w_1, w_2)) \) is a recipe that represents a computation that can be performed by the attacker. Indeed, he can decrypt the first message with the second one, and then apply a projection operator.

The decryption of an encryption with the right key yields the plaintext. Similarly, the left (or right) projection of a concatenation yields the left (or right) component. These properties are reflected in the three following convergent rewrite rules:

\[
\text{dec}(\text{enc}(x, y), y) \rightarrow x, \text{and} \ \text{proj}_i((x_1, x_2)) \rightarrow x_i \ i \in \{1, 2\}.
\]

A term \( u \) can be rewritten in \( v \) if there is a position \( p \) in \( u \), and a rewriting rule \( g(t_1, \ldots, t_n) \rightarrow t \) such that \( u|_p = g(t_1, \ldots, t_n) \theta \) for some substitution \( \theta \). Moreover, we assume that \( t_1 \theta, \ldots, t_n \theta \) as well as \( \theta \) are messages. This assumption slightly differs from [19]. Here, whenever an inner decryption/projection fails then the overall evaluation fails. Intuitively, we model eager evaluation while [19] models lazy evaluation. Our rewriting system is convergent, and we denote \( u \downarrow \) the normal form of a given term \( u \).
\[
\text{Example 3:} \text{ Let } t \text{ be the term given in Example 1, we have that } \text{proj}_1(\text{dec}(t, k_{as})) = k_{ab}. \text{ Indeed, we have that: }
\]
\[
\text{proj}_1(\text{dec}(t, k_{as})) = \text{proj}_1(\text{dec}(\text{enc}(\langle k_{ab}, a \rangle, k_{as}), k_{as}))
\]
\[
\rightarrow \text{proj}_1(\langle k_{ab}, a \rangle)
\]
\[
\rightarrow k_{ab}
\]

**Process algebra:** We only consider public channels and we assume that each process communicates on a dedicated channel. In practice, an attacker can typically distinguish between protocol participants thanks to their IP address and even between protocol sessions thanks to session identifiers. Technically, this assumption avoids non determinism. Formally, we assume an infinite set \(Ch\) of channels and we consider the fragment of simple processes without replication built on basic processes as defined e.g. in [22]. A basic process represents a party in a protocol, which may sequentially perform actions such as waiting for a message of a certain form, and outputing a message. Then, a simple process is a parallel composition of such basic processes playing on distinct channels.

**Definition 1:** The set of basic processes on \(c \in Ch\) is defined as follows (with \(u_1, u_2 \in T(\Sigma_c, \Sigma_0 \cup N \cup \chi)\)):

\[
P, Q := 0 \mid \text{in}(c, u_1).P \mid \text{out}(c, u_2).P
\]

A simple process \(P = \{P_1, \ldots, P_n\}\) is a multiset of basic processes \(P_i\) on pairwise distinct channels \(c_i\).

The process 0 does nothing. The process “\(\text{in}(c, u_1).P\)” expects a message \(m\) of the form \(u_1\) on channel \(c\) and then behaves like \(P\sigma\) where \(\sigma\) is a substitution such that \(m = u_2\sigma\) is a message. The process “\(\text{out}(c, u_2).P\)” emits \(u_2\) on channel \(c\), and then behaves like \(P\). We assume that names are implicitly freshly generated, and therefore we do need a specific action to model name generation. The construction “new” is important in the presence of replication but we do not consider replication here. For the sake of clarity, we may omit the null process. We write \(\text{fs}(P)\) for the set of free variables that occur in \(P\), i.e. the set of variables that are not in the scope of an input.

**Definition 2:** A protocol is a simple process \(P\) that is ground, i.e. \(\text{fs}(P) = \emptyset\).

**Example 4:** The Denning Sacco protocol [23] (without timestamps) is a key distribution protocol using symmetric encryption and a trusted server. Informally, we have:

1. \(A \rightarrow S : \ A, B\)
2. \(S \rightarrow A : \ {B, K_{ab}, \{K_{ab}, A\}_{Ky}}\)
3. \(A \rightarrow B : \ {K_{ab}, A}\)

where \(\{m\}_k\) denotes the symmetric encryption of a message \(m\) with key \(k\). The agents \(A\) and \(B\) aim at authenticating each other and establishing a session key \(K_{ab}\) through a trusted server \(S\). The key \(K_{as}\) (resp. \(K_{bs}\)) is a long term key shared between \(A\) and \(S\) (resp. \(B\) and \(S\)).

To model the Denning Sacco protocol, we introduce several atomic data: \(k_{as}, k_{bs}, k_{ab}\) are names, \(a\) and \(b\) are constants from \(\Sigma_0\), and \(c_1, c_2, c_3\) are channel names from \(Ch\). Each role is modelled by a basic process that is described below. Below, we denote by \(\langle x_1, x_2, x_3 \rangle\) the term \(\langle x_1, \langle x_2, x_3 \rangle \rangle\).

- \(P_A = \) \text{out}(c_1, \langle a, b \rangle).
- \text{in}(c_1, \text{enc}(b, x_{AB}, x_B), k_{as})).
- \text{out}(c_1, x_B)
- \(P_S = \) \text{in}(c_2, \langle a, b \rangle).
- \text{out}(c_2, \text{enc}(b, k_{ab}, \text{enc}(\langle k_{ab}, a \rangle, k_{bs})), k_{as}))
- \(P_B = \) \text{in}(c_3, \text{enc}(\langle y_{AB}, a \rangle, k_{bs})))

The protocol is then modelled by the simple ground process \(P_{DS} = \{P_A, P_S, P_B\}\). In order to model several sessions of the same protocol, we simply have to consider several instances of the basic processes \(P_A, P_S,\) and \(P_B\). We will use different channel names to get a simple process, different names to model fresh names, and we will rename variables to avoid clashes. Two sessions of the Denning-Sacco protocol (between honest participants) are therefore modelled by:

\[
P'_{DS} = \{P'_A, P'_S, P'_B\}
\]

where \(P'_A, P'_B,\) and \(P'_S\) are given below:

- \(P'_A = \) \text{out}(c_4, \langle a, b \rangle).
- \text{in}(c_4, \text{enc}(b, x'_{AB}, x'_B), k_{as})).
- \text{out}(c_4, x'_B)
- \(P'_S = \) \text{in}(c_5, \langle a, b \rangle).
- \text{out}(c_5, \text{enc}(b, k'_{ab}, \text{enc}(\langle k'_{ab}, a \rangle, k_{bs})), k_{as}))
- \(P'_B = \) \text{in}(c_6, \text{enc}(\langle y'_{AB}, a \rangle, k_{bs})))

**B. Semantics**

The operational semantics of a process is defined using a relation over configurations, i.e. triples \((P; \phi; \sigma)\) where:

- \(P\) is a multiset of processes with \(\text{fs}(P) \subseteq \text{dom}(\sigma)\);
- \(\phi\) is a frame, i.e. a substitution of the form
  \[
  \{w_1 \triangleright m_1, \ldots, w_n \triangleright m_n\}
  \]
  where \(w_1, \ldots, w_n \in W,\) and \(m_1, \ldots, m_n \in M_{\Sigma_0};\)
- \(\sigma\) is a substitution such that \(\text{dom}(\sigma) \subseteq \chi,\) and \(\text{img}(\sigma) \subseteq M_{\Sigma_0}.\)

We often write \(P\) instead of \((P; \emptyset; \emptyset),\) and \(P \cup \emptyset\) instead of \((P)\cup \emptyset\). The terms in \(\phi\) represent the messages that are sent out and therefore known by the attacker whereas the substitution \(\sigma\) is used to store parts of the messages received so far. The operational semantics of a process is induced by the relation \(\xrightarrow{\sigma}\) over configurations defined below:

**In:**

\[
(\text{in}(c, u).P \cup \emptyset; \phi; \sigma) \xrightarrow{\text{in}(c,R)} (P \cup \emptyset; \phi; \sigma \uplus \sigma_0)
\]

where \(R \in R_{\Sigma_0}\) such that \(R\sigma_0 \in M_{\Sigma_0},\) and \(R\phi_1 = (u\sigma)\sigma_0\) for \(\sigma\) with \(\text{dom}(\sigma_0) = \text{vars}(u\sigma).\)

**Out:**

\[
(\text{out}(c, u).P \cup \emptyset; \phi; \sigma) \xrightarrow{\text{out}(c,w)} (P \cup \emptyset; \phi \uplus \{w \triangleright u\sigma\}; \sigma)
\]

with \(w\) a fresh variable from \(W,\) and \(u\sigma \in M_{\Sigma_0}.

A process may input any term that an attacker can built from publicly available terms and symbols (rule In). The second rule corresponds to the output of a term: the corresponding term is added to the frame of the current configuration, which
means that the attacker has now access to it. Note that the term is outputted provided that it is a message. In case the evaluation of the term yields an encryption with a non atomic key, the evaluation fails and there is no output. We do not need to model internal communications since we assume public channels: all communications are controlled by the attacker.

The relation $\rightarrow$ between configurations (where $\rightarrow$ is a possibly empty sequence of actions) is defined in the usual way. Given $\Sigma \subseteq \Sigma_0$, and a protocol $\mathcal{P}$ we define its set of traces w.r.t. $\Sigma$ as follows:

$\text{trace}_\Sigma(\mathcal{P}) = \{(tr, \phi) \mid (P; \psi; \emptyset) \rightarrow (P'; \phi; \sigma) \}
\text{for some configuration } (P'; \phi; \sigma)\text{ and any recipe occurring in } tr \text{ is in } \mathcal{R}_\Sigma\}.$

Note that, for any $(tr, \phi) \in \text{trace}_\Sigma(\mathcal{P})$, we have that $tr \phi_\downarrow$ only contains messages in $M_{\Sigma_0}$.

**Example 5:** Consider the following sequence $tr$:

$tr = \text{out}(c_1, w_1).\text{in}(c_2, w_2).\text{out}(c_2, w_2).
\text{in}(c_1, w_2).\text{out}(c_1, w_3).\text{in}(c_3, w_3)$

This sequence $tr$ allows one to reach the frame:

$\phi = \{w_1 \triangleright (a, b), w_2 \triangleright \text{enc}(b, kab, \text{enc}(k_{ab}, a), k_{ba}), k_{as}\},
w_3 \triangleright \text{enc}(k_{ab}, a), k_{ba}\}$.

We have that $(tr, \phi) \in \text{trace}_\Sigma(\mathcal{P}_{DS})$. This trace corresponds to a normal execution of the protocol.

**C. Trace equivalence**

Trace equivalence can be used to formalise many interesting security properties, in particular privacy-type properties. We assume keys to be atomic and encryption to fail for non atomic keys. We define trace equivalence accordingly, by letting the attacker observe when an encryption fails. We first define equivalence on sequences of messages.

**Definition 3:** A frame $\phi_1$ is statically included w.r.t. $\Sigma \subseteq \Sigma_0$ in a frame $\phi_2$, denoted $\phi_1 \subseteq_\Sigma \phi_2$, when we have that $\text{dom}(\phi_1) = \text{dom}(\phi_2)$, and:

- for any $R \in \mathcal{R}_\Sigma, R \phi_1 \downarrow \in M_{\Sigma_0}$ implies that $R \phi_2 \downarrow \in M_{\Sigma_0}$;
- for any $R_1, R_2 \in \mathcal{R}_\Sigma$ with $R_1 \phi_1 \downarrow, R_2 \phi_1 \downarrow \in M_{\Sigma_0}, R_1 \phi_1 \downarrow = R_2 \phi_1 \downarrow$ implies that $R_1 \phi_2 \downarrow = R_2 \phi_2 \downarrow$.

They are in static equivalence w.r.t. $\Sigma$, denoted $\phi_1 \sim_\Sigma \phi_2$, when $\phi_1 \subseteq_\Sigma \phi_2$, and $\phi_2 \subseteq_\Sigma \phi_1$ (both w.r.t. $\Sigma$).

**Example 6:** Consider $\phi_1 = \phi \cup \{w_4 \triangleright \text{enc}(m_1, k_{ab})\}$ and $\phi_2 = \phi \cup \{w_4 \triangleright \text{enc}(m_2, k)\}$ where $\phi$ has been introduced in Example 5. The terms $m_1, m_2$ are public constants from $\Sigma_0$, and $k$ is a name from $\mathcal{N}$. We have that the two frames $\phi_1$ and $\phi_2$ are statically equivalent (w.r.t. any $\Sigma$). Intuitively, at the end of a normal execution between honest participants, an attacker can not distinguish whether the key used to encrypt a message (here the constants $m_1$ and $m_2$) is the session key that has been established or a fresh key $k$.

In contrast, the frames $\phi_1' = \phi \cup \{w_5 \triangleright k_{ab}\}$ and $\phi_2' = \phi \cup \{w_5 \triangleright k_{ab}\}$ are not in static equivalence. Actually $\phi_1'$ is not statically included in $\phi_2'$. Indeed, an attacker can observe that the 4th message of $\phi_1$ can be decrypted by the 5th message, which is not the case in $\phi_2'$. Formally, considering $R = \text{dec}(w_4, w_5)$, we have $R \phi_1' \downarrow \in M_{\Sigma_0}$, while $R \phi_2' \downarrow \notin M_{\Sigma_0}$.

Then, we lift this notion of equivalence from frames to configurations.

**Definition 4:** Let $\Sigma \subseteq \Sigma_0$. A protocol $\mathcal{P}$ is trace included w.r.t. $\Sigma$ in a protocol $\mathcal{Q}$, written $\mathcal{P} \subseteq_\Sigma \mathcal{Q}$, if for every $(tr, \phi) \in \text{trace}_\Sigma(\mathcal{P})$, there exists $(tr', \psi) \in \text{trace}_\Sigma(\mathcal{Q})$ such that $tr = tr'$ and $\phi \subseteq_\Sigma \psi$ w.r.t. $\Sigma$. The protocols $\mathcal{P}$ and $\mathcal{Q}$ are trace equivalent w.r.t. $\Sigma$, written $\mathcal{P} \approx_\Sigma \mathcal{Q}$, if $\mathcal{P} \subseteq_\Sigma \mathcal{Q}$ and $\mathcal{Q} \subseteq_\Sigma \mathcal{P}$ (both w.r.t. $\Sigma$).

This notion of equivalence (even when $\Sigma = \Sigma_0$) does not coincide in general with the usual notion of trace equivalence as defined e.g. in [22]. It is actually coarser since we simply require the resulting frames to be in static inclusion ($\phi \subseteq_\Sigma \psi$) instead of static equivalence ($\phi \sim_\Sigma \psi$). However, these two notions actually coincide (see [10]) for the class of simple processes that we consider in this paper.

Assume given two protocols $\mathcal{P}$ and $\mathcal{Q}$ such that $\mathcal{P} \not\subseteq_\Sigma \mathcal{Q}$ w.r.t. $\Sigma$. A witness of this non-inclusion is a trace $tr$ w.r.t. $\Sigma$ for which there exists $\phi$ such that $(tr, \phi) \in \text{trace}_\Sigma(\mathcal{P})$ and:

- either there is no $\psi$ such that $(tr, \psi) \in \text{trace}_\Sigma(\mathcal{Q})$;
- or such $\psi$ exists and $\phi \not\subseteq_\Sigma \psi$ w.r.t. $\Sigma$.

Note that for a simple process, once the sequence $tr$ is fixed, the resulting frame reachable through $tr$ is uniquely defined (when it exists) since simple processes are deterministic.

**Example 7:** The protocol $\mathcal{P}_{DS}'$ presented in Example 4 models two sessions of the Denning Sacco protocol. Assume now that we wish to check strong secrecy of the exchanged key, as received by the agent $A$. This can be expressed by checking whether $\mathcal{P}_{DS}' \approx_\Sigma \mathcal{P}_{DS}'$ where:

- $\mathcal{P}_{DS}'$ is as $\mathcal{P}_{DS}$ but we add “$\text{out}(c_4, \text{enc}(m_1, x_{AB}))$” at the end of the process $P_A$, and “$\text{out}(c_4, \text{enc}(m_1, x_{AB}))$” at the end of $P_A$;
- $\mathcal{P}_{DS}'$ is as $\mathcal{P}_{DS}$ but we add the instruction “$\text{out}(c_4, \text{enc}(m_2, k))$” at the end of $P_A$, and “$\text{out}(c_4, \text{enc}(m_2, k'))$” at the end of $P_A$.

The terms $m_1$ and $m_2$ are two public constants from $\Sigma_0$ whereas $k$ and $k'$ are names from $\mathcal{N}$.

While the key received by $A$ cannot be learnt by an attacker, strong secrecy of this key is not guaranteed. Indeed, due to the lack of freshness, the same key can be sent several times to $A$, and this can be observed by an attacker. Formally, the attack is as follows. Consider the sequence

$tr = tr \cdot \text{out}(c_4, w_4).\text{in}(c_4, w_2).\text{out}(c_4, w_5).
\text{out}(c_1, w_6).\text{out}(c_1, w_7)$

where $tr$ has been defined in Example 5. The attacker simply replays an old session. The resulting (unique) frames are:

- $\phi_1' = \phi \cup \phi' \cup \{w_6 \triangleright \text{enc}(m_1, k_{ab}), w_7 \triangleright \text{enc}(m_1, k_{ab})\}$,
- $\phi_2' = \phi \cup \phi' \cup \{w_6 \triangleright \text{enc}(m_2, k), w_7 \triangleright \text{enc}(m_2, k')\}$

where $\phi'$ is the frame given in Example 5, and

$\phi' = \{w_4 \triangleright (a, b), w_5 \triangleright \text{enc}(k_{ab}, a), k_{ba}\}.$
We have that \((\text{tr}', \phi'_1) \in \text{trace}_{\Sigma_0}(\mathcal{P}_{\text{DS}}^1)\) and \((\text{tr}', \phi'_2) \in \text{trace}_{\Sigma_1}(\mathcal{P}_{\text{DS}}^2)\). However, we have that \(\phi'_1 \not\subset \phi'_2\) since \(w_0 = w_7\) in \(\phi'_1\) but not in \(\phi'_2\). Thus \(\mathcal{P}_{\text{DS}}^1\) is not trace included in \(\mathcal{P}_{\text{DS}}^2\). To avoid this attack, the messages of the Denning-Sacco protocol shall include timestamps or nonces.

The goal of the paper is to provide an efficient and practical procedure for checking trace equivalence.

III. REDUCTION RESULTS

Even when considering finite processes (i.e. processes without replication), the problem of checking trace equivalence is difficult due to several sources of unboundedness:

- the size of messages which can be forged by an attacker is unbounded;
- the number of nonces and constants that can be used by an attacker is unbounded too.

Recently, [19] has established how to reduced the search space for attacks by bounding the size of messages involved in a minimal attack. From a theoretical point of view, this also yields a bound on the number of nonces/constants involved in such a minimal attack. However, this bound is far from being practical. In this section, we show that the small attack property of [19] still holds even if our semantics has slightly changed (due to eager evaluation) and we further demonstrate that the number of constants can be significantly reduced since only three constants need to be considered (and no nonces), in addition to those explicitly mentioned in the protocol.

A. Bounding the size of messages

As in [19], we consider type-compliant protocols, and we restrict ourselves to typing systems that preserve the structure of terms. A typing system is defined as follows.

Definition 5: A typing system is a pair \((\mathcal{T}_0, \delta_0)\) where \(\mathcal{T}_0\) is a set of elements called atomic types with a special atomic type denoted \(\tau_\alpha\), and \(\delta_0\) is a function mapping atomic terms in \(\Sigma_0 \cup \mathcal{N} \cup \mathcal{X}\) to types \(\tau\) generated using the following grammar:

\[
\tau, \tau_1, \tau_2 = \tau_0 \mid \langle \tau_1, \tau_2 \rangle \mid \text{enc}(\tau_1, \tau_2) \text{ with } \tau_0 \in \mathcal{T}_0.
\]

We further assume the existence of an infinite number of constants in \(\Sigma_0\) (resp. variables in \(\mathcal{X}\), names in \(\mathcal{N}\)) of any type, and the existence of three special constants denoted \(c_{\langle \omega, \omega \rangle}\), \(c^1_{\omega}\), and \(c^2_{\omega}\) of type \(\tau_\alpha\). The constant \(c_{\langle \omega, \omega \rangle}\) can not be used in key position. Then, \(\delta_0\) is extended to constructor terms as follows:

\[
\delta_0(f(t_1, \ldots, t_n)) = f(\delta_0(t_1), \ldots, \delta_0(t_n)) \text{ with } f \in \Sigma_c.
\]

Example 8: Continuing our running Example, we consider the typing system generated from the set \(\mathcal{T}_{\text{DS}} = \{\tau_\alpha, \tau_m, \tau_k, \tau_k\}\) of atomic types and the function \(\delta_{\text{DS}}\) that associates the expected type to each constant/name, and the following types to variables:

- \(\delta_{\text{DS}}(x_{AB}) = \delta_{\text{DS}}(x'_{AB}) = \delta_{\text{DS}}(y_{AB}) = \delta_{\text{DS}}(y'_{AB}) = \tau_k\);
- \(\delta_{\text{DS}}(x_B) = \delta_{\text{DS}}(x'_B) = \text{enc}(\langle \tau_k, \tau_k \rangle, \tau_k)\).

A protocol is type-compliant if two unifiable subterms have the same type. Formally, we use the definition given in [19], which is similar to the one introduced in [24].

We write \(St(t)\) (resp. \(St(\tau)\)) for the set of (syntactic) subterms of a term \(t\) (resp. type \(\tau\)), and \(ESt(t)\) the set of its encrypted subterms, i.e.

\[
ESt(t) = \{u \in St(t) \mid u \text{ is of the form } \text{enc}(u_1, u_2)\}.
\]

In the following definition, \(\delta_P(\mathcal{P})\) is the set of \(\delta_P(t)\) for every term \(t\) occurring in protocol \(\mathcal{P}\).

Definition 6: A protocol \(\mathcal{P}\) is type-compliant w.r.t. a typing system \((\mathcal{T}_P, \delta_P)\) if \(\tau_\alpha \notin \mathcal{H}(\delta_P(\mathcal{P}))\), and for every \(t, t' \in ESt(\mathcal{P})\) we have that:

\[
t \text{ and } t' \text{ unifiable implies that } \delta_P(t) = \delta_P(t').\]

Example 9: The protocol \(\mathcal{P}_{\text{DS}}^1\) (resp. \(\mathcal{P}_{\text{DS}}^2\)) is type-compliant w.r.t. the typing system given in Example 8. Indeed, the encrypted subterms of \(\mathcal{P}_{\text{DS}}^1\) are:

1) \(t_A = \text{enc}(\langle b, x_{AB}, x_B \rangle, k_{as})\);
2) \(t_B1 = \text{enc}(\langle y_{AB}, a \rangle, k_{ba})\);
3) \(t_B2 = \text{enc}(m_1, y_{AB})\);
4) \(t_S1 = \text{enc}(\langle b, k_{ab}, \text{enc}(\langle k_{ab}, a \rangle, k_{ba}) \rangle, k_{as})\);
5) \(t_S2 = \text{enc}(\langle k_{ab}, a \rangle, k_{as})\)

as well as the renaming of these terms obtained by replacing \(k_{ab}, x_{AB}, y_{AB}\), and \(x_B\) with fresh names/variables of the same type, namely \(k'_{ab}, x'_{AB}, y'_{AB}\), and \(x'_B\).

It is easy to check that the type-compliance condition is satisfied for any pair of terms. For instance, we have that \(t_A\) and \(t_{S1}\) are unifiable, and they have indeed the same type:

\[
\delta_{\text{DS}}(t_A) = \text{enc}(\langle \tau_\alpha, \tau_\alpha, \text{enc}(\langle \tau_k, \tau_k \rangle, \tau_k) \rangle, \tau_k) = \delta_{\text{DS}}(t_{S1}).
\]

Consider a protocol \(\mathcal{P}\) that is type-compliant w.r.t. to a typing system \((\mathcal{T}_P, \delta_P)\), an execution \(\mathcal{P} \xrightarrow{\sigma} (\mathcal{P}', \phi'; \sigma')\) is well-typed if \(\sigma'\) is a well-typed substitution, i.e. every variable of its domain has the same type as its image. We say that a trace \((\text{tr}, \phi) \in \text{trace}_{\Sigma_0}(\mathcal{P})\) is well-typed if its underlying execution (unique due to the class of protocols we consider in this paper) is well-typed. Given a protocol \(\mathcal{P}\), we denote \(\Sigma_\mathcal{P}\) the constants from \(\Sigma_0\) that occur in \(\mathcal{P}\).

We first show that the small attack property from [19] still holds: whenever two processes are not in trace equivalence, then there is a well-typed witness of non-equivalence. In addition, we show that the recipes involved in such a trace have a simple form: they are built using constructor symbols on top of destructors.

Definition 7: Let \(R\) be a recipe. We say that \(R\) is destructor-only if \(R \in T(\Sigma_d, \Sigma \cup W)\). It is simple if there exist destructor-only recipes \(R_1, \ldots, R_k\), and a context \(C\) made of constructors such that \(R = C[R_1, \ldots, R_k]\).

Theorem 1: Let \(\mathcal{P}\) be a protocol type-compliant w.r.t. \((\mathcal{T}_P, \delta_P)\) and \(Q\) be another protocol. We have that \(\mathcal{P} \not\xRightarrow{\phi} Q\) w.r.t. \(\Sigma_0\) if, and only if, there exists a witness \(\text{tr}\) of this non-inclusion that only contains simple recipes and such that one of the following holds:

1) \((\text{tr}, \phi) \in \text{trace}_{\Sigma_0}(\mathcal{P})\) for some \(\phi\) and \((\text{tr}, \phi)\) is well-typed w.r.t. \((\mathcal{T}_P, \delta_P)\);
2) \( \text{tr} = \text{tr}' \{ c_0 \mapsto c_{(\omega, \omega)} \} \) for some \( c_0 \in \Sigma_0 \cap \Sigma_P \), and 
\( (\text{tr}', \phi') \in \text{trace}_{\Sigma_0}(P) \) for some \( \phi' \) and \( (\text{tr}', \phi') \) is well-typed w.r.t. \( (T_P, \delta_P) \).

Since we consider atomic keys, some execution may fail when a protocol is about to output an encryption with a non
atomic key. In order to detect this kind of behaviours, it is important to consider slightly ill-typed traces as defined in
Item 2.

Example 10: Continuing our running example, we have seen that \( P^1_{DS} \not\sqsubseteq P^2_{DS} \). The witness \( \text{tr}' \) of this non-inclusion (given in Example 7) only contains simple recipes, and 
\( (\text{tr}', \phi') \in \text{trace}_{\Sigma_0}(P^1_{DS}) \) is well-typed w.r.t. \( (T_{DS}, \delta_{DS}) \) (the typing system given in Example 8).

B. Bounding the number of constants

The previous result implicitly bounds the number of constants
used in an attack but the induced bound would be impractical. We show here that actually, two constants are
sufficient. The proof technique is inspired from [25] and [26] which respectively reduce the number of nonces and agents
in the context of equivalence properties. A direct application of the proof technique would however yield two constants of
each type, which represents still a high number of constants. Instead, we show here that just two constants are enough,
provided they are of special type \( \tau_* \). To obtain this result, we slightly relax the notion of well-typedness.

Given a typing system \( (T_0, \delta_0) \), we denote by \( \leq \) the smallest relation on types defined as follows:

- \( \tau_* \leq \tau \) and \( \tau \leq \tau \) for any type \( \tau \) (atomic or not);
- \( f(\tau_1, \tau_2) \leq f(\tau'_1, \tau'_2) \) when \( \tau_1 \leq \tau'_1, \tau_2 \leq \tau'_2 \), and \( f \in \Sigma_c \).

Consider a protocol \( P \) that is type-compliant w.r.t. to a
typing system \( (T_P, \delta_P) \), an execution \( \text{tr} \xrightarrow{\Sigma} (P'; \phi'; \sigma') \) is quasi-well-typed if \( \delta_P(x; \sigma') \leq \delta_P(x) \) for every variable \( x \in \text{dom}(\sigma') \). We say that a trace \( \text{tr}(\phi) \in \text{trace}_{\Sigma}(P) \) is quasi-
well-typed if its underlying execution (unique due to the class of protocols we consider in this paper) is quasi-well-typed.

If two processes are not in trace equivalence, then there is
a witness of non equivalence that is quasi-well typed and uses
at most two extra constants plus eventually \( c_{(\omega, \omega)} \) to detect
slightly ill-typed traces.

Theorem 2: Let \( P \) be a protocol type-compliant w.r.t.
\( (T_P, \delta_P) \) and \( Q \) be another protocol. Let \( \Sigma = \Sigma_P \cup \{ c^1_0, c^1_1, c_{(\omega, \omega)} \} \). We have that \( P \not\sqsubseteq Q \) w.r.t. \( \Sigma_0 \) if, and only if, there exists a witness \( \text{tr} \) of this non-inclusion w.r.t. \( \Sigma_0 \) that only contains simple recipes, and such that \( \text{tr}(\phi) \in \text{trace}_{\Sigma_0}(P) \) for some \( \phi \) and \( \text{tr}(\phi) \) is quasi-well-typed w.r.t. \( (T_P, \delta_P) \).

Intuitively, we can show that non equivalence relies on at
most one disequality, and thanks to our equational theory,
only two constants \( c^1_0, c^1_1 \) are necessary to produce such a
disequality.

IV. FROM STATIC EQUIVALENCE TO PLANNING

The overall objective of this paper is to provide a practical
algorithm for deciding trace equivalence, using planning
graphs and SAT-solving. We start here with the static case and
show how to reduce static equivalence to a planning problem.
Given two frames, we show how to build a planning problem
such that the planning problem has a solution if, and only if,
the two corresponding frames are not in static equivalence.

We consider two frames \( \phi \) and \( \psi \) having same domain. We
denote \( \Sigma \) the constants from \( \Sigma_0 \) that occur either in \( \phi \) or in \( \psi \).

A. Planning problems

We first recall the definition of a planning problem, slightly
simplified from [27]. Intuitively, a planning system defines a
transition system from sets of facts to sets of facts. New facts
may be produced and some old facts may be deleted.

Definition 8: A planning system is tuple \( (\text{Fact}_0, \text{Init}, \text{Rule}) \) where \( \text{Fact} \) is a set of variable-free atomic formulas called
facts, \( \text{Init}_0 \subseteq \text{Fact} \) is a set of facts representing the initial
state, and \( \text{Rule} \) is a set of rules of the form:

\[ \text{Pre} \rightarrow \text{Add}; \text{Del} \]

where \( \text{Pre}, \text{Add}, \text{Del} \) are finite sets of facts such that \( \text{Add} \cap \text{Del} = \emptyset, \text{Del} \subseteq \text{Pre} \). We write \( \text{Pre} \rightarrow \text{Add} \) when \( \text{Del} = \emptyset \).

Given a rule \( r \in \text{Rule} \) of the form \( \text{Pre} \rightarrow \text{Add}; \text{Del} \), we
denote \( \text{Pre}(r) = \text{Pre}, \text{Add}(r) = \text{Add}, \text{and} \text{Del}(r) = \text{Del} \). Moreover, if \( S \subseteq \text{Fact} \) are such that \( \text{Pre}(r) \subseteq S \), then we
say that the rule is applicable in \( S \), denoted \( S \xrightarrow{r} S' \), and the
state \( S' = (S \setminus \text{Del}) \cup \text{Add} \) is the state resulting from the
application of \( r \) to \( S \). A planning path from \( S_0 \subseteq \text{Fact} \) to
\( S_n \subseteq \text{Fact} \) is a sequence of rules \( r_1, \ldots, r_n \in \text{Rule} \) such that
there exist states \( S_1, \ldots, S_{n-1} \subseteq \text{Fact} \) such that:

\[ S_0 \xrightarrow{r_1} S_1 \xrightarrow{r_2} \ldots S_{n-1} \xrightarrow{r_n} S_n \]

A planning problem for a system \( \Theta = (\text{Fact}, \text{Init}, \text{Rule}) \) is a pair \( \Pi = (\Theta, S_f) \) where \( S_f \subseteq \text{Fact} \) represents the target
facts. A solution to \( \Pi = (\Theta, S_f) \), called a plan, is a planning
path from \( \text{Init} \) to a state \( S_n \) such that \( S_f \subseteq S_n \).

In this paper, we consider an (infinite) set of facts \( \text{Fact}_0 \) that consists of:

- all atomic formulas of the form \( \text{att}(u_P, u_Q) \) with \( u_P, u_Q \in M_S \);
- all atomic formulas of the form \( \text{state}^c_{P, Q}(\sigma_P, \sigma_Q) \) where \( c \in Ch, P, Q \) are basic processes on channel \( c \), and \( \sigma_P \)
(resp. \( \sigma_Q \)) is a grounding substitution for \( P \) (resp. \( Q \));
- a special symbol \( \text{bad} \).

The rest of this section is dedicated to the reduction of
static equivalence to the (non) existence of a solution of a
planning system. Therefore, we will consider planning systems
with facts that represent the attacker’s knowledge, i.e. those
of the form \( \text{att}(u_P, u_Q) \) (plus the symbol \( \text{bad} \)). Later on, in
Section V, we will additionally consider the facts of the form
\( \text{state}^c_{P, Q}(\sigma_P, \sigma_Q) \) that model internal states of the agents.

B. Attacker rules

We first describe the planning rules that correspond to the
attacker behaviours. Instead of considering rules on ground
facts, we start by describing a set of abstract rules that we
instantiated later on, yielding a (concrete) planning system.
The attacker behaviour is modelled by the following set RuleA of abstract rules:

\[ \text{att}(\langle x_1, x_2 \rangle, \langle y_1, y_2 \rangle) \rightarrow \text{att}(x_1, y_1) \]
\[ \text{att}(\langle x_1, x_2 \rangle, \langle y_1, y_2 \rangle) \rightarrow \text{att}(x_2, y_2) \]
\[ \text{att}(\text{enc}(x_1, x_2), \text{enc}(y_1, y_2)), \text{att}(x_2, y_2) \rightarrow \text{att}(x_1, y_1) \]

Note that there is no Del since the attacker never forgets. Interestingly, the rules only model decomposition. There is no rule to synthesize messages. In general, this would be unsound but we will show why we can get rid of synthesis rules, thanks to the flattening technique. This is a key point of our algorithm to avoid building large terms.

We now explain how to obtain concrete planning rules from the abstract ones. This step is called concretization. Basically, we distinguish two kinds of concrete rules: the positive one, and the negative one. We start in this subsection by defining the positive one.

Given an abstract attacker rule \( r \in \text{RuleA} \), we define its positive concretizations by simply instantiating the abstract rules such that the resulting terms are messages.

Concrete\( ^+ (r) = \{ rs | \sigma \text{ a substitution grounding for } r \text{ such that } rs \sigma \text{ involves messages in } M_{\Sigma} \} \)

Let \( \phi \) and \( \psi \) be two frames with \( \text{dom}(\phi) = \text{dom}(\psi) \). The set of facts associated to \( \phi \) and \( \psi \) is defined as the set of couples of all identical constants and the couples of associated messages of the two frames.

\[ \text{Fact}(\phi, \psi) = \{ \text{att}(a, a) | a \in \Sigma \} \cup \{ \text{att}(w\phi, w\psi) | w \in \text{dom}(\phi) \} \]

It is easy to show that, applying (concrete) attacker rules to \( \text{Fact}(\phi, \psi) \), we compute the set of couples \( (u, v) \) that can be reached by applying destructor-only recipes to \( \phi \) and \( \psi \).

**Lemma 1**: Let \( \phi, \psi \) be two frames with \( \text{dom}(\phi) = \text{dom}(\psi) \). Let \( \Theta = \langle \text{Fact}_0, \text{Fact}(\phi, \psi), \text{Concrete}^+(\text{RuleA}) \rangle \) and \( \Pi = \langle \Theta, \{ \text{att}(u, v) \} \rangle \) for some \( u, v \in M_{\Sigma} \). We have that \( \Pi \) has a solution if, and only if, there is a destructor-only recipe \( R \in R_\Sigma \) such that \( R\phi \downarrow = u \), and \( R\psi \downarrow = v \).

**C. Case of failures**

To break static equivalence, an attacker may build new terms but also check for equalities and computation failures. Therefore, we encode when a computation can be performed on the right-hand side but can not be mimicked on the left.

We say that a fact \( f = \text{att}(u_0, v_0) \) \((u_0, v_0 \in M_{\Sigma})\) left-unifies (resp. right-unifies) with \( \text{att}(u, v) \) if there exists \( \sigma \) such that \( u\sigma = u_0 \) (resp. \( v\sigma = v_0 \)). Similarly, a sequence of facts \( \text{att}(u_1, v_1), \ldots, \text{att}(u_k, v_k) \) left-unifies with a sequence \( \text{att}(u_1', v_1'), \ldots, \text{att}(u_k', v_k') \) if there exists \( \sigma \) such that \( u_1' \sigma = u_1, \ldots, u_k' \sigma = u_k \) (and symmetrically for right-unification).

Given an abstract attacker rule \( r = \text{Pre} \rightarrow \text{Add} \) (note that Del is empty for attacker rule), we define Concrete\( ^- (r) \) as the set of concrete planning rules that contains:

\[ f_1, \ldots, f_k \rightarrow \text{bad} \]

for any sequence of facts \( f_1, \ldots, f_k \in \text{Fact}_0 \) such that \( f_1, \ldots, f_k \) left-unifies with \( \text{Pre} \), whereas \( f_1, \ldots, f_k \) does not right-unify with \( \text{Pre} \). This is the generic way to compute the failure rules from abstract attacker rules. In case of the set of abstract rules RuleA that we consider here, we obtain the following infinite set of rules, denoted Concrete\( ^- (\text{RuleA}) \):

\[ \text{att}(u_1, u_2, v) \rightarrow \text{bad} \]

for any \( u_1, u_2, v \in M_{\Sigma} \) such that \( v \) is not a pair \( \text{att}(\text{enc}(u_1, u_2), v), \text{att}(u_2, v') \rightarrow \text{bad} \)

for any \( u_1, u_2, v, v' \in M_{\Sigma} \) such that \( \text{enc}(u_1, u_2) \in M_{\Sigma} \), and \( d\psi \downarrow \notin M_{\Sigma} \).

In order to capture static inclusion, we have to consider some additional cases of failure, in particular those corresponding to an equality that holds in one side but not in the other side. For this, we introduce the set \( R_{\text{fail}}^\text{test} : \)

\[ R_{\text{fail}}^\text{test} = \{ \text{att}(u, v), \text{att}(u, v) \rightarrow \text{bad} | v_1 \neq v_2 \} \]

However, as exemplified below, due to the absence of rule to compose terms, this is not sufficient.

**Example 11**: Let \( \phi = \{ w \triangleright k \} \) and \( \psi = \{ w \triangleright \text{enc}(s, k) \} \) where \( s, k \in \mathcal{N} \). We have that \( \phi \nless \psi \). Indeed, consider \( R = \text{enc}(w, w) \), we have that \( R\phi \downarrow \subseteq M_{\Sigma} \) whereas \( R\psi \downarrow \notin M_{\Sigma} \). However, we have no mean to witness this non-inclusion without relying on synthesis rules (that we do not have).

Therefore, we introduce in addition the ability to check whether a message is an atom or not (different from the special constant \( c_{(\omega, \omega)} \)). More formally, we consider the set:

\[ R_{\text{fail}}^\text{atom} = \{ \text{att}(u, v) \rightarrow \text{bad} | u \text{ is an atom different from } c_{(\omega, \omega)} \text{ but not } v \} \]

Given a set Rule of abstract rules, we denote Concrete\( (\text{Rule}) = \text{Concrete}^+ (\text{Rule}) \cup \text{Concrete}^- (\text{Rule}) \).

Two frames are in static inclusion if, and only if, the corresponding planning system has no solution.

**Proposition 1**: Let \( \phi, \psi \) be two frames with \( \text{dom}(\phi) = \text{dom}(\psi) \), and \( \Theta = \langle \text{Fact}_0, \text{Fact}(\phi, \psi), R \rangle \) where \( R = \text{Concrete}(\text{RuleA}) \cup R_{\text{fail}}^\text{test} \cup R_{\text{fail}}^\text{atom} \).

Let \( \Pi = \Theta, \{ \text{bad} \} \). We have that \( \phi \nless \psi \) if, and only if, \( \Pi \) has a solution.

As we shall see later, in order to obtain an efficient algorithm, we do not enumerate all ground attacker rules. Instead, they are generated on the fly, only when they are needed.

**V. FROM TRACE EQUIVALENCE TO PLANNING**

In the previous section, we have shown how to encode static inclusion into a planning system. We now show how to encode trace inclusion. We consider a protocol \( \mathcal{P} \) that is type-compliant w.r.t. \( (T_\mathcal{P}, \delta_\mathcal{P}) \), and another protocol \( Q \). For simplicity we assume that variables of \( \mathcal{P} \) and \( Q \) are disjoint. Let \( \Sigma = \{ \Sigma_\mathcal{P} \cup \Sigma_Q \} \cup \{ c_0, c_1, c_{(\omega, \omega)} \} \). Moreover, we assume that variables occurring in \( \mathcal{P} \) are given with types.

**A. Protocol rules**

We first define the abstract rules describing the protocol behaviour. Given \( P \) and \( Q \) two basic processes on channel \( c \), we write \( \text{St}(P, Q) = \text{state}_{\mathcal{P}_Q}(id_P, id_Q) \) where \( id_P \) (resp. \( id_Q \)) is the identity substitution of domain \( fc(P) \) (resp. \( fc(Q) \)). Then,
the transformation Rule($P;Q$) from processes to abstract planning rules is defined as follows: We distinguish several cases depending on the shape of $P$.

1. Case $P = 0$:
   \[ \text{Rule}(P; Q) = \emptyset. \]

2. Case $P = \text{out}(c, u).P'$:
   \[ \text{Rule}(P; Q) = \text{Rule}(P'; Q') \cup \{ \text{St}(P, Q) \rightarrow \text{att}(u', v), \text{St}(P', Q'); \text{St}(P, Q) \} \]
   when $Q = \text{out}(c, v).Q'$
   \[ \text{Rule}(P; Q) = \{ \text{St}(P, Q) \rightarrow \text{att}(u, c_B), \text{bad} \} \]
   otherwise.

3. Case $P = \text{in}(c, u).P'$:
   \[ \text{Rule}(P; Q) = \text{Rule}(P'; Q') \cup \{ \text{St}(P, Q), \text{att}(u, v) \rightarrow \text{St}(P', Q'); \text{St}(P, Q) \} \]
   when $Q = \text{in}(c, v).Q'$
   \[ \text{Rule}(P; Q) = \{ \text{St}(P, Q), \text{att}(u, x) \rightarrow \text{bad} \} \]
   otherwise (with $x$ fresh).

Intuitively, abstract rules simply try to mimic each step of $P$ by a similar step of $Q$. Clearly, if $Q$ cannot follow $P$, the two processes are not in trace equivalence, which is modelled here by the bad state. It then remains to check whether the bad state is indeed reachable. Note that, in case $P = \text{out}(c, u).P'$ whereas $Q$ is not ready to perform an output, bad will be trigger only if the outputted term is indeed a message.

**Example 12:** We consider protocols $P_{DS}^1$ and $P_{DS}^2$ as given in Example 7. We focus on the computations of the abstract protocol rules for the basic process defined on channel $c_1$, i.e.

\[ \text{Rule}(P_A^1 \cdot \text{out}(c_1, \text{enc}(m_1, x_1^{AB})), P_A^2 \cdot \text{out}(c_1, \text{enc}(m_2, x_2^{AB}))) \]

where $P_A^i = \text{out}(c_1, \langle a, b \rangle), \text{in}(c_1, \text{enc}(b, x_1^{AB}, x_2^{AB}), k_{as})\), \text{out}(c_1, x_2^{AB})\), \text{out}(c_1, \text{enc}(m_1, x_1^{AB})$ with $i \in \{1, 2\}$.

We have simply renamed bound variables to ensure disjointness between the variables of $P_A^1$ and those of $P_A^2$. Moreover, for sake of conciseness, below, we write state instead of state$P_1,P_2$ where $P_1$ (resp. $P_2$) with $i \in \{1, 4\}$ represents the subprocess of $P_A^1$ (resp. $P_A^2$) starting at the $i$th action. We write $id_X$ the identity substitution with $\text{dom}(id_X) = X$. Since this basic process is made up of 4 actions, we obtain 4 abstract protocol rules, among which the following abstract rule $r_3$:

\[ \text{state}_{P_A^1}^3(id_{x_1^{AB}, x_1^{AB}}, id_{x_2^{AB}}) \rightarrow \text{att}(x_1^{AB}, x_2^{AB}), \text{state}_{P_A^2}^4(id_{x_1^{AB}}; \emptyset); \text{state}_{P_A^2}^3(id_{x_1^{AB}, x_1^{AB}}, id_{x_2^{AB}}) \]

Since both basic processes have the same shape, no abstract rule with bad in conclusion have been computed at this stage.

This transformation is then extended to protocols in a natural way. Assume w.l.o.g. that both simple processes are made of $n$ basic processes (we can complete with null processes if needed). That is, $P = \{P_1, \ldots, P_n\}$ and $Q = \{Q_1, \ldots, Q_n\}$. In addition, assume w.l.o.g. that $P_i$ and $Q_i$ are basic processes on channel $c_i$. We define

\[ \text{Rule}(P, Q) = \text{Rule}(P_1, Q_1) \cup \ldots \cup \text{Rule}(P_n, Q_n). \]

Given a substitution $\sigma$, and state$P_{DS}(\sigma_P, \sigma_Q)$ occurring in a protocol abstract rule, the application of $\sigma$ to the abstract state is defined as follows:

\[ \text{state}_{P_{DS}}(\sigma_P, \sigma_Q) = \text{state}_{P_{DS}}(\sigma \circ \sigma_P, \sigma \circ \sigma_Q). \]

**B. Flattening**

In terms of efficiency, one key step of our algorithm is to avoid composition rules from the attacker. For this, we transform protocol rules in order to pre-compute all necessary composition steps. For example, consider the second step of the Denning Sacco protocol, presented in Example 4. The agent $A$ expects a message $m$ of the form $\{b, x_{AB}, x_B\}_{k_{as}}$ and answers with $x_B$. Either the attacker obtains $m$ as an existing ciphertext or he builds the ciphertext himself, provided he knows the key $k_{as}$. In the later case, we may avoid a composition step by considering the following (informal) rule:

\[ b, x_{AB}, x_B, k_{as} \rightarrow x_B \]

This rule is clearly useless for this particular example but illustrates our flattening technique. Note that such rules will become useful for the analysis of a more complex scenario, in particular those involving dishonest participants.

We now explain how formally to compute the set of flattened rules from a given abstract rule $r$. For this, we start by explaining how to decompose a fact $\text{att}(u, v)$.

**Definition 9:** Given a term $u \in T(\Sigma_e, \Sigma \cup \mathcal{N} \cup \mathcal{X})$, we say that $u$ is decomposable when:

- either $u \in \mathcal{X}$ and $\delta_P(u)$ is not an atomic type;
- or $u \notin \Sigma \cup \mathcal{N} \cup \mathcal{X}$.

Intuitively, a variable of non atomic type is decomposable since it may be instantiated by a non atomic term which, in turns, may have been obtained by composition. Given $\text{att}(u, v)$ with $u$ decomposable, we define split$\text{att}(u, v)$ as follows:

\[ \text{split}(\text{att}(u, v)) = (f; \{\text{att}(x_1, y_1), \text{att}(x_2, y_2)\}; \sigma_P; \sigma_Q) \]

where

- $\delta_P(u) = f$ for some $\tau_1, \tau_2$ and some $f \in \Sigma_e$;
- $x_1$ (resp. $x_2$) is a fresh variable of type $\tau_1$ (resp. $\tau_2$) and $\sigma_P = \text{mgu}(u, f(x_1, x_2))$;
- $y_1, y_2$ are fresh variables, $\sigma_Q = \text{mgu}(v, f(y_1, y_2))$.

Note that $\sigma_P$ exists and is necessarily a well-typed substitution. By convention, we assume that $\text{mgu}(u, u') = \bot$ when $u$ and $u'$ are not unifiable.

Let $r$ be an abstract rule of the form $\text{Pre} \rightarrow \text{Add}; \text{Del}$ with $f = \text{att}(u, v) \in \text{Pre}$ such that $u$ is decomposable and $\text{split}(f) = (f, S, \sigma_P, \sigma_Q)$. The decomposition of $r$ w.r.t. $f$, denoted decompo$(r, f)$, is defined as follows:

1. $((\text{Pre} \setminus f) \cup S \rightarrow \text{bad}) \sigma_P$ in case $\sigma_Q = \bot$;
2. $((\text{Pre} \setminus f) \cup S \rightarrow \text{Add}; \text{Del})(\sigma_P \cup \sigma_Q)$ otherwise.

Then, decomposition is applied recursively on each rule.

\[ \text{Flat}(r) = \text{Flat}(\{\text{decompo}(r, f) \mid f = \text{att}(u, v) \in \text{Pre}(r) \text{ with } u \text{ decomposable}\}) \cup \{r\} \]
Example 13: Considering the abstract protocol rule $r_3$ given in Example 12, the set $\text{Flat}(r_3)$ contains (among others):

$$
\begin{align*}
\text{state}_2^c(\emptyset, \emptyset), \\
\text{att}(b, x_1^A, x_B^A), \text{att}(k_{as}, k_{as}) \\
\rightarrow \text{state}_2^c(\emptyset, \emptyset), \\
\text{state}_2^c(\emptyset, \emptyset), \\
\text{att}(b, b), \text{att}(x_1^A, x_2^A), \text{att}(k_{as}, k_{as}) \\
\rightarrow \text{state}_2^c(\emptyset, \emptyset)
\end{align*}
$$

where

- $\sigma_1 = \{ x_1^A \mapsto x_1^B, x_2^A \mapsto \text{enc}(x_2^B, x_2^B) \}$
- $\sigma_2 = \{ x_2^B \mapsto \text{enc}(x_2^B, x_2^B) \}$
- $x_2^B$ (resp. $x_2^B$) is of type $(\tau_b, \tau_a)$ (resp. $(\tau_b)$).

C. Concretization

Given an abstract rule $r$, we denote $\text{vars}_{\text{left}}(r)$ the variables occurring on the left (first parameter) of a predicate occurring in $r$, and similarly for $\text{vars}_{\text{right}}(r)$. More precisely,

- $\text{vars}_{\text{left}}(\text{att}(u, v)) = \text{vars}(u)$; and
- $\text{vars}_{\text{left}}(\text{state}_{P,Q}(\sigma_P, \sigma_Q)) = \text{vars}(\text{img}(\sigma_P))$.

We have that $\text{vars}(r) = \text{vars}_{\text{left}}(r) \uplus \text{vars}_{\text{right}}(r)$.

Given an abstract protocol rule $r$, its positive concretization simply consists in all its instantiations that are well-typed w.r.t. the left side of the rule.

Let $K' = (P; \phi; \psi)$ and $K_Q = (Q; \sigma_Q; \psi)$ be two configurations with $\text{dom}(\phi) = \text{dom}(\psi)$. The set of facts associated to $K'$ and $K_Q$ is defined as follows:

$$
\text{Fact}(K', K_Q) = \text{Fact}(\phi, \psi) \cup \{ \text{state}_{P,Q}(\sigma_P, \sigma_Q) | P \in P, Q \in Q \text{ are basic processes on channel } c, \sigma_P = \sigma_{P|f(P)} \text{ and } \sigma_Q = \sigma_{Q|f(P)} \}
$$

We denote by $\text{Fact}(K', K_Q) \uparrow S'$ when the set of facts $S'$ can be obtained from the set of facts $\text{Fact}(K', K_Q)$ by adding only deducible facts (using destructor recipes only).

Definition 10: Given two sets of facts $S$ and $S'$ such that $S = \text{Fact}(K', K_Q)$ with $K' = (P; \phi; \sigma_P)$ and $K_Q = (Q; \psi; \sigma_Q)$ with $\text{dom}(\phi) = \text{dom}(\psi)$, we write $\text{Fact}(K', K_Q) \uparrow S'$ when:

- $\text{Fact}(K', K_Q)$ and $S'$ coincide on states;
- for any $\text{att}(u, v) \in \text{Fact}(K', K_Q)$, $\text{att}(u, v) \in S'$; and
- for any $\text{att}(u, v) \in S'$, there exists a destructor-only recipe $R$ such that $R\phi|_u = u$ and $R\psi|_v = v$.

The solutions of the planning system obtained as the positive concretization of the abstract rules of $P$ and $Q$ exactly corresponds to the set of (quasi-well-typed) traces of $P$ that can be mimicked by $Q$.

Lemma 2: Let $P$ be a protocol type-compliant w.r.t. $(TP, TP)$, and $Q$ be another protocol. Let $\Theta$ be the following planning system:

$$
\langle \text{Fact}_0, \text{Fact}(P, Q), \mathcal{R} \rangle
$$

where $\mathcal{R} = \text{Concrete}^+(\text{Rule}_A \cup \text{Flat}(\text{Rule}(P, Q)))$.

Let $(tr, \phi) \in \text{trace}_{\Sigma}(P)$ for some $\phi$ and such that:

- $(tr, \phi)$ contains simple recipes;
- $(tr, \phi)$ is well-typed w.r.t. $(TP, TP)$;
- $(tr, \psi) \in \text{trace}_{\Sigma}(Q)$ for some $\psi$.

Then, there exist a planning path $r_1, \ldots, r_n$ of some length $n$ from $\text{Fact}(P, Q)$ to some $S_n$ such that $\text{Fact}(K'_{P}, K_Q) \uparrow S_n$ where $K'_{P}$ (resp. $K_Q$) is the resulting configuration starting from $P$ (resp. $Q$) and executing $tr$.

Conversely, let $r_1, \ldots, r_n$ be a planning path from $\text{Fact}(P, Q)$ to $S_n$ such that $\text{bad} \notin S_n$. Then, there exist a trace $tr$ and frames $\phi$ and $\psi$ such that:

- $(tr, \phi)$ contains simple recipes;
- $(tr, \phi)$ is well-typed w.r.t. $(TP, TP)$;
- $(tr, \psi) \in \text{trace}_{\Sigma}(Q)$ for some $\psi$; and
- $\text{Fact}(K'_{P}, K_Q) \uparrow S_n$ where $K'_{P}$ (resp. $K_Q$) is the resulting configuration starting from $P$ (resp. $Q$) and executing $tr$.

D. Case of failures

Similarly to the static case, we need to make sure that we can detect when $P$ and $Q$ are not in trace inclusion. For this, we consider additional rules that express when a step that can be performed on the left hand side cannot be mimicked on the right hand side.

Given an abstract protocol rule $r = \text{Pre} \rightarrow \text{Add}; \text{Del}$, $\text{Concrete}^-(r)$ is the set of planning rules that contains:

$$
f_1, \ldots, f_k \rightarrow \text{bad}
$$

for any sequence of facts $f_1, \ldots, f_k \in \text{Fact}_0$ such that $f_1, \ldots, f_k$ left-unify with $\text{Pre}$ with substitution $\sigma_L$ and $u \in \mathcal{M}_X$ for any $\text{att}(u, v) \in \text{Add}\sigma_L$, and such that one of the following conditions holds:

- $f_1, \ldots, f_k$ does not right-unify with $\text{Pre}$;
- $f_1, \ldots, f_k$ right-unify with $\text{Pre}$ with substitution $\sigma_R$ but $v \notin \mathcal{M}_X$ for some $\text{att}(u, v) \in \text{Add}\sigma_R$.

Our main technical result states that our encoding in planning system is sound and complete: two protocols are in trace inclusion if, and only if, the corresponding planning system (obtained by considering both positive and negative concretizations) has a solution.

Theorem 3: Let $P$ a protocol type-compliant w.r.t. $(TP, TP)$, and $Q$ be another protocol. We consider the following set $\mathcal{R}$ of concrete rules:

$$
\mathcal{R} = \text{Concrete}(\text{Rule}_A \cup \text{Flat}(\text{Rule}(P, Q))) \cup \mathcal{R}_{\text{fail}}^\text{test} \cup \mathcal{R}_{\text{fail}}^\text{atom}
$$

Let $\Theta = \langle \text{Fact}_0, \text{Fact}(P, Q), \mathcal{R} \rangle$ and $\Pi = \langle \Theta, \{ \text{bad} \} \rangle$. We have that $P \not\subseteq Q$ if, and only if, $\Pi$ has a solution.
This reduction to a planning system is a key ingredient of our result. But of course, it does not immediately yield an algorithm since the planning system encoding trace inclusion of a process \( P \) w.r.t. a process \( Q \) is actually infinite. Indeed, consider for example the positive concretizations of an abstract rule in Rule\((P; Q)\). There are finitely many instantiations for the “left” part, that corresponds to \( Q \) thanks to the typing system. However, the “right” part (corresponding to \( P \)) may be instantiated arbitrarily. We explain how to design an (efficient) algorithm in the next section.

VI. ALGORITHM

Our algorithm takes as input a protocol \( P \) that is type-compliant w.r.t. a typing system \((T_P, \delta_P)\) and another protocol \( Q \). We explain here how to check trace inclusion of \( P \) in protocol \( Q \). Then, trace equivalence is obtained by checking trace inclusion of \( P \) in \( Q \) and \( Q \) in \( P \).

Step 1: Compute the abstract rules of \((P; Q)\). As explained in Section V-A, we compute the abstract rules Rule\((P; Q)\) associated to \((P; Q)\), and then their flattened version flat(Rule\((P; Q)\)), as described in Section V-B. Together with the attacker rules (defined in Section IV-B), this yields

\[
\text{Rule}_A \cup \text{flat}(\text{Rule}(P; Q)).
\]

Step 2: Initial state. Thanks to Theorem 2, it is sufficient to consider at most three extra constants in addition to the constants of \( P \) and \( Q \), that is, it is sufficient to consider \( \Sigma = \Sigma_P \cup \Sigma_Q \cup \{c_0, c_1, c_{(\omega,\omega)}\} \). We then add the initial states of the protocols. More formally, we start with the initial state

\[
\text{Fact}(P, Q).
\]

Step 3: Planning graph algorithm. Given a planning system, the standard technique for finding a solution to the planning system is to apply the planning graph algorithm (see [14]), that we briefly recall here. The algorithm consists in building a graph (called planning graph), that consists in an alternance of facts layers and rules layers, linked with four kinds of edges: \( Pre \), \( Add \) and \( Del \) edges, that are edges between the fact layers and the rule layers; and mutex (as in mutual exclusion) are edges between vertices of the same layers. Mutex edges indicate when vertices may not be obtained simultaneously.

More precisely, the planning graph algorithm proceeds as follows. Let \( i \) denote the number of layers. Initially, \( i := 0 \).

1) The first fact layer is \( N^i_0 = \text{Fact}(P, Q) \) (the set of initial facts) and the first rule layer is empty, \( N^i_0 = \emptyset \).

2) From the fact layer \( N^i_f \), compute the set \( R \) of all concrete rules (either from Concrete\(^+\) or Concrete\(\sim\)) that are applicable from \( N^i_f \) without any mutex edge between facts of their precondition. Since there are a finite number of abstract protocol rules and since the facts in \( N^i_f \) are ground and finite, the set \( R \) of concrete rules applicable from \( N^i_f \) is finite as well.

3) Compute the new mutex edges between the rules. Rules are in mutex if they either interfere (one deletes a precondition or an add-effect of the other) or have competing needs (there is a mutex edge between their preconditions).

4) Build \( N^i_{i+1} \) from \( N^i_f \) by adding the facts introduced by the rules in \( N^i_f \). We have that:

\[
N^i_{i+1} = \bigcup_{r \in N^i_f} r.
\]

5) We compute the mutex edges between facts. There is a mutex edge between two facts \( f, f' \) if each rule that adds \( f \) is in mutex with each rule that adds \( f' \).

6) \( i := i + 1 \)

7) Check whether \( N_{i+1} := N_{i+1}^i \) (same facts and same mutex). If yes, then stop. Otherwise, go back to Point 2.

When the planning graph algorithm stops, we obtain a graph, that represents an over-approximation of the states reachable from the planning system, starting from the initial state. While we are looking for a solution to an infinite planning system (finitely described through abstract rules), we only need to consider a finite number of concrete rules at each round of the algorithm (Point 2 of the algorithm). Note that this construction is not a naive saturation that would explore all possible paths. The mutex edges ensure a not too coarse over-approximation and provide a mean for considering the application of a rule to a family of facts instead a single fact.

Step 4: SAT encoding. If bad does not occur in the resulting planning graph, then trace inclusion is guaranteed since the planning graph is an over-approximation of the reachable states. If bad does occur in the planning graph, we can check whether bad is indeed reachable through SAT solvers. More precisely, we encode the existence of a solution to the planning system into a SAT instance, using the same technique as SATMC (see [16]), and relying on the SAT solver minisat [28].

If bad is reachable, the SAT solver provides us with a solution, which is translated back to an attack trace. If bad is not reachable (that is, the SAT solver guarantees that there is no solution), then trace inclusion is guaranteed.

Conclusion. Thanks to Theorem 3, \( P \not\subseteq Q \) if, and only if, the corresponding planning system \( R \) has a solution, that is, bad is reachable. Therefore our algorithm is correct and complete: it provides an attack if, and only if, \( P \not\subseteq Q \). Since \( P \approx_t Q \) is defined as \( P \subseteq_t Q \) and \( Q \subseteq_t P \), we can then easily check whether two processes are in trace equivalence (\( P \approx_t Q \)).

Termination. Our procedure is not guaranteed to terminate. This may be surprising since Theorem 1 ensures that it is sufficient to consider traces that are well-types w.r.t. \((T_P, \delta_P)\). Then, since the processes are deterministic, a given trace of \( P \) can only be followed by at most one trace in \( Q \), hence a finite number of traces need to be considered. However, the planning graph step over-approximates the set of facts that need to be considered. Therefore, it may consider several facts of the form \( att(u, u_1), att(u, u_2), \ldots, att(u, u_n) \) with distinct, uncontrolled, \( u_i \). One way to enforce termination would be to check at each step that the planning graph only considers reachable facts (applying our SAT encoding). However, this would considerably slow down our algorithm while our experiments show that, not only our algorithm
terminates in practice, but it is also much more efficient than other existing tools.

VII. CASE STUDIES

In this section, we analyse several protocols of the literature and compare the results obtained using different tools that decide equivalence for a bounded number of sessions. The characteristics of these tools are given in Section VII-A, the different scenarios including some scenarios with corruption are described in Section VII-B. The result are described In Section VII-C and a discussion is provided in Section VII-D. Our tool as well as the source files to reproduce the benchmarks are available at [20].

A. Tools

Spec [9], [29] deals with a fixed set of cryptographic primitives, namely symmetric encryption and pairs, and protocols with no else branch. The procedure is sound and complete w.r.t. open bisimulation (a notion that is strictly stronger than trace equivalence [30]) and its termination is proved [9].

Apte decides trace equivalence [8], [11], [31] for a fixed but richer set of cryptographic primitives (i.e. symmetric/asymmetric encryptions, signature, pair, and hash functions). Processes are also more general: they include private channels, internal communications, processes that are not necessarily simple, and possibly with else branches.

Systems we are interested in are highly concurrent and a naive exploration of all possible interleavings limits the practical impact of those tools. Recent works [12], [13] have partially addressed the state space explosion problem due to naive exploration of all possible interleavings implemented in this tool. These dedicated partial order reduction (POR) techniques have been implemented in Apte-por (as an option of the Apte tool) yielding a significant speed-up.

Akiss implements the procedure described in [10], [32] and deals with rich user-defined term algebras including symmetric encryption and pairs. It is able to check an over-approximation of trace equivalence that actually coincides with trace equivalence for the class of simple processes that we study in this paper. Its termination has been established for the particular set of primitives used in this paper [33], and the performance of the tool has been recently improved relying on POR techniques mentioned above.

Of course, not all the tools consider exactly the same semantics. For example, Akiss considers a true equational theory while Spec, Akiss, and SAT-equiv consider a rewrite system (with again subtle differences). We believe nevertheless that they prove very similar properties and we therefore compare here their performance.

B. Scenarios with corruption

The scenario we considered so far for the Denning-Sacco protocol is quite simple. We only consider two sessions involving honest agents. This scenario involves 6 roles in parallel, and is denoted DS-6 in the table given in Section VII-C.

In the same spirit, we consider a simpler scenario, denoted DS-3, that corresponds to only one instance of each role (between honest agents). Such scenarios are known to be too simplistic and some attacks may be missed.

To go further, we consider scenario where honest agents are willing to engage communications with a dishonest agent c. Let us develop this corruption scenario on the Denning-Sacco protocol. Formally, we consider in addition of the three basic processes used to model scenario DS-3, a basic process to model that the agent a may be involved in another session with a corrupted agent c, and the server S is ready to answer a request coming from them. Similarly, we consider also two additional basic processes to model the fact that agent b may be involved in another session where the role A is played by the corrupted agent c. This scenario is therefore made up of 7 basic processes and is named accordingly DS-7.

To be more complete, we can also consider the cases where the role of A is played by b, and the role of B is played by a (scenario DS-10), and then we add again processes to model sessions with a corrupted agent (scenario DS-12 and DS-14).

We consider the case where the property is encoded on role B (strong secrecy of the key as received by B). We may also decide to encode the property on the two instances of the roles of B (scenario DS-6-bis) or only once (scenario DS-6).

C. Review of symmetric key protocols

Most of the protocols we considered from [23] actually fall in our class. We sometimes need to include some explicit tags to ensure type-compliance (this check is performed automatically by our tool). We now report on experimental results. We ran the different tools on a single Intel 3.1 GHz Xeon core with 190Go of RAM (shared with the other 19 cores) and we compare their performances on several protocols. For SAT-Equiv, we further indicate the number of ground facts and rules considered when computing the planning graph.

We decide to stop each experiment after 24h, and we indicate by TO (Time Out) when the tool does not return an answer within this timeframe, SO when we encounter a stack overflow, and MO in case the tool used more than 64 Go of Memory. We encountered some bugs that are indicated by BUG when interacting with Apte (internal errors or wrong results). We have reported these bugs to the authors.

Some protocols are subject to replay attacks, detected by the scenario 6-bis mentioned earlier. Even if scenarios that correspond to an attack are less interesting regarding performances comparison (since most of the tools stop their exploration once an attack has been found), we report the corresponding analysis in the last row of each table, whenever applicable, that is, whenever there is indeed an attack.

Denning Sacco. The Denning Sacco protocol has been described in Example 4. There is a replay attack on DS-6-bis due to a lack of freshness on the messages that are exchanged. This attack is similar to the one explained in Example 7.
**Wide Mouth Frog.** We consider the protocol as described in [23] but without timestamps as described below:

\[
A \rightarrow S : A, \{B, K_{ab}\} K_{as} \\
S \rightarrow B : \{A, K_{ab}\} K_{bs}
\]

Therefore, there is a replay attack on WMF-6-bis due to a lack of freshness on the messages that are exchanged.

<table>
<thead>
<tr>
<th>WMF</th>
<th>Spec</th>
<th>Akiss</th>
<th>Apte</th>
<th>Apte-por</th>
<th>Sat-Eq</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6s</td>
<td>0.04s</td>
<td>0.06s</td>
<td>0.01s</td>
<td>0.10s</td>
</tr>
<tr>
<td>6</td>
<td>58m</td>
<td>1.6s</td>
<td>55m</td>
<td>1.5s</td>
<td>1s</td>
</tr>
<tr>
<td>7</td>
<td>TO</td>
<td>5.3s</td>
<td>TO</td>
<td>2s</td>
<td>2s</td>
</tr>
<tr>
<td>10</td>
<td>8m30s</td>
<td>TO</td>
<td>22m</td>
<td>7s</td>
<td>40s</td>
</tr>
<tr>
<td>12</td>
<td>SO</td>
<td>128</td>
<td>118s</td>
<td>118s</td>
<td>312</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6-bis</td>
<td>13m</td>
<td>5.7s</td>
<td>0.06s</td>
<td>0.06s</td>
<td>1s</td>
</tr>
</tbody>
</table>

**Needham-Schroeder.** We consider the Needham-Schroeder protocol based on symmetric encryption as described in [23] (see below).

\[
A \rightarrow S : A, B, N_a \\
S \rightarrow A : \{B, N_a, K_{ab}\} K_{as} \\
A \rightarrow B : \{A, K_{ab}\} K_{bs} \\
B \rightarrow A : \{\text{Req}, N_b\} K_{ab} \\
A \rightarrow B : \{\text{Rep}, N_b\} K_{ab}
\]

<table>
<thead>
<tr>
<th>NS</th>
<th>Spec</th>
<th>Akiss</th>
<th>Apte</th>
<th>Apte-por</th>
<th>Sat-Eq</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>63s</td>
<td>4.4s</td>
<td>0.4s</td>
<td>0.03s</td>
<td>2s</td>
</tr>
<tr>
<td>6</td>
<td>MO</td>
<td>TO</td>
<td>TO</td>
<td>11m</td>
<td>54s</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td>153s</td>
<td>342</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td>8m</td>
<td>475</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td>22m</td>
<td>622</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td>77m</td>
<td>838</td>
</tr>
</tbody>
</table>

**Yahalom-Paulson.** We consider the protocol as described in [23]. To ensure type-compliance, we consider a tagged version of the protocol.

\[
A \rightarrow B : A, N_a \\
B \rightarrow S : B, N_b, \{1, A, N_a\} K_{bs} \\
S \rightarrow A : N_b, \{2, B, K_{ab}, N_a\} K_{as}, \{3, A, B, K_{ab}, N_b\} K_{bs} \\
A \rightarrow B : \{3, A, B, K_{ab}, N_b\} K_{bs}, \{4, N_b\} K_{ab}
\]

<table>
<thead>
<tr>
<th>YL</th>
<th>Spec</th>
<th>Akiss</th>
<th>Apte</th>
<th>Apte-por</th>
<th>Sat-Eq</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>11s</td>
<td>3s</td>
<td>12s</td>
<td>0.12s</td>
<td>5s</td>
</tr>
<tr>
<td>6</td>
<td>MO</td>
<td>TO</td>
<td>TO</td>
<td>35m</td>
<td>BUG</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td>19m</td>
<td>549</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td>206m</td>
<td>934</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td>19h</td>
<td>1391</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td>TO</td>
<td></td>
</tr>
</tbody>
</table>

**Otway-Rees.** We have also analysed a tagged version of the Otway-Rees protocol (see [23]).

\[
A \rightarrow B : M, A, B, \{1, N_a, M, A, B\} K_{as} \\
B \rightarrow S : M, A, B, \{1, N_a, M, A, B\} K_{as}, \\
\{2, N_b, M, A, B\} K_{bs} \\
S \rightarrow A : M, \{3, N_a, K_{ab}\} K_{as}, \{4, N_b, K_{ab}\} K_{bs} \\
B \rightarrow A : M, \{3, N_a, K_{ab}\} K_{as}
\]

<table>
<thead>
<tr>
<th>OR</th>
<th>Spec</th>
<th>Akiss</th>
<th>Apte</th>
<th>Apte-por</th>
<th>Sat-Eq</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>16m</td>
<td>225s</td>
<td>BUG</td>
<td>24s</td>
<td>104s</td>
</tr>
<tr>
<td>6</td>
<td>MO</td>
<td>TO</td>
<td>TO</td>
<td>SO</td>
<td>46m</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td>SO</td>
<td>50m</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td>SO</td>
<td>276m</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td>9h40m</td>
<td>1265</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td>TO</td>
<td></td>
</tr>
</tbody>
</table>

**Simple stateful example.** Some protocols are stateful (see [34] for a detailed discussion). For example, a process may lock a resource which cannot be used until it is unlocked. We consider here a mock protocol that reflects this type of behaviors. The protocol \(P_{\text{yes}}(n)\) with \(n\) tokens is described informally below \((1 \leq i \leq n)\), and is made of \(3n\) processes running in parallel on distinct channels.

1. \(\text{tok}_i \rightarrow \{\text{tok}_i\}_{k_i}, \{\text{tok}_i\}_{k_i}\)
2. \(\{x\}_{k_i} \rightarrow x\)
3. \(\text{tok}_i, \text{tok}_i \rightarrow \text{yes}\)

Here, yes and no are public constants, whereas \(k_i, \text{tok}_i, \text{tok}_i\) are names unknown by the attacker. The protocol \(P_{\text{pos}}(n)\) can be defined similarly. Intuitively, \(P_{\text{yes}}(n) \approx P_{\text{pos}}(n)\) holds since rule 2 can be used only once for each key \(k_i\). Therefore,
it is never possible to trigger a rule of type 3. We checked equivalence using ProVerif and, unsurprisingly, it found a false attack. This is due to the fact that ProVerif cannot properly model “a finite amount of time”.

<table>
<thead>
<tr>
<th># tok.</th>
<th>Spec</th>
<th>Akiss</th>
<th>Apte</th>
<th>Apte-por</th>
<th>Sat-Eq</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15s</td>
<td>0.02s</td>
<td>0.09s</td>
<td>0.01s</td>
<td>0.16s</td>
</tr>
<tr>
<td>2</td>
<td>MO</td>
<td>0.37s</td>
<td>240m</td>
<td>0.15s</td>
<td>1s</td>
</tr>
<tr>
<td>3</td>
<td>18s</td>
<td>MO</td>
<td>5s</td>
<td>2s</td>
<td>144</td>
</tr>
<tr>
<td>4</td>
<td>SO</td>
<td>9min32s</td>
<td>6s</td>
<td>188</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>TO</td>
<td>155s</td>
<td>540</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
<td>85m</td>
<td>1596</td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
<td>6h40m</td>
<td>2652</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

D. Discussion

For ease of comparison, we decided to run our experiments using a single core machine since not all the tools are able to take advantage of more cores. Running these examples using more cores would have benefited to our tool that reaches its optimum when it is launched using 4 cores (2 inclusions have to be checked with constants \( c_0 \) and \( c_1 \) (resp. \( c_{\omega,\omega} \)), and also to Akiss on which the saturation process is highly parallelizable.

The obtained results give evidence that our technique is less sensitive to the number of concurrent sessions analysed. On the contrary, the other tools that handle messages symbolically are less sensitive to the size of messages, which explains why our tool is typically slower on a small number of sessions. Moreover, on all our secure examples on which no attack is found, the planning graph is an over-approximation that appears to be precise enough, and does not require calls to the SAT solver. For the examples where an attack has been found (DS-6-bis and WMF-6-bis), the resulting SAT formulas contain about 750 variables and 4000 clauses.

VIII. Conclusion

Our tool SAT-Equiv outperforms all existing tools, even for the new Apte-por variant of Apte and the recently updated Akiss tool on which POR techniques have also been integrated. We also discovered several bugs in Apte-por, which prevented us from a thorough comparison of the two tools. SAT-Equiv is sometimes slower for a small number of sessions but in all cases, SAT-Equiv is the tool that allows to analyze the largest number of sessions.

One limitation of our tool is the fact that it covers protocols with symmetric encryption only. This is not an intrinsic limitation of our approach but rather a current limitation of the typing result [19], which states that we can limit ourselves to well-type attack traces. We plan to extend [19] to all standard primitives and we believe that the extension to SAT-Equiv to all standard primitives would then follow quite easily.

Note also that our tool is not guaranteed to terminate. We could enforce termination by checking reachability of the considered facts while building the planning graph, at the price of considerably slowing down our tool. Instead, as future work, we plan to formally prove termination of the planning graph construction or to identify under which assumptions, termination can be guaranteed.

Acknowledgments

This work has been partially supported by the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation program (grant agreements No 645865-SPOOC and No 714955-POPSTAR) and the DGA.
REFERENCES


A. Bounding the size of messages

The goal of this section is to prove Theorem 1. A similar result is stated and proved in [19] but for a slightly different semantics regarding rewriting of terms. Here, whenever an inner decryption/projection fails then the overall evaluation fails. Intuitively, we model eager evaluation while [19] models lazy evaluation. More formally, in [19], a term $u$ can be rewritten in $v$ if there is a position $p$ in $u$, and a rewriting rule $\ell \rightarrow r$ and a substitution $\sigma$ such that $u[p] = \ell \sigma$, and $v = u[\tau \sigma]_p$, i.e. $v$ in which the subterm at position $p$ has been replaced by $\tau \sigma$. We will denote $u \Rightarrow v$ this notion of rewriting, and $u \downarrow$ the associated notion of normal form of $u$.

Example 14: Let $u = \text{dec(enc}(c, \text{proj}_{j_1}(a)), \text{proj}_1(a))$. We have that $u \Rightarrow c$ whereas $u$ cannot be reduced w.r.t. $\rightarrow$ since $\text{proj}_1(a)$ is not a message.

Definition 11: A recipe $R$ is without detour if it does not contain any subterm of the form $\text{dec}(\text{enc}(R_1, R_2, R_3))$, $\text{proj}((R_1, R_2))$, and $\text{proj}_2((R_1, R_2))$ for any recipes $R_1, R_2, R_3$.

Lemma 3: Let $\phi$ be a frame, and $R$ be a recipe without detour. If $R\phi \downarrow \in \mathcal{M}_{\Sigma_0}$ then we have that $R\phi \downarrow \in \mathcal{M}_{\Sigma_0}$. Moreover, we have that $R$ is actually a recipe made of constructors on top of destructors.

Proof. We first show by structural induction on $R$ the following claim.

Claim. Let $\phi$ be a frame, $R$ be a recipe without detour such that $\text{root}(R) \notin \Sigma_c$, and $\text{root}(R\phi \downarrow) \notin \Sigma_d$. We have that $R\phi \downarrow = R\phi \downarrow \in \mathcal{M}_{\Sigma_0}$ and $R$ is destructor-only.

Base case: $R = w$ for some $w \in \text{dom}(\phi)$. In such a case, the result trivially holds.

Inductive case: $R = \text{dec}(R_1, R_2)$ (or $R = \text{proj}_{i}(R')$ with $i \in \{1, 2\}$). First, we know that $\text{root}(R_1) \neq \text{enc}$ since $R$ is without detour, and we have also that $\text{root}(R_1) \neq \bot$ since $\text{root}(\text{dec}(R_1, R_2)\phi \downarrow) \notin \Sigma_d$. Therefore, we have that $\text{root}(R_1) \notin \Sigma_c$. We have that $R\phi \downarrow = \text{dec}(R_1, R_2)\phi \downarrow = \text{dec}(R_1\phi \downarrow, R_2\phi \downarrow)$. Since $\text{root}(R\phi \downarrow) \notin \Sigma_d$, we deduce that $\text{root}(R_1\phi \downarrow) \notin \Sigma_d$. Therefore, we can apply our induction hypothesis on $R_1$, and we deduce that $R_1\phi \downarrow = R_1\phi \downarrow \in \mathcal{M}_{\Sigma_0}$, and $R_1$ is destructor-only. We know that $R_1\phi \downarrow = R_1\phi \downarrow = \text{enc}(u_1, u_2)$ and $R_2\phi \downarrow = u_2$ for some terms $u_1, u_2$. Since $\text{enc}(u_1, u_2) \in \mathcal{M}_{\Sigma_0}$, we have that $u_2 \in \mathcal{M}_{\Sigma_0}$, and since it occurs in key position, it is an atom. Therefore, we have that $\text{root}(R_2) \notin \Sigma_c$, and $\text{root}(R_2\phi \downarrow) \notin \Sigma_d$. Thus, we can apply our induction hypothesis, and we obtain that $R_2\phi \downarrow = R_2\phi \downarrow \in \mathcal{M}_{\Sigma_0}$, and $R_2$ is destructor-only. This allows us to conclude that $R = \text{dec}(R_1, R_2)$ is destructor-only and $R\phi \downarrow = R\phi \downarrow \in \mathcal{M}_{\Sigma_0}$.

The case where $R = \text{proj}_{i}(R')$ with $i \in \{1, 2\}$ can be done in a similar way. This concludes the proof of the claim.

Now, we prove the result stated in the lemma. Let $R$ be a recipe without detour. Let $C$ be a context built using constructor symbols and $R_1, \ldots, R_k$ be recipes such that $\text{root}(R_i) \notin \Sigma_c$ (for $i \in \{1, \ldots, k\}$), and $R = C[R_1, \ldots, R_k]$. Note that $C$ can be the empty context in case $\text{root}(R) \notin \Sigma_c$, and in such a case we conclude thanks to the previous claim. Otherwise, we have that $R\phi \downarrow = C[R_1\phi \downarrow, \ldots, R_k\phi \downarrow]$, and therefore $R_i\phi \downarrow \in \mathcal{M}_{\Sigma_0}$ for $i \in \{1, \ldots, k\}$, and $R_1, \ldots, R_k$ are recipes without detour. We apply our claim and we obtain that, for $1 \leq i \leq k$, $R_i\phi \downarrow = R_i\phi \downarrow \in \mathcal{M}_{\Sigma_0}$ and $R_i$ is destructor-only. This concludes the proof.

Lemma 4: Let $\phi$ be a frame, and $R$ be a recipe without detour. If $R\phi \downarrow \notin \mathcal{M}_{\Sigma_0}$ then there exists $f \in \Sigma_c \cup \Sigma_d$ such that:

- $R = f(R_1, \ldots, R_k)$ for some $R_1, \ldots, R_k$;
- $R\phi \downarrow = f(u_1, \ldots, u_k)$ for some $u_1, \ldots, u_k$; and
- $R_1\phi \downarrow = u_1, \ldots, R_k\phi \downarrow = u_k$.

Proof. We show this result by induction on $R$.

Base case: $R = w$. In such a case, $R\phi \downarrow = R\phi \downarrow \in \mathcal{M}_{\Sigma_0}$, and we are done.

Inductive step. Consider the case where $R = \text{enc}(R_1, R_2)$. In such a case, we have that $R\phi \downarrow = \text{enc}(R_1\phi \downarrow, R_2\phi \downarrow)$, and we are done. The case where $R = (R_1, R_2)$ can be done in a similar way. Now, consider the case where $R = \text{dec}(R_1, R_2)$. In such a case, we have that $R\phi \downarrow = \text{dec}(R_1\phi \downarrow, R_2\phi \downarrow)\phi \downarrow$. In case the dec symbol at the root position does not reduce, we are done. Otherwise, we have that $R_1\phi \downarrow = \text{enc}(u_1, u_2)$, and $R_2\phi \downarrow = u_2$ for some $u_1, u_2$. Moreover, we know that $u_1 \notin \mathcal{M}_{\Sigma_0}$, and thus $R_1\phi \downarrow \notin \mathcal{M}_{\Sigma_0}$, and thanks to our induction hypothesis, we have that $\text{root}(R_1) = \text{enc}$, and this contradicts the fact that $R$ is without detour. The case where $R = \text{proj}_{i}(R_1)$ can be done in a similar way.

Lemma 5: Let $R$ be a recipe. There exists a recipe $R'$ without detour such that $R'\phi \downarrow = R\phi \downarrow$ for any frame $\phi$ such that $R\phi \downarrow \in \mathcal{M}_{\Sigma_0}$.

Proof. We introduce special notation to describe terms in this proof. For any terms $t, t'$ and position $p$ of the term $t$, $t'_p$ is the subterm of $t$ at position $p$, and $t[t'_p]$ denotes the term $t$ in which the term occurring at position $p$ has been replaced by $t'$. Given a term $t$ and a position $p$ of $t$, we denote $\text{seq}_p(t)$ the sequence of function symbols encountered along the path from $\epsilon$ (the root of $t$) to $p$.

Let $R$ be a recipe. We consider a derivation in prefix form from $R$ to its normal form w.r.t. the following linear rewriting system. Note that the last rule is not the usual one.

$\text{proj}_1((x, y)) \rightarrow x$, $\text{proj}_2((x, y)) \rightarrow y$, $\text{dec}(\text{enc}(x, y), z) \rightarrow x$

We will denote the associated normal form by $\downarrow_\ell$.

We have that:

$R = R_0 \rightarrow_{p_1} R_1 \rightarrow_{p_2} \cdots \rightarrow_{p_k} R_n = R'$

Assume by contradiction that there exists a first index $i$ such that $R_i\phi \downarrow \neq R_{i+1}\phi \downarrow$. So the reduction at position $p_{i+1}$ is not possible in $R_i\phi$. Therefore $R_i\phi \downarrow_{p_{i+1}} = \text{dec}(\text{enc}(t_1, t_2), t_3)$ for some $t_1, t_2, t_3$, as projections pass in $R_i\phi$ whenever they pass in $R_i$. As $R_i\phi \downarrow \neq R_{i+1}\phi \downarrow$, $t_2 \downarrow \neq t_3 \downarrow$. So $\text{dec}(\text{enc}(t_1, t_2), t_3)$
will never reduce. We show that it will never be deleted by a reduction occuring above.

More precisely, we show by induction on $n$ the following property. For every $n$, if $R_i \phi \to^n t$ (for the lazy evaluation, i.e. the one w.r.t. $\downarrow$ semantics), then

1) $p_{i+1}$ is a path of $t$, $t[p_{i+1}] = \text{dec} (\text{enc}(t_1, t_2, t_3)) \downarrow$ and $\text{seq}_{p_{i+1}}(t) = \text{seq}_{p_{i+1}}(R_i)$;

2) for any $p \leq_{\text{lex}} p_{i+1}$ such that $p$ is a path of $t$, if $t[p] \downarrow$ is headed with a constructor symbol then
   - either $p$ is a path of $R_i$ and $\text{seq}_p(t) = \text{seq}_p(R_i)$,
   - or $t[p]$ is a subterm of a term of $\text{img}(\phi)$.

The case $n = 0$ is straightforward. Consider a new term $t'$ such that $R_i \phi \to^n t \to t'$. By induction hypothesis, the properties above hold for $t$. We consider the position $p'$ at which the reduction occurs in $t$.

- If $p' \geq_{\text{lex}} p_{i+1}$ and $p_{i+1}$ is not a prefix of $p'$, then the induction hypothesis is trivially preserved.
- If $p_{i+1}$ is a prefix of $p'$, then clearly, $t[p_{i+1}] = t[p'] = \text{dec} (\text{enc}(t_1, t_2, t_3)) \downarrow$ and the positions $p'' <_{\text{lex}} p_{i+1}$ are left unchanged.
- If $p' <_{\text{lex}} p_{i+1}$, we consider two cases. We have $t' = t[p'] = t[l] \theta$ for some $\theta$ and $l \to r$ is one of the rewrite rules.

1) either $p'$ is not a prefix of $p_{i+1}$, then property (1) is easily preserved for $t'$. Assume that $l = \text{dec} (\text{enc}(x, y, y))$. Since $p' <_{\text{lex}} p_{i+1}$, we must have $p' \leq_{\text{lex}} p_{i+1}$ (we may even note that $p' \leq_{\text{lex}} p_{i+1}$ since $p'$ is not a prefix of $p_{i+1}$). Since $t[p'] \downarrow$ is headed by $\text{enc}$, we have by induction hypothesis that
   - either $t[p']_1 = \text{enc}(t'_1, t'_2)$ is a subterm of a term of $\text{img}(\phi)$ and thus $t' = t[t'_1]$ satisfies property (2).
   - or $p' \downarrow$ is a path of $R_i$ and $\text{seq}_{p' \downarrow}(t) = \text{seq}_{p' \downarrow}(R_i)$, which means that $R_i$ could have been reduced at position $p'$, contradiction. (Remember that these reductions have been performed following the prefix order.)

The case of the projection rule is similar.

2) or $p'$ is a prefix of $p_{i+1}$. If $p' \downarrow$ is also a prefix of $p_{i+1}$, then $R_i$ could have been reduced at position $p'$ (as $\text{seq}_{p_{i+1}}(t) = \text{seq}_{p_{i+1}}(R_i)$), contradiction. Thus we must have that $l = \text{dec} (\text{enc}(x, y, y))$ and $p' \leq_{\text{lex}} p_{i+1}$.

Since $p' \downarrow \leq_{\text{lex}} p' \downarrow \leq_{\text{lex}} p_{i+1}$ and $t[p']_1 = \text{enc}(t_1, t_2)$, by induction hypothesis, we deduce that
   - either $t[p']_1 = \text{enc}(t'_1, t'_2)$ is a subterm of a term of $\text{img}(\phi)$. As $p' \downarrow$ is a prefix of $p_{i+1}$, $t[p']_1$ is a subterm of $t[p']_2$. Moreover, since there is a reduction $\text{dec}(\text{enc}(x, y, y)) \to x$ at position $p'$, we know that $t[p']_1 = t'_2 = t[p']_2$. Therefore, we have that $t[p']_1$ is a subterm of $t[p']_2$. (a constructor term). However, $t[p']_1$ contains a destructor since $\text{seq}_{p_{i+1}}(t) = \text{seq}_{p_{i+1}}(R_i)$ (by induction hypothesis, item 1). Hence, we obtain a contradiction.
   - or $p' \downarrow$ is a path of $R_i$ and $\text{seq}_{p' \downarrow}(t) = \text{seq}_{p' \downarrow}(R_i)$, which means that $R_i$ could have been reduced at position $p'$, contradiction.

Thanks to item 1, we have that $[R_i \phi]_p \downarrow = \text{dec}(\text{enc}(t_1, t_2, t_3)) \downarrow$ is a subterm of $R_i \phi$, which is not a message. It is what we wanted to prove.

**Lemma 6.** Let $\phi$ and $\psi$ be two frames. We have that $\phi \subseteq \psi$ if, and only if, $\phi$ is statically included in $\psi$ w.r.t. the semantics $\downarrow$.

**Proof.** We show the two directions separately.

($\Rightarrow$). Let $\phi$ and $\psi$ be two frames such that $\phi \subseteq \psi$ w.r.t. the semantics $\downarrow$. We have to show that $\phi \subseteq \psi$. We show this result by induction on the number of steps to make the recipe $R$ (or the test $R_1 = R_2$) in normal form w.r.t. the rules below (and considering an innermost derivation). Note that the last rule is not the usual one.

\[
\text{proj}_1((x, y)) \to x, \ \text{proj}_2((x, y)) \to y, \ \text{dec}(\text{enc}(x, y), z) \to x
\]

**Base case:** $R$ (or $R_1, R_2$) is without detour.

- Let $R$ be a recipe without detour such that $R \phi \downarrow \in M_{\Sigma_0}$. We have also that $R \phi \downarrow \in M_{\Sigma_0}$, and thanks to our hypothesis, we know that $R \psi \downarrow \in M_{\Sigma_0}$. Applying Lemma 3, we deduce that $R \psi \downarrow \in M_{\Sigma_0}$.

- Let $R_1, R_2$ be two recipes without detour such that $R_1 \phi \downarrow, R_2 \phi \downarrow \in M_{\Sigma_0}$, and $R_1 \phi \downarrow = R_2 \phi \downarrow$. We have also that $R_1 \psi \downarrow, R_2 \psi \downarrow \in M_{\Sigma_0}$, and $R_1 \phi \downarrow = R_2 \psi \downarrow$. Since $\phi \subseteq \psi$ w.r.t. the semantics $\downarrow$, we deduce that $R_1 \psi \downarrow, R_2 \psi \downarrow \in M_{\Sigma_0}$, and $R_1 \psi \downarrow = R_2 \psi \downarrow$. Applying Lemma 3, we deduce that $R_1 \psi \downarrow, R_2 \psi \downarrow \in M_{\Sigma_0}$ and therefore $R_1 \psi \downarrow = R_2 \psi \downarrow$.

**Inductive step.**

- Let $R$ be a recipe that contains $n$ detours such that $R \phi \downarrow \in M_{\Sigma_n}$. We assume w.l.o.g. that $R = C[\text{dec}(\text{enc}(R_1, R_2), R_3)]$, and $R_1, R_2, R_3$ are without detour. Let $R' = C[R_1]$. By definition of the rewriting, we have that $R_2 \psi \downarrow, R_3 \psi \downarrow \in M_{\Sigma_n}$, and $R_2 \phi \downarrow = R_3 \phi \downarrow$. We have also that $\text{enc}(R_1 \phi \downarrow, R_1 \psi \downarrow) \in M_{\Sigma_n}$. Since $R_2, R_3$ and $\text{enc}(R_1, R_2)$ are without detour, we know that $R_2 \psi \downarrow, R_3 \psi \downarrow \in M_{\Sigma_n}$, $R_2 \psi \downarrow = R_3 \psi \downarrow$, and $\text{enc}(R_1, R_2) \psi \downarrow \in M_{\Sigma_n}$. Moreover, we have that $R' \phi \downarrow \in M_{\Sigma_n}$, and applying the induction hypothesis, we deduce that $R' \psi \downarrow \in M_{\Sigma_n}$, therefore we deduce that

\[
R \psi \downarrow = C[\text{dec}(\text{enc}(R_1 \psi \downarrow, R_2 \psi \downarrow), R_3 \psi \downarrow)] \downarrow = C[R_1 \psi \downarrow] \downarrow = R' \psi \downarrow \in M_{\Sigma_n}
\]

- Let $R_1, R_2$ be two recipes that contains $n_1 + n_2$ detours and such that $R_1 \phi \downarrow, R_2 \phi \downarrow \in M_{\Sigma_n}$, and $R_1 \phi \downarrow = R_2 \phi \downarrow$. Moreover, we assume w.l.o.g. that $R_1 = C[\text{dec}(\text{enc}(R_a, R_b), R_c)]$ and $R_a, R_b, R_c$ are recipes without detour. Let $R'_1 = C[R_a]$. By definition of the rewriting, we have that $R_a \phi \downarrow, R_b \phi \downarrow \in M_{\Sigma_n}$, and $R_a \phi \downarrow = R_c \phi \downarrow$. We have also that $\text{enc}(R_a \phi \downarrow, R_c \phi \downarrow) \in M_{\Sigma_n}$, and $R_a \phi \downarrow = R_c \phi \downarrow$.
$M_{\Sigma_0}$. Since $R_b, R_c$ and $\text{enc}(R_a, R_b)$ are without detour, we know that $R_b \psi \downarrow, R_c \psi \downarrow \in M_{\Sigma_0}$, $R_b \psi \downarrow = R_c \psi \downarrow$, and $\text{enc}(R_a, R_b) \psi \downarrow \in M_{\Sigma_0}$. Moreover, we have that $R_1' \phi \downarrow, R_2 \phi \downarrow \in M_{\Sigma_0}$, and $R_1' \phi \downarrow = R_2 \phi \downarrow$. Applying our induction hypothesis, we deduce that $R_1' \psi \downarrow, R_2 \psi \downarrow \in M_{\Sigma_0}$, and $R_1' \psi \downarrow = R_2 \psi \downarrow$. Therefore, we deduce that $R_1 \psi \downarrow, R_2 \psi \downarrow \in M_{\Sigma_0}$, and $R_1 \psi \downarrow = R_2 \psi \downarrow$.

This allows us to conclude that $\phi \subseteq_s \psi$ w.r.t. the semantics $\downarrow$ implies that $\phi \subseteq_s \psi$.

$(\Rightarrow)$ Let $\phi$ and $\psi$ be two frames such that $\phi \subseteq_s \psi$. We have to show that $\phi \subseteq_s \psi$ w.r.t. the semantics $\downarrow$. We show this result (or more precisely a slightly stronger result) by induction on the number of steps to make the recipe $R$ (or the test $R_1 = R_2$) in normal form w.r.t. the rules below (and considering an outermost derivation). Note that the last rule is not the usual one.

Let $\psi = \text{proj}_1((x, y)) \rightarrow x$, $\text{proj}_2((x, y)) \rightarrow y$, $\text{dec}(\text{enc}(x, y), z) \rightarrow x$.

Actually, we transfer all the tests even those that do not reduce to messages.

**Base case:** $R$ (or $R_1, R_2$) is without detour.

- Let $R$ be a recipe without detour such that $R \phi \downarrow \in M_{\Sigma_0}$. Thanks to Lemma 3, we have that $R \phi \downarrow \in M_{\Sigma_0}$. Since $\phi \subseteq_s \psi$, we deduce that $R \psi \downarrow \in M_{\Sigma_0}$, and therefore $R \psi \downarrow \in M_{\Sigma_0}$.

- Let $R_1, R_2$ be two recipes without detour such that $R_1 \phi \downarrow, R_2 \phi \downarrow \in M_{\Sigma_0}$, and $R_1 \phi \downarrow = R_2 \phi \downarrow$. Thanks to Lemma 3, we have that $R_1 \psi \downarrow, R_2 \psi \downarrow \in M_{\Sigma_0}$, and $R_1 \psi \downarrow = R_2 \psi \downarrow$. Since $\phi \subseteq_s \psi$, we deduce that $R_1 \psi \downarrow, R_2 \psi \downarrow \in M_{\Sigma_0}$, and $R_1 \psi \downarrow = R_2 \psi \downarrow$. Therefore, we conclude that $R_1 \psi \downarrow, R_2 \psi \downarrow \in M_{\Sigma_0}$, and $R_1 \psi \downarrow = R_2 \psi \downarrow$.

- Let $R_1, R_2$ be two recipes without detour such that $R_1 \phi \downarrow, R_2 \phi \downarrow \in M_{\Sigma_0}$, and $R_1 \phi \downarrow = R_2 \phi \downarrow$ are not both in $M_{\Sigma_0}$). We show that $R_1 \psi \downarrow = R_2 \psi \downarrow$ by induction on $|R_1| + |R_2|$, i.e. number of symbols in $R_1$ and $R_2$. When $R_1 \phi \downarrow, R_2 \phi \downarrow \in M_{\Sigma_0}$, we have already shown the result, therefore we consider the case where $R_1 \phi \downarrow, R_2 \phi \downarrow \notin M_{\Sigma_0}$. Applying Lemma 4 on $R_1, R_2$, we deduce that we are in one of the following cases:

1) $R_1 = f(R_1', R_1')$, $R_2 = f(R_2', R_2')$ for some $R_1', R_1', R_2', R_2', f \in \{\text{enc}, \text{dec}, \langle\rangle\}$. Moreover, we have that $R_1 \phi \downarrow = R_2 \phi \downarrow = f(u', u'')$ for some $u', u''$ such that $R_1' \phi \downarrow = R_2' \phi \downarrow = u'$, and $R_1' \phi \downarrow = R_2' \phi \downarrow = u''$.

2) $R_1 = \text{proj}_1(R_1'), R_2 = \text{proj}_1(R_2')$ for some $R_1', R_2'$. Moreover, we have that $R_1 \phi \downarrow = R_2 \phi \downarrow = \text{proj}_1(u')$ for some $u'$ such that $R_1' \phi \downarrow = R_2' \phi \downarrow = u'$. We apply our induction hypothesis on the test $R_1' = R_2'$ (and $R_1' = R_2'$). This allows us to conclude that $R_1' \psi \downarrow = R_2' \psi \downarrow$ (and $R_1' \psi \downarrow = R_2' \psi \downarrow$), and therefore conclude that $R_1 \psi \downarrow = f(R_1' \psi \downarrow, R_1' \psi \downarrow) \downarrow = f(R_2' \psi \downarrow, R_2' \psi \downarrow) \downarrow = R_2 \psi \downarrow$.

**Inductive step.**

- Let $R$ be a recipe that contains $n$ detours such that $R \phi \downarrow \in M_{\Sigma_0}$. We assume w.l.o.g. that $R = C[\text{dec}(\text{enc}(R_1, R_2), R_3)]$, and this pattern corresponds to the outermost detour. Let $R' = C[R_1]$. By definition of the rewriting, we have that $R_2 \phi \downarrow = R_3 \phi \downarrow$ (even if we do not know whether they are messages or not). Applying our induction hypothesis, we know that $R_2 \psi \downarrow = R_3 \psi \downarrow$, and $R' \psi \downarrow \in M_{\Sigma_0}$. Therefore, we deduce that $R \psi \downarrow = C[\text{dec}(R_1 \psi \downarrow, R_2 \psi \downarrow), R_3 \psi \downarrow)] \downarrow = C[R_1 \psi \downarrow] \downarrow = R' \psi \downarrow \in M_{\Sigma_0}$.

- Let $R_1, R_2$ be two recipes that contain $n_1 + n_2$ detours and such that $R_1 \phi \downarrow = R_2 \phi \downarrow$. We assume w.l.o.g. that $R = C[\text{dec}(\text{enc}(R_4, R_5), R_6)]$, and this pattern corresponds to the outermost detour. Let $R_1' = C[R_a]$. By definition of the rewriting, we have that $R_1 \phi \downarrow = R_c \phi \downarrow$ (even if we do not know whether they are messages or not). Applying our induction hypothesis, we know that $R_1 \psi \downarrow = R_2 \psi \downarrow$. Therefore, we deduce that $R_1 \psi \downarrow = R_2 \psi \downarrow$.

This allows us to conclude that $\phi \subseteq_s \psi$ implies that $\phi \subseteq_s \psi$ w.r.t. the semantics $\downarrow$. 

**Theorem 1:** Let $\mathcal{P}$ be a protocol type-compliant w.r.t. ($\mathcal{P}, \delta_\mathcal{P}$) and $\mathcal{Q}$ be another protocol. We have that $\mathcal{P} \not\models \mathcal{Q}$ w.r.t. $\Sigma_0$ if, and only if, there exists a witness $\tau$ of this non-inclusion that only contains simple recipes and such that one of the following holds:

1) $(\tau, \phi) \in \text{trace}_{\Sigma_0}(\mathcal{P})$ for some $\phi$ and $(\tau, \phi)$ is well-typed w.r.t. $(\mathcal{T}_\mathcal{P}, \delta_\mathcal{P})$;

2) $\tau = \text{tr}^c(c_0 \mapsto c_{(\omega, \omega)})$ for some $c_0 \in \Sigma_0 \setminus \Sigma_P$, and $(\tau', \phi') \in \text{trace}_{\Sigma_0}(\mathcal{P})$ for some $\phi'$ and $(\tau', \phi')$ is well-typed w.r.t. $(\mathcal{T}_\mathcal{P}, \delta_\mathcal{P})$.

**Proof.** We have that $\mathcal{P} \not\models \mathcal{Q}$ w.r.t. $\Sigma_0$, and therefore there exists a witness $(\tau, \phi)$ of this non-inclusion. We have that $(\tau, \phi) \in \text{trace}_{\Sigma_0}(\mathcal{P})$ and:

1) either $(\tau, \psi) \notin \text{trace}_{\Sigma_0}(\mathcal{Q})$ for any frame $\psi$;

2) or $(\tau, \psi) \in \text{trace}_{\Sigma_0}(\mathcal{Q})$ for some $\psi$ but $\phi \subseteq_s \psi$ w.r.t. $\Sigma_0$.

Note that, due to the fact that we consider simple protocols, the frame $\psi$ is uniquely defined when it exists. Moreover, we chose $\tau$ minimal in length. We first establish that $\mathcal{P}$ is not included in $\mathcal{Q}$ w.r.t. the semantics $\downarrow$ for terms. We distinguish the two cases we mentioned above. Regarding case 2, we have that $(\tau, \phi)$ (resp. $(\tau, \psi)$) is a trace of $\mathcal{P}$ (resp. $\mathcal{Q}$) w.r.t. the semantics $\downarrow$, and therefore we conclude thanks to Lemma 6. Regarding case 1, we have that $(\tau, \phi)$ is a trace of $\mathcal{P}$ w.r.t. the semantics $\downarrow$, and we know also that $\tau^{-1}$ (i.e. $\tau$ without the last action) is a trace of $\mathcal{Q}$ w.r.t. the semantics $\downarrow$. Let $\alpha$ be such that $\tau = \tau^{-1} \cdot \alpha$. Since we know that $\tau = \tau^{-1} \cdot \alpha$ is not a trace of $\mathcal{Q}$ (w.r.t. the semantics $\downarrow$), the only case where $\tau$ can become a trace of $\mathcal{Q}$ w.r.t. the semantics $\downarrow$ is when $\alpha = \text{in}(c, R)$ with $R \psi \downarrow \in M_{\Sigma_0}$.
whereas $R\psi \downarrow \notin M_{S_0}$. However, we know that $\phi \subseteq_a \psi$ w.r.t. semantics $\downarrow$ (thanks to the minimality of our witness), and since we have $R\phi \downarrow \notin M_{S_0}$, we know that $R\phi \downarrow \notin M_{S_0}$. Therefore this case can not happen. We have shown that $P \nsubseteq Q$ w.r.t. semantics $\downarrow$.

Now, we can apply the typing result as stated and proved in [19], we deduce that there exists a witness $(tr, \phi)$ of this non-inclusion (w.r.t. the semantics $\downarrow$) such that one of the following holds:

1. $(tr, \phi)$ is a trace of $P$ (w.r.t. the semantics $\downarrow$) and $(tr, \phi)$ is well-typed w.r.t. $(T_P, \delta_P)$; or
2. $tr = tr'[c_0 \mapsto c_{(\omega, \omega)}]$ for some $c_0 \in \Sigma_0 \setminus \Sigma_P$, and some $(tr', \phi')$ a trace of $P$ (w.r.t. the semantics $\downarrow$) for some $\phi'$, and $(tr', \phi')$ is well-typed w.r.t. $(T_P, \delta_P)$.

We consider among these witnesses one that is of minimal length. We have that $(tr, \phi) \in \text{trace}(P)$ (w.r.t. the semantics $\downarrow$) and:

1. either $(tr, \psi) \notin \text{trace}(Q)$ (w.r.t. the semantics $\downarrow$) for any frame $\psi$;
2. or $(tr, \psi) \in \text{trace}(Q)$ (w.r.t. the semantics $\downarrow$) for some $\psi$ but $\phi \subseteq_a \psi$ (w.r.t. the semantics $\downarrow$).

Regarding case 2, we apply Lemma 5 and we consider $\mathbf{\mathfrak{T}}$ a trace made of recipes without detours such that $\mathbf{\mathfrak{T}}\phi \downarrow = tr\phi \downarrow$, and $\mathbf{\mathfrak{T}}\psi \downarrow = tr\psi \downarrow$. Thanks to Lemma 3, the resulting trace is made of simple recipes. We have that $\phi \subseteq_a \psi$ (w.r.t. the semantics $\downarrow$) thanks to Lemma 6. Regarding case 1, we have to show that $\mathbf{\mathfrak{T}}$ is not a trace of $Q$ w.r.t. the semantics $\downarrow$. We know that $tr = tr^{-1} \cdot \alpha$, and in such a case $\alpha = in(c, R)$ whereas its counterpart in $\mathbf{\mathfrak{T}}$ is $in(c, R')$. We know that $R\phi \downarrow = \mathbf{\mathfrak{T}}\phi \downarrow \in M_{S_0}$, and therefore $R\psi \downarrow = \mathbf{\mathfrak{T}}\psi \downarrow \in M_{S_0}$ since otherwise we would contradict the minimality of our witness. Therefore, $\mathbf{\mathfrak{T}}$ is not a trace of $Q$ w.r.t. the semantics $\downarrow$, and thanks to Lemma 3, we deduce that $\mathbf{\mathfrak{T}}$ is not a trace of $Q$ w.r.t. the semantics $\downarrow$.

Note that $tr\phi \downarrow = \mathbf{\mathfrak{T}}\phi \downarrow$, and since $tr$ satisfies the requirements regarding typing, we easily deduce that $(\mathbf{\mathfrak{T}}, \phi)$ satisfies the requirements regarding typing. \qed

**B. Bounding the number of constants**

First, we establish the following result.

**Proposition 2:** Let $P$ be a protocol type-compliant w.r.t. $(T_P, \delta_P)$ and $Q$ be another protocol. Let $\Sigma = \Sigma_P \cup \{c \in \Sigma_0 \mid \delta_0(c) = \tau_c\}$. We have that $P \nsubseteq Q$ w.r.t. $\Sigma_0$ if, and only if, there exists a witness $tr$ of this non-inclusion w.r.t. $\Sigma$ that only contains simple recipes, and such that $(tr, \phi) \in \text{trace}_\Sigma(P)$ for some $\phi$ and $(tr, \phi)$ is quasi-well-typed w.r.t. $(T_P, \delta_P)$.

**Proof.** First note that the converse is obvious: if there is any kind of witness of non-inclusion, then $P \nsubseteq Q$ w.r.t. $\Sigma_0$. So we only prove the direct part.

Assume that $P \nsubseteq Q$ w.r.t. $\Sigma_0$. By Theorem 1, it implies that there exists a witness $tr$ of this non-inclusion that only contains simple recipes and such that one of the following holds:

1. $(tr, \phi) \in \text{trace}_\Sigma(P)$ for some $\phi$ and $(tr, \phi)$ is well-typed w.r.t. $(T_P, \delta_P)$;
2. $tr = tr'[c_0 \mapsto c_{(\omega, \omega)}]$ for some $c_0 \in \Sigma_0 \setminus \Sigma_P$, and $(tr', \phi') \in \text{trace}_\Sigma(P)$ for some $\phi'$ and $(tr', \phi')$ is well-typed w.r.t. $(T_P, \delta_P)$.

Assume that one of those conditions holds. In each case, $tr$ is a trace w.r.t. $\Sigma_0$ quasi-well-typed w.r.t. $(T_P, \delta_P)$, as $c_{(\omega, \omega)}$ has type $\tau_c$ (the smallest for $\geq$). There is therefore a finite number of public constants of $\Sigma_0 \setminus \Sigma_P$ occurring in $(tr, \phi)$. Call them $\{c_1, \ldots, c_n\}$. As $\{c \in \Sigma_0 \setminus \Sigma_Q \mid \delta(c) = \tau_c\}$ is infinite, we choose $c_1', \ldots, c_n'$ in this set (different from $c_{(\omega, \omega)}$) and define a substitution $\sigma$ such that $\sigma(c_i) = c_i'$ for each $i$. As $c_1, \ldots, c_n$ do not occur in $P$, $(tr, \sigma)$ is a trace of $P$ w.r.t. $\Sigma$. If $(tr, \phi)$ was not a trace of $Q$, then $(tr, \sigma, \phi)$ is still not because $c_1', \ldots, c_n'$ do not occur in $Q$. If $(tr, \psi)$ was a trace of $Q$, it passes in $Q$ with frame $\psi$. As $\sigma$ is a bijective substitution, $R(\sigma, \phi)$ is a message if and only if $(R^0(\sigma, \phi))$ is. So if there is a message $R$ such that $R\phi \downarrow$ is a message, but $R\psi \downarrow$ is not, then $(R(\sigma))\phi \downarrow$ is a message, but $(R(\sigma))\psi \downarrow$ is not.

The equality case is similar, so $tr\sigma$ is still a witness of non-inclusion. \qed

Note that we still have an unbounded number of constants to consider. We have to consider constants that occur in $P$ and in addition an unbounded number of constants of type $\tau_c$. In the following, we show that it is actually sufficient to consider two constants of type $\tau_c$. To prove Theorem 2, we have to state and prove some useful properties about renamings.

Given a set of atomic data $A$, an $A$-renaming is a function $\rho$ such that $\text{dom}(\rho) \cup \text{img}(\rho) \subseteq A$.

**Lemma 7:** Let $\Sigma \subseteq \Sigma_0$, and $t, t'$ be two terms in $T(\Sigma_{\text{std}}, \Sigma \cup N)$.

1. If $t \downarrow \in M_\Sigma$, then $(t \downarrow)\rho = (t' \downarrow)$ for any $\Sigma$-renaming $\rho$.
2. If $t \downarrow \notin M_\Sigma$, then there exists $c_0 \in \Sigma$ such that for any $\Sigma$-renaming $\rho$ such that $c_0 \notin \text{dom}(\rho) \cup \text{img}(\rho)$, we have that $t\rho \downarrow \notin M_\Sigma$.
3. If $t \downarrow, t' \downarrow \in M_\Sigma$ and $t \downarrow \neq t' \downarrow$, there exists $c_0 \in \Sigma$ such that for any $\Sigma$-renaming $\rho$ such that $c_0 \notin \text{dom}(\rho) \cup \text{img}(\rho)$, we have that $t\rho \downarrow \neq t' \rho \downarrow$.

**Proof.** We prove the three items separately.

**Item 1.** Let $t \in T(\Sigma_{\text{std}}, \Sigma \cup N)$ such that $t \downarrow \in M_\Sigma$. We show the result by structural induction on $t$.

**Base case:** $t \in \Sigma \cup N$. In such a case, we have that $(t \downarrow)\rho = (t' \downarrow)$.

**Inductive case:** In such a case, $t = f(t_1, t_2)$ with $f \in \{\text{enc}, \text{dec}, (\cdot), \text{proj}_1, \text{proj}_2\}$.

- Case $f = \text{enc}$. In such a case, we have that $t = \text{enc}(t_1, t_2)$, and $t_1 = \text{enc}(t_1, t_2')$. Therefore, we know that $t_1 \downarrow, t_2 \downarrow \in M_\Sigma$, and we conclude relying on our
induction hypothesis:

\[ (t)\rho = \begin{cases} \text{enc}(t_1\downarrow, t_2\downarrow)\rho & \text{if } t \rho \neq \ldots \in M_{\Sigma}, \\
\text{enc}(t_1\downarrow, \rho, (t_2)\rho) & \text{is not } \ldots \in M_{\Sigma}, \\
\text{enc}(t_1\rho) = (t_2\rho) & \text{is not } \ldots \in M_{\Sigma}, \\
\text{enc}(t_1\rho, \rho) & \text{is not } \ldots \in M_{\Sigma}, \\
(t)\rho & \text{is not } \ldots \in M_{\Sigma}. 
\end{cases} \]

Moreover, we know that \( v \neq v' \). In case \( v \) or \( v' \) is not a constant in \( \Sigma \), we still have that \( v \rho \neq v' \rho \) for any \( \Sigma \)-renaming \( \rho \), and therefore we are done: \( (t)\rho \notin M_{\Sigma}. \)

In case, \( v, v' \) are both in \( \Sigma \), let \( c_0 \) be the constant \( v' \), and consider any renaming \( \rho \) such that \( c_0 \notin \text{dom}(\rho) \cup \text{img}(\rho) \).

We have that \( v \rho \neq c_0 \), and thus \( v \rho \neq v' \rho \), and this allows us to conclude.

- Case \( f = \text{proj}_1 \) (or \( \text{proj}_2 \)). In such a case, we have that \( t = \text{proj}_1(t') \) for some \( t' \). In case \( t' \notin M_{\Sigma} \), we conclude by applying our induction hypothesis. Now, assuming that \( t' \notin M_{\Sigma} \), we know that \( t' \) is not of the form \( \langle u, v \rangle \) (since otherwise \( t \notin M_{\Sigma} \)). Thanks to Item 1, we have that \( (t')\rho = (t')\rho \notin M_{\Sigma} \). Therefore, we know that \( (t')\rho \) is not of the form \( \langle u', v' \rangle \) for any \( \Sigma \)-renaming \( \rho \), and thus \( t \rho \notin M_{\Sigma} \) for any \( \Sigma \)-renaming \( \rho \).

Item 3. Let \( t_1, t_2 \in T(\Sigma_{\text{std}}, \Sigma \cup N) \) such that \( t_1 \notin M_{\Sigma} \) and \( t_2 \notin M_{\Sigma} \). Thanks to Item 1, we have that \( (t_1)\rho = (t_1)\rho \) and \( (t_2)\rho = (t_2)\rho \). Therefore, we can simply show that if \( t_1, t_2 \notin M_{\Sigma} \) and \( t_1 \neq t_2 \) then there exists \( c_0 \in \Sigma \) such that \( t_1\rho \neq t_2\rho \) for any \( \Sigma \)-renaming \( \rho \) such that \( c_0 \notin \text{dom}(\rho) \cup \text{img}(\rho) \).

Base case: The only non trivial base case is when both \( t_1 \) and \( t_2 \) are in \( \Sigma \). In such a case, let \( c_0 \) be \( 2 \). Clearly, since \( t_1 \neq t_2 \), we have that \( t_1\rho \neq t_2\rho \) for any \( \Sigma \)-renaming \( \rho \) such that \( c_0 \notin \text{dom}(\rho) \cup \text{img}(\rho) \).

Other base cases where either \( t_1 \) or \( t_2 \) is in \( \Sigma \cup N \) are trivial and we may actually choose any \( \Sigma \)-renaming \( \rho \).

Inductive case: Now, in case \( t_1 \) and \( t_2 \) are not atomic, we distinguish two cases. In case they do not have the same function symbol has their root, again we can actually choose any \( \Sigma \)-renaming \( \rho \), and the disequality between \( t_1\rho \) and \( t_2\rho \) will be preserved. Now, assume that \( t_1 = f(u_1, v_1) \) and \( t_2 = f(u_2, v_2) \) with \( f \in \{\text{enc}, \langle \rangle\} \). We know that either \( u_1 \neq u_2 \) or \( v_1 \neq v_2 \). Assume w.l.o.g. that \( u_1 \neq u_2 \). We can apply our induction hypothesis to conclude that there exists \( c_0 \) such that \( u_1\rho \neq u_2\rho \) for any \( \Sigma \)-renaming \( \rho \) such that \( c_0 \notin \text{dom}(\rho) \cup \text{img}(\rho) \).

Considering any \( \Sigma \)-renaming that satisfies such a condition will allow us to conclude.

\[ \Box \]

Lemma 8: Let \( \Sigma \subseteq \Sigma_0 \), \( m \in M_{\Sigma} \), and \( u \) be a term in \( T(\Sigma_\epsilon, \Sigma \cup N \cup \Sigma_0) \). If \( f \neq \text{u}\sigma \) for any substitution \( \sigma \), then there exists \( c_0 \in \Sigma \) such that for any \( \Sigma \)-renaming \( \rho \) such that \( c_0 \notin \text{dom}(\rho) \cup \text{img}(\rho) \), we have that \( m\rho \neq (u\rho)\sigma \) for any \( \sigma \).

Proof. If \( m \) and \( u \) do not match because their structure differ, then no renaming will change that. Else, the only possibility is that there are two leaves of \( u \) that have the same variable, but \( m \) has different subtrees \( t_1 \neq t_2 \) at those positions. Thanks to Lemma 7 (item 3), there is a constant \( c_0 \) such that for any \( \Sigma_0 \)-renaming \( \rho \) with \( c_0 \notin \text{dom}(\rho) \cup \text{img}(\rho) \), \( t_1\rho \neq t_2\rho \).

For such a \( \rho \), \( m\rho \) does not unify with \( u\rho \).

We say that a renaming \( \rho \) is type-preserving if for any \( a \in \text{dom}(\rho) \) we have \( \delta(a\rho) = \delta(a) \).

Lemma 9: Let \( \mathcal{P} \) be a protocol type-compliant w.r.t. \( (T_\mathcal{P}, \delta_\mathcal{P}) \) and \( \rho \) be a \( \Sigma \)-renaming where \( \Sigma = \Sigma_\epsilon \setminus \Sigma_\mathcal{P} \). Let
(P'; φ'; σ') be a configuration such that $P \xrightarrow{u} (P'; φ'; σ')$ for some tr. We have that $P \xrightarrow{u} (P'; φ'; σ')$.

Moreover, when $ρ$ is type-preserving and (tr, $φ'$) is quasi-well-typed w.r.t. $(T, δ)$, then (tr, $φ'$) is quasi-well-typed w.r.t. $(T, δ)$.

Proof. We show this result by induction on the length $ℓ$ of the execution trace $(P; φ; 0) \xrightarrow{u^ℓ} (P'; φ'; σ')$.

Base case. In case $ℓ = 0$, the result trivially holds.

Induction case. In such a case, we have that:

$$P \xrightarrow{u^ℓ} (P''; φ''; σ'') \xrightarrow{ρ} (P'; φ'; σ')$$

Thanks to our induction hypothesis, we have that $P \xrightarrow{u^ℓ ρ} (P''; φ''ρ; σ'')$.

We distinguish two cases depending on the action $α$.

- $α = \text{in}(c, R)$. In such a case, we have that $P'' = \text{in}(c, u).P_0 + P_0$ for some $u$, $P_0$, and $P_0$. We have also that $φ' = φ''$, and $σ' = σ'' ω ∥ ω$ for some $σ_0$ such that $Rφ''_R = (uφ''_R)σ_0$ and $Rφ''_R ∈ M_Σ$. To conclude that $(P''; φ''ρ; σ''ρ) \xrightarrow{αρ} (P'; φ'; ρ; σ')$, it remains to show that $(Rφ''_φ)ρ = u(σ''ρ)αρ$ and $σρ = σ''ρ τ ω$ for some $σ_0$. Since $Rφ''_φ = (uφ''_φ)σ_0$, we deduce that $(Rφ''_φ)ρ = (uσ''ρ)σ_0$ and thanks to Lemma 7 (item 1), we have that $(Rφ''_φ)ρ = (u(σ''ρ))σ_0ρ$. Finally, since $ρ$ is a $Σ$-renaming and $Σ = Σ_0 \ wedge Σ_r$, we know that $uρ = u$, and therefore we have that $(Rφ''_φ)ρ = (u(σ''ρ))σ_0ρ$. Moreover, since $σ' = σ'' ω ∥ ω$, we have that $σρ = σ''ρ ω ∥ ω$. Therefore, choosing $σ''ρ = σ_0ρ$ allows us to conclude.

- $α = \text{out}(c, w)$. In such a case, we have that $P'' = \text{out}(c, u).P_0 + P_0$ for some $u$, $P_0$, and $P_0$. We have also that $σ' = σ''$, and $φ' = φ'' ∪ \{w → uσ''\}$. To conclude that $(P''; φ''ρ; σ''ρ) \xrightarrow{αρ} (P'; φ'; ρ; σ')$, it is sufficient to show that $(uσ''ρ) = u(σ''ρ)$. Actually, we have that $(uσ''ρ) = u(σ''ρ)$, and since $Σ = Σ_0 \ wedge Σ_r$, we deduce that $u = u$, and this allows us to conclude.

Note that $δ(σ''ρ) = δ(σ''ρ)$ since $ρ$ is type-preserving, and therefore the resulting trace is quasi-well-typed when (tr, $φ'$) is quasi-well-typed.

\[\Box\]

Theorem 2: Let $P$ be a protocol type-compliant w.r.t. $(T_P, δ_P)$ and $Q$ be another protocol. Let $Σ = Σ_ρ υ \{c_0, c_1, c_ω, c_ω\}$. We have that $P \not\succeq P$ w.r.t. $Σ_0$, and only if, there exists a witness tr of this non-inclusion w.r.t. $Σ_0$ that only contains simple recipes, and such that (tr, $φ$) is trace in $P$ for some $φ$ and (tr, $φ$) is quasi-well-typed w.r.t. $(T_P, δ_P)$.

Proof. One direction is trivial. Therefore, we focus on the other one. Let $Σ_ρ = Σ_ρ υ \{c ∈ Σ_0 | δ_0(c) = t_ρ\}$. Proposition 2 states that if $P \not\succeq P$ w.r.t. $Σ_ρ$ then there exists a witness tr of this non-inclusion w.r.t. $Σ_0$ that only contains simple recipes, and such that (tr, $φ$) is trace in $P$ for some $φ$ and (tr, $φ$) is quasi-well-typed w.r.t. $(T_P, δ_P)$.

Let tr be such a witness having a minimal length. To conclude, we establish the following claim.

Claim: There exists a type-preserving renaming $ρ$ with $dom(ρ) ⊆ Σ_0 \ wedge Σ_r$, and $im(ρ) ⊆ Σ_0 \ wedge Σ_r$ such that tr $ρ$ is a witness of the non-inclusion $P \not\succeq P$ w.r.t. $Σ_0$.

Since tr is a witness of minimal length, we have that:

1) either tr is not a trace of $Q$ but tr $^{-1}$ (i.e. tr without its last element) is a trace of $Q$;

2) or $Q \xrightarrow{α} (P'; ω; σ_0)$ for some $ψ$ (that is uniquely defined) but $φ \not\succeq ψ$.

Case 1): tr is not a trace of $Q$. In such a case, we have that tr = tr $0$ and we know that

$$P \xrightarrow{u} (P''; φ''; σ'') \xrightarrow{ρ} (P'; φ'; σ')$$

and also that $Q \xrightarrow{ρ} (Q_0; ω; σ_0)$. We distinguish several cases depending on the action $α$ and also the reason that prevents this step to be mimicked in $Q$.

1) $α = \text{in}(c, R)$, so there is in(c, u).P $0$ in $P_0$, but there is no process on channel c starting with an input in $Q_0$. Let $ρ_0$ be the renaming that maps any constant in $Σ_0 \ wedge Σ_r$ to the constant $c_0$. Thanks to Lemma 9, we have that

$$P \xrightarrow{ρ_0} (P''; φ''; σ'') \xrightarrow{ρ} (P'; φ'; σ')$$

and also that $Q \xrightarrow{ρ_0} (Q_0; ω; σ_0)$. There is still no process on channel c starting with an input in $Q_0$, and thus tr $ρ_0$ is the witness we are looking for.

2) $α = \text{in}(c, R)$ and there is in(c, u).P $0$ in $P_0$ as well as in(c, v).Q $0$ in $Q_0$. Since tr is minimal and we know that $Rφ''_φ = M_{Σ_0}$, we also have that $Rφ''_φ = M_{Σ_0}$. Therefore, we know that $Rφ''_φ ∩ σ_0$ does not unify with $vτ_0$. Thanks to Lemma 8, there is a constant c in $Rφ''_φ$ such that for any $ρ$ with $c \not\in dom(ρ) ∪ im(ρ)$, we have that $(Rφ''_φ)ρ$ does not unify with $vτ_0$. If $c \not\in Σ_0 \ wedge Σ_r$, then $ρ_0$ (as defined above) satisfies the requirement. Therefore, applying Lemma 9, we obtain that:

$$P \xrightarrow{ρ_0} (P''; φ''; σ'') \xrightarrow{ρ_0} (P'; φ'; σ')$$

and also that $Q \xrightarrow{ρ_0} (Q_0; ω; σ_0)$. We have also that $(Rφ''_φ)ρ_0$ does not unify with $vτ_0$, and we have that

$$(Rφ''_φ)ρ_0 = Rφ''_φ ∩ σ_0 = (Rφ''_φ)(vτ_0)$$

thanks to Lemma 7 (item 1). This allows us to conclude that this step can not be mimicked by $Q$.

If $c \not\in Σ_0 \ wedge Σ_r$, then up to a bijective $α$-renaming, we can assume that $c = c_1$. Let $ρ_1$ be the renaming that maps any constant in $Σ_0 \ wedge Σ_r$ on $c_1$ except $c_1$ that is left unchanged. Applying Lemma 9, we have that:

$$P \xrightarrow{ρ_1} (P''; φ''; σ'') \xrightarrow{ρ_1} (P'; φ'; σ')$$

and also that $Q \xrightarrow{ρ_1} (Q_0; ω; σ_0)$. We have also that $(Rφ''_φ)ρ_1$ does not unify with $vτ_1$, and we have that

$$(Rφ''_φ)ρ_1 = Rφ''_φ ∩ σ_1 = (Rφ''_φ)(vτ_1)$$
thanks to Lemma 7 (item 1). This allows us to conclude that this step can not be mimicked by \( Q \).

3) \( \alpha = \text{out}(c, w) \). In this case, either there is no corresponding output in \( Q_0 \), or there is a corresponding output, but the corresponding term is not a message. This can happen in case an encryption with a non-atomic key occurs in the outputted term. However, since renaming will not change that, we easily conclude considering \( p_0 \) as defined above.

Case 2): \( \operatorname{tr} \) is a trace of \( Q \). In such a case, we know that \( P \xrightarrow{\alpha} (P_0; \phi_0; \sigma_0) \) and \( Q \xrightarrow{\alpha} (Q_0; \psi_0; \tau_0) \) for some \( \phi_0 \) that is uniquely defined, and we have that \( \phi_0 \in \Sigma_s, \psi_0 \). Following the definition of static inclusion, we distinguish two cases:

1) There is a recipe \( W \) r.w.r.t. \( \Sigma_s \) such that \( R\phi_0 \downarrow \) is a message but \( R\psi_0 \downarrow \) is not. Then by Lemma 7, there is a constant \( c \) such that for any renaming \( p \) such that \( c \notin \text{dom}(p) \cup \text{img}(p) \), we have that \( R\psi_0 \downarrow \) is not a message.

If \( c \notin \Sigma_s \setminus \Sigma_p \), then \( p_0 \) (as defined above) is a renaming such that \( c \notin \text{dom}(p_0) \cup \text{img}(p_0) \). Thanks to Lemma 9, we have that \( P \xrightarrow{\alpha} (P_0; \phi_0; \sigma_0) \) and \( Q \xrightarrow{\alpha} (Q_0; \psi_0; \tau_0) \). Note that the only constants occurring in this execution are those of \( \Sigma \). Thanks to Lemma 7 (item 1), we have that \( (R\phi_0)(\phi_0\downarrow) = (R\phi_0)\downarrow = (R\phi_0)\downarrow \in M \). However, we have that \( (R\psi_0)(\psi_0\downarrow) = (R\psi_0)\downarrow \notin M \). Therefore, we have \( \phi_0 \notin \Sigma_s, \psi_0 \).

If \( c \in \Sigma_s \setminus \Sigma_p \), then up to a bijective \( \alpha \)-renaming, we can assume that \( c = c_1 \). Applying Lemma 9 with the renaming \( p_1 \) as defined above, we have that \( P \xrightarrow{\alpha} (P_0; \phi_0; \sigma_0) \) and \( Q \xrightarrow{\alpha} (Q_0; \psi_0; \tau_0) \). Note that the only constants occurring in this execution are those of \( \Sigma \). Thanks to Lemma 7 (item 1), we have that \( (R\phi_0)(\phi_0\downarrow) = (R\phi_0)\downarrow = (R\phi_0)\downarrow \in M \). However, we have that \( (R\psi_0)(\psi_0\downarrow) = (R\psi_0)\downarrow \notin M \). Therefore, we have that \( \phi_0 \notin \Sigma_s, \psi_0 \).

2) There are two recipes \( R_1 \) and \( R_2 \) r.w.r.t. \( \Sigma_s \) such that \( R_1\phi_0 \downarrow \) and \( R_2\phi_0 \downarrow \) are messages, and \( R_1\phi_0 \downarrow = R_2\phi_0 \downarrow \). We may also assume that \( R_1\phi_0 \downarrow \neq R_2\phi_0 \downarrow \). Then by Lemma 7 (item 3), there is a constant \( c \) such that for any renaming \( p \) such that \( c \notin \text{dom}(p) \cup \text{img}(p) \), we have that \( R_1\psi_0 \downarrow \neq R_1\psi_0 \downarrow \).

If \( c \notin \Sigma_s \setminus \Sigma_p \), then \( p_0 \) (as defined above) is a renaming such that \( c \notin \text{dom}(p_0) \cup \text{img}(p_0) \). Thanks to Lemma 9, we have that \( P \xrightarrow{\alpha} (P_0; \phi_0; \sigma_0) \) and \( Q \xrightarrow{\alpha} (Q_0; \psi_0; \tau_0) \). Note that the only constants occurring in this execution are those of \( \Sigma \). Thanks to Lemma 7 (item 1), we have that \( (R\phi_0)(\phi_0\downarrow) = (R\phi_0)\downarrow = (R\phi_0)\downarrow \in M \) for \( i = 1, 2 \), and \( (R\phi_0)(\phi_0\downarrow) = (R\phi_0)\downarrow = (R\phi_0)\downarrow \in M \) for \( i = 1, 2 \). Moreover, we have that \( R_1\phi_0 \downarrow \neq R_1\phi_0 \downarrow \) i.e. \( (R_1\phi_0)(\phi_0\downarrow) \neq (R_2\phi_0)(\phi_0\downarrow) \), whereas \( R_1\phi_0 \downarrow = R_2\phi_0 \downarrow \), i.e. \( (R_1\phi_0)(\phi_0\downarrow) = (R_2\phi_0)(\phi_0\downarrow) \). Hence, we have our witness of non-inclusion.

If \( c \in \Sigma_s \setminus \Sigma_p \), then up to a bijective \( \alpha \)-renaming, we can assume that \( c = c_1 \). Applying Lemma 9 with the renaming \( p_1 \) as defined above, we have that \( P \xrightarrow{\alpha} (P_0; \phi_0; \sigma_0) \) and \( Q \xrightarrow{\alpha} (Q_0; \psi_0; \tau_0) \). Note that the only constants occurring in this execution are those of \( \Sigma \). Thanks to Lemma 7 (item 1), we have that \( (R_1\phi_0)(\phi_0\downarrow) = (R_2\phi_0)(\phi_0\downarrow) \), whereas \( R_1\phi_0 \downarrow = R_2\phi_0 \downarrow \), i.e. \( (R_1\phi_0)(\phi_0\downarrow) = (R_2\phi_0)(\phi_0\downarrow) \). Hence, we have our witness of non-inclusion.

This allows us to establish the claim and therefore concludes the proof of our theorem.

\( \square \)

**APPENDIX B**

**FROM STATIC EQUIVALENCE TO PLANNING**

**Lemma 1.** Let \( \phi, \psi \) be two frames with \( \text{dom}(\phi) = \text{dom}(\psi) \). Let \( \Theta = (\text{Fact}, (\text{Fact}(\phi, \psi), \text{Concrete}^{2}(\text{Rule}_{A})) \) and \( \Pi = (\Theta, \{\text{at}(u, v)\}) \) for some \( u, v \in M_{\Sigma} \). We have that \( \Pi \) has a solution if, and only if, there is a destructor-only recipe \( R \in R_{\Sigma} \) such that \( R\phi \downarrow = u \), and \( R\psi \downarrow = v \).

**Proof.** We show the two directions separately.

\( \Rightarrow \) Let \( \pi = r_1, \ldots, r_n \) be a planning path from \( S_0 \) to \( S_n \), and \( \text{at}(u, v) \in S_n \). We show the result by induction on the length of \( \pi \).

**Base case.** We have that \( \pi \) is empty. In such a case, by definition of \( S_0 \), the result trivially holds.

**Inductive case.** We know that \( r_n \) is an instance of one of the abstract rules in \( \text{Rule}_{A} \), e.g.

\[ \text{at}(\text{enc}(u_1, u_2), \text{enc}(v_1, v_2)), \text{at}(u_2, v_2) \rightarrow \text{at}(u_1, v_1) \]

Thanks to our induction hypothesis, we know that there exist:

- \( R_1 \in R_{\Sigma} \) such that \( R_1\phi \downarrow = \text{enc}(u_1, u_2) \), and \( R_1\psi \downarrow = \text{enc}(v_1, v_2) \);
- \( R_2 \in R_{\Sigma} \) such that \( R_2\phi \downarrow = u_2 \), and \( R_2\psi \downarrow = v_2 \).

Therefore, the recipe \( R = \text{dec}(R_1, R_2) \) allows us to conclude.

\( \Leftarrow \) Let \( R \in R_{\Sigma} \) be a destructor-only recipe such that \( R\phi \downarrow = u \) and \( R\psi \downarrow = v \). We show the result by structural induction on \( R \).

**Base case.** We have that \( R \) is either \( w \in \text{dom}(\phi) \) or a constant in \( \Sigma \). In both cases, by definition of \( S_0 \), the empty path allows us to conclude.

**Inductive case.** In such a case, we have that \( R = \text{dec}(R_1, R_2) \) or \( R = \text{proj}_{i}(R') \) with \( i \in \{1, 2\} \). We assume that \( R = \text{dec}(R_1, R_2) \). Since we know that \( \text{R}\phi \downarrow \) and \( \text{R}\psi \downarrow \) are messages, we have that:

- \( R_1\phi \downarrow \) and \( R_2\phi \downarrow \) are messages of the form \( \text{enc}(u_1, u_2) \) and \( u_2 \) for some terms \( u_1, u_2 \);
• $R_1 \psi \downarrow$ and $R_2 \psi \downarrow$ are messages of the form $\text{enc}(v_1, v_2)$ and $v_2$ for some terms $v_1, v_2$.

Thanks to our induction hypothesis, we know that there exists $\pi_1$ a planning path from $S_0$ to $S_1$ with $\text{att}(R_1 \phi \downarrow, R_1 \psi \downarrow) \in S_1$. We have also that there exists $\pi_2$ a planning path from $S_0$ to $S_2$ with $\text{att}(R_2 \phi \downarrow, R_2 \psi \downarrow) \in S_2$. Therefore, the planning path obtained by concatenating $\pi_1$ and $\pi_2$ is a planning path from $S_0$ to $S_1 \cup S_2$, and we have that $\text{att}(\text{enc}(u_1, u_2), \text{enc}(v_1, v_2))$ and $\text{att}(u_2, v_2) \rightarrow \text{att}(u_1, v_1)$ which is indeed an instance of a rule in Rule_A, we obtain a planning path from $S_0$ to $S$ such that $\text{att}(R \phi \downarrow, R \psi \downarrow) \in S$. The other cases can be done in a similar way. □

We define the notion of static inclusion for destructors, which is roughly the classical notion of static inclusion restricted to destructor-only recipes.

Definition 12: A frame $\phi$ is statically included for destructors (w.r.t. $\Sigma \subseteq \Sigma_0$) in a frame $\psi$, denoted $\phi \subseteq_{\text{des}} \psi$, when $\text{dom}(\phi) = \text{dom}(\psi)$, and:

1) for any destructor-only recipe $R \in \mathcal{R}_\Sigma$, $R \phi \downarrow \in \mathcal{M}_{\Sigma_0}$ implies that $R \psi \downarrow \in \mathcal{M}_{\Sigma}$; and

2) for any destructor-only recipes $R_1, R_2 \in \mathcal{R}_\Sigma$ with $R_1 \phi \downarrow, R_2 \phi \downarrow \in \mathcal{M}_{\Sigma_0}$, we have that $R_1 \phi \downarrow = R_2 \phi \downarrow$ implies that $R_1 \psi \downarrow = R_2 \psi \downarrow$;

3) for any destructor-only recipe $R \in \mathcal{R}_\Sigma, R \phi \downarrow$ is an atom different from $c(\omega, \omega)$. The only interesting case is when $R = w$. In such a case, we have that $R \phi \downarrow = w \phi \downarrow = f(\phi \downarrow, R \phi \downarrow)$ which is not an atom, thus contradiction.

Before proving the inductive case, we establish the following claim.

Claim. Let $R_d$ be a destructor-only recipe that contains at least one destructor and such that $w$ occurs in $R_d$, and $R_d \phi \downarrow \in \mathcal{M}_{\Sigma_0}$. There exists a destructor-only recipe $R_d$ smaller than $R_d$ (i.e. with less destructor symbols than $R_d$) such that $R_d \psi \downarrow = R_d \phi \downarrow$.

Proof of the claim. We distinguish two cases depending on whether $f = \text{enc}$ or $f = \emptyset$.

Case $f = \text{enc}$. Then the destructor directly above a $w$ must be a dec (because $\text{proj}_j(w \phi \downarrow) = \text{proj}_j(\text{enc}(at \phi \downarrow, at \phi \downarrow))$ would never reduce, as a term only reduces when their subterms are messages). Moreover, it is impossible that we have $\text{dec}(R', w)$ for some recipe $R'$. In this case, we would have $R' \phi \downarrow = \text{enc}(t, w \phi \downarrow)$ for some $t$ as there is a reduction at top level in $\text{dec}(R', w \phi \downarrow)$. But $w \phi \downarrow$ is not atomic, so $\text{enc}(t, w \phi \downarrow)$ is not a message, so $\text{dec}(R', w \phi \downarrow)$ does not reduce, contradiction.

So $w$ only occurs in $\text{dec}(w, R')$ patterns. There is such a pattern where $\text{vars}(R') \subseteq \text{dom}(\phi)$, so we assume that we are in this case. We have $R' \phi \downarrow = R' \phi \downarrow$. There is a reduction at top level in $\text{dec}(w, R') \phi \downarrow = \text{dec}(\text{enc}(at \phi \downarrow, at \phi \downarrow), R') \phi \downarrow$, therefore $R' \phi \downarrow = R' \phi \downarrow \downarrow = at \phi \downarrow$. As $\phi \subseteq_{\text{des}} \psi$ and $R'$ and $at$ are destructor recipes, $R' \psi \downarrow = at \phi \downarrow$. So $\text{dec}(\text{enc}(at \phi \downarrow, at \phi \downarrow), R') \psi \downarrow = at \phi \downarrow$.

We replace one occurrence of $\text{dec}(w, R')$ by $at_1$ in $R_d$ and we get $R'_d$. Clearly, $R'_d$ has less destructors than $R_d$ and $R'_d \phi \downarrow = R_d \phi \downarrow, R'_d \psi \downarrow = R_d \psi \downarrow$ because we proved
that the equality between $\text{dec}(w, R')$ and $a_1$ was true in both frames.

Case $f = \langle \rangle$. Similarly as above, the destructor occurring directly above $w$ must be $\text{proj}_{i}$ for some $i$. But $\text{proj}_{i}(w)φ^{+} ↓ = a_{i} c_{\varphi}$ and $\text{proj}_{i}(w)\psi^{+} ↓ = a_{i} c_{\psi}$. So we replace one occurrence of $\text{proj}_{i}(w)$ by $a_{i}$ in $R_{d}$ and we get $R'_{d}$. We have that $R'_{d}φ^{+} ↓ = R_{d}φ^{+} ↓$ and $R'_{d}ψ^{+} ↓ = R_{d}ψ^{+} ↓$, and $R'_{d}$ has less destructors than $R_{d}$. This concludes the proof of the claim.

Now, we assume that the result holds for $n$, and we establish it for $n + 1$.

1) Let $R$ be a destructor recipe with $n + 1$ destructor symbols, and assume that $Rφ^{+} ↓ \in M_{Σ_{ω}}$. The only interesting case is when $w$ occurs in $R$. Our claim gives us a $R'$ with less than $n$ destructors such that $R'φ^{+} ↓ = Rφ^{+} ↓$ and $R'ψ^{+} ↓ = Rψ^{+} ↓$. Then, our induction hypothesis (item 1) applies to $R'$ so $R'φ^{+} ↓ = Rφ^{+} ↓ \in M_{Σ_{ω}}$.

2) Let $R_{1}$ (resp. $R_{2}$) be a destructor recipe with $n_{1}$ (resp. $n_{2}$) destructor symbols with $n_{1} + n_{2} \leq n + 1$, and $R_{1}φ^{+} ↓ = R_{2}φ^{+} ↓ \in M_{Σ_{ω}}$. In case $w$ occurs neither in $R_{1}$ nor in $R_{2}$, the result trivially holds. Therefore, we assume w.l.o.g. that $w$ occurs in $R_{1}$. If $R_{1}$ is not atomic, our claim (item 2) gives us a $R'_{1}$ such that $R'_{1}φ^{+} ↓ \in M_{Σ_{ω}}$. Then, applying our induction hypothesis (item 2) to $R'_{1}$ and $R_{2}$, we get that $R_{1}φ^{+} ↓ = R_{1}ψ^{+} ↓ = R_{2}ψ^{+} ↓$. Now assume that $R_{1} = w$. We can also assume that $w$ does not occur in $R_{2}$ (else $R_{2}$ is not atomic and contains $w$; we just proved the result for this case). So $R_{2}φ^{+} ↓ = R_{2}ψ^{+} ↓ = R_{2}ψ^{+} ↓$. Now, we distinguish two cases depending on $f$.

Case $f = \text{enc}$. In such a case, we have that $\text{enc}(a_{1} φ, a_{2} φ) = R_{1}φ^{+} ↓ = R_{2}φ^{+} ↓$. So $a_{1} φ = \text{dec}(R_{2}, a_{2} φ)$. As $φ \in M_{Σ_{ω}}$, $a_{1} φ = \text{dec}(R_{2}, a_{2} φ)$, which implies that $R_{1}φ^{+} ↓ = \text{enc}(a_{1} φ, a_{2} φ) = R_{2}φ^{+} ↓$. Case $f = \langle \rangle$. This case can be done in a similar way.

3) Let $R$ be a destructor recipe with $n + 1$ destructors symbols such that $Rφ^{+} ↓$ is an atom but is not $c_{(ω, ω)}$. Again, the only interesting case is when $w$ occurs in $R$. In such a case, our claim gives us a recipe $R'$ with less than $n$ destructor symbols such that $R'φ^{+} ↓ = Rφ^{+} ↓$ and $R'ψ^{+} ↓ = Rψ^{+} ↓$. Applying our induction hypothesis (item 3) on $R'$, we obtain that $R'ψ^{+} ↓$ is a message different from $c_{(ω, ω)}$. Therefore, $Rψ^{+} ↓$ is a message different from $c_{(ω, ω)}$.

This concludes the proof.

Lemma 11: Let $φ$ and $ψ$ be two frames such that $φ \in M_{Σ_{ω}}$. Let $R$ be a recipe such that $Rφ^{+} ↓$ is a message. We have that $Rψ^{+} ↓$ is a message and $φ^{+} \in M_{Σ_{ω}}$ where $φ^{+} = φ \cup \{w \rightarrow Rφ^{+} ↓\}$ and $ψ^{+} = ψ \cup \{w \rightarrow Rψ^{+} ↓\}$ for any fresh variable $w$.

Proof. We prove this result by structural induction on $R$.

Base case. $R \in \Sigma \cup \text{dom}(φ)$. In such a case, the result trivially holds.

Inductive cases. In such a case, we have that $R = f(R_1, R_2)$ with $f = \{\text{enc}, \text{dec}, \langle \rangle\}$, or $R = \text{proj}_{i}(R')$ with $i \in \{1, 2\}$. We distinguish two cases depending on whether $f \in \{\text{enc}, \langle \rangle\}$ or $f \in \{\text{dec}, \text{proj}_{i}, \text{proj}_{j}\}$.

$f$ is a destructor symbol. We assume w.l.o.g. that $R = \text{dec}(R_1, R_2)$. Since $Rψ^{+} ↓$ is a message, we know that $R_1 φ^{+} ↓$ and $R_2 ψ^{+} ↓$ are messages. Thanks to our induction hypothesis, we know that $R_1 ψ^{+} ↓$ and $R_2 ψ^{+} ↓$ are messages too, and $φ' \in \text{des} ψ'$ where:

- $φ' = φ \cup \{w_1 \rightarrow R_1 φ^{+} ↓, w_2 \rightarrow R_2 φ^{+} ↓\}$;
- $ψ' = ψ \cup \{w_1 \rightarrow R_1 ψ^{+} ↓, w_2 \rightarrow R_2 ψ^{+} ↓\}$.

Since $f = \text{dec}$ is a destructor symbol, $φ' \in \text{des} ψ'$, and $f(R_1 φ^{+} ↓, R_2 φ^{+} ↓) = Rψ^{+} ↓$ is a message, we deduce that $f(R_1 ψ^{+} ↓, R_2 ψ^{+} ↓) = Rψ^{+} ↓$ is a message. We have also that $ψ'' \in \text{des} ψ''$ where $ψ'' = ψ' \cup \{w \rightarrow Rψ^{+} ↓\}$ and $ψ'' = ψ' \cup \{w \rightarrow Rψ^{+} ↓\}$, and thus we conclude that $φ^{+} \in \text{des} ψ^{+}$.

This concludes the proof.

Now, we prove that the two notions of static inclusion coincide.

Proposition 3: Let $φ$ and $ψ$ be two frames. We have that $φ \subseteq ψ$ if, and only if, $φ \in M_{Σ_{ω}}$.

Proof. Let $φ$ and $ψ$ be two frames. We show the two directions separately.

$(\Rightarrow)$ We assume that $φ \subseteq ψ$, and we have to establish that $φ \in M_{Σ_{ω}}$. First, the items 1 and 2 of Definition 12 are direct consequences of the definition of static inclusion. Now, let $R$ be a destructor recipe such that $Rφ^{+} ↓$ is atomic and different from $c_{(ω, ω)}$. Let $c$ be a constant in $Σ_{0}$, and let $R' = \text{enc}(c, R)$. We have that $R'φ^{+} ↓ = \text{enc}(c, Rφ^{+} ↓)$ is a message. Therefore, $R'ψ^{+} ↓ = \text{enc}(c, Rψ^{+} ↓)$ is a message as $φ \subseteq ψ$. Hence, we have that $Rψ^{+} ↓$ is atomic and different from $c_{(ω, ω)}$. This allows us to conclude.

$(\Leftarrow)$ We assume that $φ \in M_{Σ_{ω}}$, and we have to establish that $φ \subseteq ψ$. We show that the two items of Definition 3 are satisfied. First, let $R$ be a recipe such that $Rφ^{+} ↓$ is a message. Thanks to Lemma 11, we know that $Rψ^{+} ↓$ is a message too.

Second, let $R_1$ and $R_2$ be two recipes such that $R_1 φ^{+} ↓ = R_2 ψ^{+} ↓ \in M_{Σ_{ω}}$. Thanks to Lemma 11, we obtain that $φ' \in M_{Σ_{ω}}$ where:

- $φ' = φ \cup \{w_1 \rightarrow R_1 φ^{+} ↓, w_2 \rightarrow R_2 φ^{+} ↓\}$;
- $ψ' = ψ \cup \{w_1 \rightarrow R_1 ψ^{+} ↓, w_2 \rightarrow R_2 ψ^{+} ↓\}$.

In particular the equation $w_1 = w_2$ holds in $φ'$ and $ψ$, and $w_1$ and $w_2$ are destructor recipes so this equation also holds in $ψ'$, i.e. $R_1 ψ^{+} ↓ = R_2 ψ^{+} ↓$. □
Now we can state and prove our main proposition:

**Proposition 1:** Let \( \phi \) and \( \psi \) be two frames with \( \text{dom}(\phi) = \text{dom}(\psi) \), and \( \Theta = (\text{Fact}_0, \text{Fact}(\phi, \psi), \mathcal{R}) \) where

\[
\mathcal{R} = \text{Concrete}(\text{Rule}_\Lambda) \cup \mathcal{R}_{\text{fail}}^\text{test} \cup \mathcal{R}_{\text{fail}}^\text{atom}.
\]

Let \( \Pi = (\Theta, \{\text{bad}\}) \). We have that \( \phi \not\subseteq_s \psi \) if, and only if, \( \Pi \) has a solution.

**Proof.** First, thanks to Proposition 3, it is sufficient to show that \( \phi \not\subseteq_s \psi \) if, and only if, \( \Pi \) has a solution. We show the two directions separately.

(\( \Rightarrow \)) We have that \( \phi \not\subseteq_s \psi \). Following the definition of \( \sqsubseteq_s \), we consider 3 cases.

1. There is a destructor recipe \( R \) such that \( R\phi_\downarrow \in \mathcal{M}_{\mathcal{S}_0} \) but \( R\psi_\downarrow \notin \mathcal{M}_{\mathcal{S}_0} \). We consider one having a minimal size. Since both \( \phi \) and \( \psi \) are frames, we have that \( R = \text{dec}(R_1, R_2) \) or \( R = \text{proj}_i(R') \) with \( i \in \{1, 2\} \).

2. There are two destructor recipes \( R_1 \) and \( R_2 \) such that \( R_1\phi_\downarrow = R_2\phi_\downarrow \in \mathcal{M}_{\mathcal{S}_0} \) whereas \( R_1\psi_\downarrow \neq R_2\psi_\downarrow \). First, thanks to the first item, we may assume that \( R_1\phi_\downarrow \) and \( R_2\psi_\downarrow \) are both in \( \mathcal{M}_{\mathcal{S}_0} \). Therefore, thanks to Lemma 1, we know that there exists a plan \( \pi_1 \) (resp. \( \pi_2 \)) or \( \text{att}(R_1\phi_\downarrow, R_1\psi_\downarrow) \) (resp. \( \text{att}(R_2\phi_\downarrow, R_2\psi_\downarrow) \)). We consider the plan obtained by ”concatenating” the plans \( \pi_1 \) and \( \pi_2 \), and we consider the rule \( r \) of the form:

\[
\text{att}(R_1\phi_\downarrow, R_1\psi_\downarrow), \text{att}(R_2\psi_\downarrow, R_2\psi_\downarrow) \rightarrow \text{bad}
\]

which is indeed an instance of a rule \( \mathcal{R}_{\text{fail}}^\text{test} \). Therefore, we have shown that \( \Pi \) has a solution.

3. There is a destructor recipe \( R \) such that \( R\phi_\downarrow \) is an atom different from \( c(\omega_\omega) \), whereas \( R\psi_\downarrow \) is either not an atom or it is \( c(\omega_\omega) \). First, thanks to the first item, we may assume that \( R\psi_\downarrow \in \mathcal{M}_{\mathcal{S}_0} \). Therefore, thanks to Lemma 1, we know that there exists a plan \( \pi \) of or \( \text{att}(R\phi_\downarrow, R\psi_\downarrow) \). We consider the rule \( r \) of the form:

\[
\text{att}(R\phi_\downarrow, R\psi_\downarrow) \rightarrow \text{bad}
\]

which is indeed an instance of a rule \( \mathcal{R}_{\text{fail}}^\text{atom} \), which leads to a solution for \( \Pi \). Therefore, we have also that \( \Pi \) has a solution.

So in any case we have shown that \( \Pi \) has a solution.

(\( \Leftarrow \)) We have a plan of bad. We consider such a plan \( r_1, \ldots, r_n \) of minimal length. Since this plan is minimal, we know that \( r_1, \ldots, r_{n-1} \) are rules in \( \text{Concrete}^\text{−} \) (Rule\(_\Lambda\)), and therefore, we can rely on Lemma 1 to conclude that there exists a destructor-only recipe \( R \) such that \( R\phi_\downarrow = u \) and \( R\psi_\downarrow = v \) for any \( \text{att}(u, v) \in S_{n-1} \) (the state resulting from the application of \( r_1, \ldots, r_{n-1} \)). Then, in order to derive bad, we have applied either a rule in \( \text{Concrete}^\text{−} \) (Rule\(_\Lambda\)), or a rule in \( \mathcal{R}_{\text{fail}}^\text{test} \), or a rule in \( \mathcal{R}_{\text{fail}}^\text{atom} \). The two last cases are quite obvious, and we easily derive a witness of \( \phi \not\subseteq_s \psi \) relying on item 2 (resp. item 3) of Definition 12. Regarding the first case, according to the definition of \( \text{Concrete}^\text{−} \) (Rule\(_\Lambda\)), we distinguish two cases:

- \( r_n = \text{att}((u_1, u_2), v) \rightarrow \text{bad} \) with \( v \) not a pair; or
- \( r_n = \text{att}(\text{enc}(u_1, u_2), v), \text{att}(u_2, v') \rightarrow \text{bad} \) with \( v \) not of the form \( \text{enc}(v_i, v_j) \) for some \( i_0 \).

Moreover, we have destructor-only recipes \( R_1 \) (and \( R_2 \) allowing us to derive these facts. In the first case, we conclude using the recipe \( \text{proj}_i(R_1) \) and checking whether it is a message or not. In the second case, we do the same with \( \text{dec}(R_1, R_2) \).

**APPENDIX C**

FROM TRACE EQUIVALENCE TO PLANNING

**Lemma 12:** Let \( r = \text{Pre}, \text{att}(u, v) \rightarrow \text{Add}; \text{Del} \) be an abstract rule. Let \( \sigma \) be a grounding substitution for \( r \) such that \( \delta_\text{p}(\sigma x) \leq \delta_\text{p}(x) \) for any \( x \in \text{vars}_{\text{left}}(r) \). Let \( C \) be a constructor context such that \( \sigma = C[u_1, \ldots, u_n] \) and \( \nu\sigma = C[v_1, \ldots, v_n] \). Then there exists \( r' \in \text{Flat}(r) \):

\[
r' = \text{Pre}', \text{att}(u_1', v_1'), \ldots, \text{att}(u_n', v_n') \rightarrow \text{Add'}; \text{Del'}
\]

and \( \sigma' \) a grounding substitution for \( r' \) such that:

1. \( \delta_\text{p}(\sigma' x) \leq \delta_\text{p}(x) \) for any \( x \in \text{vars}_{\text{left}}(r') \);
2. \( (\text{Pre}', \text{Add'}, \text{Del'})\sigma' = (\text{Pre}, \text{Add}, \text{Del})\sigma \); and
3. \( \text{att}(u, v)\sigma = \text{att}(C[u_1', \ldots, u_n'], C[v_1', \ldots, v_n'])\sigma' \).

**Proof.** We first establish the following claim:

**Claim.** Let \( r = \text{Pre}, \text{att}(u, v) \rightarrow \text{Add}; \text{Del} \) be an abstract rule. Let \( \sigma \) be a grounding substitution for \( r \) such that \( \delta_\text{p}(\sigma x) \leq \delta_\text{p}(x) \) for any \( x \in \text{vars}_{\text{left}}(r) \). Let \( f \in \mathcal{S}_\Lambda \) be such that \( u\sigma = f(u_1, u_2) \) and \( v\sigma = f(v_1, v_2) \) for some terms \( u_1, u_2, v_1, v_2 \). Then \( u \) is decomposable, and \( r_1 = \text{decomp}(r, \text{att}(u, v)) \) is of the following form:

\[
r_1 = \text{Pre}_1, \text{att}(u_1', v_1'), \text{att}(u_2', v_2') \rightarrow \text{Add}_1; \text{Del}_1
\]

Moreover, there is a grounding substitution \( \sigma_1 \) for \( r_1 \) such that:

1. \( \delta_\text{p}(\sigma_1 x) \leq \delta_\text{p}(x) \) for any \( x \in \text{vars}_{\text{left}}(r_1) \);
2. \( (\text{Pre}_1, \text{Add}_1, \text{Del}_1)\sigma_1 = (\text{Pre}, \text{Add}, \text{Del})\sigma \); and
3. \( u\sigma = f(u_1', u_2') \sigma_1 \) and \( v\sigma = f(v_1', v_2') \sigma_1 \).

**Proof of the Claim.** We have that \( u\sigma = f(u_1, u_2) \) and we know that \( \delta_\text{p}(\sigma x) \leq \delta_\text{p}(x) \) for any \( x \in \text{vars}_{\text{left}}(r) \). Therefore, we know that \( u \) is decomposable. We have that

\[
\text{split}(\text{att}(u, v)) = (f, \{\text{att}(x_1, y_1), \text{att}(x_2, y_2)\}, \sigma_\text{p}, \sigma_\text{q})
\]

with \( \delta_\text{p}(x_1) = \tau_1, \delta_\text{p}(x_2) = \tau_2, \sigma_\text{p} = \text{mgu}(u, f(x_1, x_2)) \) is quasi-well-typed, and \( \sigma_\text{q} = \text{mgu}(v, f(y_1, y_2)) \). Note that
\[ \sigma_Q \neq \perp \text{ since } v\sigma = f(v_1, v_2). \] Moreover, when \( u \) (resp \( v \)) is not a variable, we assume w.l.o.g. that \( x_1, x_2 \) (resp. \( y_1, y_2 \)) do not occur in \( \text{im}(\sigma P) \) (resp. \( \text{im}(\sigma Q) \)). Let \( r_1 = \text{decompo}(r, \text{att}(u, v)) \). We have that:

\[
r_1 = [\text{Pre}, \text{att}(x_1, y_1), \text{att}(x_2, y_2) \rightarrow \text{Add}; \text{Del}][\sigma P \uplus \sigma Q].
\]

Let \( \sigma_{\text{left}} = \sigma|_{\text{vars}_{\text{left}}(r)} \) and \( \sigma_{\text{right}} = \sigma|_{\text{vars}_{\text{right}}(r)} \). Note that \( \text{vars}_{\text{left}}(r) \cap \text{vars}_{\text{right}}(r) = \emptyset \) and thus \( \sigma = \sigma_{\text{left}} \uplus \sigma_{\text{right}} \).

Now, we make a distinction depending on whether \( u \) (resp. \( v \)) is a variable or not. In case \( u \) is a variable, say \( z_u \), then we have that \( \sigma = \{ z_u \mapsto f(x_1, x_2) \} \), and we let \( \sigma_{\text{left}} = \sigma_{\text{left}} \uplus \{ x_1 \mapsto u_1, x_2 \mapsto u_2 \} \). Otherwise, we have that \( u = f(a_1, a_2) \), and \( \sigma_P = \{ x_1 \mapsto a_1, x_2 \mapsto a_2 \} \), and we let \( \sigma_{\text{left}} = \sigma_{\text{left}} \).

We proceed similarly for \( v \). It remains to show that \( r_1 \) and \( \sigma_1 = \sigma_{\text{left}} \uplus \sigma_{\text{right}} \) as defined above satisfy the requirements.

We establish each property separately.

1. We have that \( \delta_r(x_{\sigma}) = \delta_r(x_{\sigma_{\text{left}}}) = \delta_r(x_{\sigma_{\text{right}}}) \) for any \( x \in \text{vars}_{\text{left}}(r_1) \). Hence, the result.

2. In case \( u \) is a variable, we have that \( (z_u \sigma_P) \sigma_{\text{left}} = z_u \sigma \), and similarly for \( v \). Therefore, it is easy to see that \( (\text{Pre}, \text{Add}, \text{Del})(\sigma P \uplus \sigma Q)_{\sigma_1} = (\text{Pre}, \text{Add}, \text{Del})_{\sigma} \).

3. We have that \( f(u_1', u_2') \sigma_1 = f(x_1 \sigma_P, x_2 \sigma_P)_{\sigma} = u_\sigma \) and similarly for \( v \).

This concludes the proof of the claim.

Now we prove the main result by induction on \( C \). The base case, i.e. \( C = f(C_1, C_2) \). From our claim, we get:

\[
r_1 = \text{decompo}(r, \text{att}(u, v)) \text{ of the following form}
\]

\[
r_1 = [\text{Pre}, \text{att}(u'_1, v'_1), \text{att}(u'_2, v'_2) \rightarrow \text{Add}; \text{Del}]
\]

and a substitution \( \sigma_1 \) grounding for \( r_1 \) such that:

1. \( \delta_r(x_{\sigma}) \leq \delta_r(x_{\sigma_{\text{left}}}(r_1)) \).
2. \( (\text{Pre}_1, \text{Add}_1, \text{Del}_1)_{\sigma_1} = (\text{Pre}, \text{Add}, \text{Del})_{\sigma} \).
3. \( u_{\sigma} = f(u_1', u_2')_{\sigma_1} \) and \( v_{\sigma} = f(v_1', v_2')_{\sigma_1} \).

In particular we have that \( C_1[u_1, \ldots , u_n] = u'_1 \sigma_1 \) and \( C_1[v_1, \ldots , v_n] = v'_1 \sigma_1 \) for each \( i \in \{1, 2\} \).

Now we write:

\[
r_2 = [\text{Pre}_2, \text{att}(u_1', v_1'), \ldots , \text{att}(u_2', v_2') \rightarrow \text{Add}_2; \text{Del}_2]
\]

and we apply our induction hypothesis with context \( C_2 \) and substitution \( \sigma_2 \). We get a rule \( r_2 \in \text{Flat}(r_1) \) such that:

\[
r_2 = [\text{Pre}_2, \text{att}(u_1', v_1'), \ldots , \text{att}(u_2', v_2') \rightarrow \text{Add}_2; \text{Del}_2]
\]

and a substitution \( \sigma_2 \) grounding for \( r_2 \) such that:

1. \( \delta_r(x_{\sigma}) \leq \delta_r(x_{\sigma_{\text{left}}}(r_2)) \).
2. \( (\text{Pre}_2, \text{Add}_2, \text{Del}_2)_{\sigma_2} = (\text{Pre}_1, \text{Add}_1, \text{Del}_1)_{\sigma_1} \).
3. \( u_0' \sigma_1 = C_2[u_1', \ldots , u_n'] \sigma_2 \) and similarly \( v_0' \sigma_1 = C_2[v_1', \ldots , v_n'] \sigma_2 \).

We can write:

\[
r_2 = [\text{Pre}_2', \text{att}(u_0', v_0'), \text{att}(u_2', v_2') \rightarrow \text{Add}_2; \text{Del}_2]
\]

where \( u_0'' \sigma_2 = u_1' \sigma_1 \) and \( v_0'' \sigma_2 = v_1' \sigma_1 \) and we apply our induction hypothesis with context \( C_1 \) and substitution \( \sigma_2 \). We get a rule \( r_3 \in \text{Flat}(r_2) \) such that:

\[
r_3 = [\text{Pre}_3, \text{att}(u_1', v_1'), \ldots , \text{att}(u_n', v_n') \rightarrow \text{Add}_3; \text{Del}_3]
\]

and a substitution \( \sigma_3 \) grounding for \( r_3 \) such that:

1. \( \delta_r(x_{\sigma}) \leq \delta_r(x_{\sigma_{\text{left}}}(r_3)) \).
2. \( (\text{Pre}_3, \text{Add}_3, \text{Del}_3)_{\sigma_3} = (\text{Pre}_2', \text{Add}_2, \text{Del}_2)_{\sigma_2} \).
3. \( u_0' \sigma_2 = C_1[u_1', \ldots , u_n'] \sigma_3 \) and similarly \( v_0' \sigma_2 = C_1[v_1', \ldots , v_n'] \sigma_3 \).

We have that \( r_3 \in \text{Flat}(r) \) and it remains to check that \( r_3 \) and \( \sigma_3 \) as defined above satisfy our three conditions. First, we have seen that \( \delta_r(x_{\sigma}) \leq \delta_r(x_{\sigma}) \) for any \( x \in \text{vars}_{\text{left}}(r_3) \).

Second, we have that:

\[
(\text{Add}_3, \text{Del}_3)_{\sigma_3} = (\text{Add}_2, \text{Del}_2)_{\sigma_2} = (\text{Add}_1, \text{Del}_1)_{\sigma_1} = (\text{Add}, \text{Del})_{\sigma} \]

As \( \text{Pre}_3 \sigma_3 = \text{Pre}_2 \sigma_2 \), we have:

\[
\text{Pre}_3 = \text{Pre}_3', \text{att}(u_1', v_1'), \ldots , \text{att}(u_n', v_n')
\]

for some \( \text{Pre}_3' \) where \( u_i' \sigma_3 = u_i' \sigma_2 \) and \( v_i' \sigma_2 = v_i' \sigma_2 \) for each \( i \in \{1, \ldots , n\} \). Hence, we have that:

\[
\text{Pre}_3' \sigma_3, \text{att}(u_1', v_1'), \ldots , \text{att}(u_n', v_n') \sigma_3, \text{att}(u_0'', v_0'') \sigma_2 = \text{Pre}_3 \sigma_3, \text{att}(u_0', v_0') \sigma_2 = \text{Pre}_3 \sigma_3, \text{att}(u_0', v_0') \sigma_2
\]

Hence, we have that \( \text{Pre}_3' \sigma_3 = \text{Pre}_3 \sigma_3 \) and we apply our induction hypothesis with context \( C \) and substitution \( \sigma_3 \). Lastly, we have that:

\[
(\text{Add}_1, \text{Del}_1)_{\sigma_1} = (\text{Add}, \text{Del})_{\sigma}
\]

We can establish in a similar way that \( v_\sigma = f(C_1[v_1', \ldots , v_n'], C_2[v_1', \ldots , v_n'])_{\sigma} \), and this concludes the proof. \( \square \)

For the next lemmas, we need to be more specific on how the fact bad has appeared. Therefore, from now on, we consider three facts instead: bad-proto, bad-flat, and bad-concrete. Moreover, we assume that in protocol rules, bad is replaced by bad-proto, in flattening rules, bad is replaced by bad-flat, and in concretization rules bad is replaced by bad-concrete. When the precise origin of the failure does not matter, we simply write bad (meaning one of the three cases above).
Lemma 13: Let $r$ be an abstract protocol rule. Let $r' \in \text{Flat}(r)$ written as

$$r' = \text{state}, \text{att}(u_1, v_1), \ldots, \text{att}(u_n, v_n) \rightarrow \text{Add}; \text{Del}$$

where $\text{state}$ is a state fluent.

- Either we have bad-flat $\notin \text{Add}$, and then there exists a constructor context $C$ and a substitution $\tau$ such that

$$\tau r = \text{state}, \text{att}(u, v) \rightarrow \text{Add}; \text{Del}.$$ 

where $u = C[u_1, \ldots, u_n]$ and $v = C[v_1, \ldots, v_m]$.

- Or $\text{Add} = \text{bad-flat}$, $\text{Del} = \emptyset$ and there exists a constructor context $C$, a substitution $\tau$ and a term $v$ and two sets $\text{Add}_0$ and $\text{Del}_0$ such that $\tau r = \text{state}, \text{att}(C[u_1, \ldots, u_n], v) \rightarrow \text{Add}_0; \text{Del}_0$ but $v$ does not unify with $C$.

**Proof.** As $r' \in \text{Flat}(r)$, there exists a sequence $r_0, \ldots, r_n$ of rules, and a sequence $f_0, \ldots, f_{n-1}$ of facts, such that $r_0 = r$, $r_n = r'$ and for each $0 \leq i \leq n-1$, we have that $r_{i+1} = \text{decompo}(r_i, f_i)$. We establish the result by induction on $n$.

The base case, i.e. $n = 0$, is trivial.

Assume that we have the result for $n$. Let

$$r_n = \text{state}_n, \text{att}(u_1, v_1), \ldots, \text{att}(u_m, v_m) \rightarrow \text{Add}_n; \text{Del}_n.$$ 

and $r_{n+1} = \text{decompo}(r_n, f_n) = \text{Pre}_n + 1 \rightarrow \text{Add}_n; \text{Del}_n$. Without loss of generality, we can assume that $f_n = \text{att}(u_m, v_m)$.

**First case.** bad-flat $\notin \text{Add}_n$. Then, as $r_{n+1} = \text{decompo}(r_n, f_n)$, bad-flat $\notin \text{Add}_n$. By induction hypothesis, there exist $C$ and $\tau_n$ such that:

$$\tau r_n = \text{state}_n, \text{att}(u_0, v_0) \rightarrow \text{Add}_n; \text{Del}_n.$$ 

where $u_0 = C[u_1, \ldots, u_m]$ and $v_0 = C[v_1, \ldots, v_m]$.

We have split($f_n$) = $(f, S, \sigma_P, \sigma_Q)$ with $S = \{\text{att}(x_1, y_1), \text{att}(x_2, y_2)\}$, $\sigma_P = \text{mgu}(u_m, f(x_1, x_2))$, and $\sigma_Q = \text{mgu}(v_m, f(y_1, y_2))$. Moreover, since bad-flat $\notin \text{Add}_n$, we have that $\sigma_Q \neq \bot$. We get the following rule $r_{n+1}$:

$$(\text{state}_n, \text{att}(u_1, v_1), \ldots, \text{att}(u_m, v_m),$$

$$(x_1, y_1), (x_2, y_2) \rightarrow \text{Add}_n; \text{Del}_n)(\sigma_P \uplus \sigma_Q).$$

Let $\tau = \tau_n(\sigma_P \uplus \sigma_Q)$ and $C = C[\ldots, f(\ldots)]$. We have that

$$\tau r_n(\sigma_P \uplus \sigma_Q) = \text{state}_n(\sigma_P \uplus \sigma_Q), \text{att}(u_0\sigma_P, v_0\sigma_Q) \rightarrow \text{Add}_n(\sigma_P \uplus \sigma_Q);$$

$$\text{Del}_n(\sigma_P \uplus \sigma_Q).$$

It only remains to establish that $u_0\sigma_P = C[u_1\sigma_P, \ldots, u_{m-1}\sigma_P, x_1\sigma_P, x_2\sigma_P]$ (and similarly for $v_0\sigma_Q$).

We have that:

$$u_0\sigma_P = C[u_1, \ldots, u_m]\sigma_P$$

$$= C[u_1\sigma_P, \ldots, u_{m-1}\sigma_P, f(x_1, x_2)\sigma_P]$$

$$= C[u_1\sigma_P, \ldots, u_{m-1}\sigma_P, x_1\sigma_P, x_2\sigma_P]$$

Hence, this case is proved.

**Second case.** bad-flat $\in \text{Add}_{n+1}$ but bad-flat $\notin \text{Add}_n$. By induction hypothesis, there exist $C$ and $\tau_n$ such that:

$$\tau r_n = \text{state}_n, \text{att}(u_0, v_0) \rightarrow \text{Add}_n; \text{Del}_n$$

where $u_0 = C[u_1, \ldots, u_m]$ and $v_0 = C[v_1, \ldots, v_m]$.

We have split($f_n$) = $(f, S, \sigma_P, \bot)$ with $S = \{\text{att}(x_1, y_1), \text{att}(x_2, y_2)\}$, $\sigma_P = \text{mgu}(u_m, f(x_1, x_2))$. We get the following rule $r_{n+1}$:

$$(\text{state}_n, \text{att}(u_1, v_1), \ldots, \text{att}(u_{m-1}, v_{m-1}),$$

$$(x_1, y_1), (x_2, y_2) \rightarrow \text{bad-flat}\sigma_P)$$

Let $\tau = \tau_n(\sigma_P \uplus \sigma_Q)$ and $C = C[\ldots, f(\ldots)]$. We have that:

$$\tau r_n(\sigma_P \uplus \sigma_Q) = \text{state}_n(\sigma_P \uplus \sigma_Q), \text{att}(u_0\sigma_P, v_0\sigma_Q) \rightarrow \text{Add}_n(\sigma_P \uplus \sigma_Q);$$

$$\text{Del}_n(\sigma_P \uplus \sigma_Q).$$

It only remains to establish that $u_0\sigma_P = C[u_1\sigma_P, \ldots, u_{m-1}\sigma_P, x_1\sigma_P, x_2\sigma_P]$ but $v_0\sigma_Q$ does not unify with $C$. We have that:

$$u_0\sigma_P = C[u_1, \ldots, u_m]\sigma_P$$

$$= C[u_1\sigma_P, \ldots, u_{m-1}\sigma_P, f(x_1, x_2)\sigma_P]$$

$$= C[u_1\sigma_P, \ldots, u_{m-1}\sigma_P, x_1\sigma_P, x_2\sigma_P]$$
and $v_0\sigma_Q$ does not unify with $C_n$, so it does not unify with $C$, which concludes the proof of this subcase.

**Subcase 3.2.** $\sigma_Q = \bot$. We get the following rule $r_{n+1}$:

$\begin{align*}
\text{state}_n, \text{att}(u_1, v_1), \ldots, \text{att}(u_{m-1}, v_{m-1}), \\
\text{att}(x_1, y_1), \text{att}(x_2, y_2) &\rightarrow \text{bad-flat}\sigma_P
\end{align*}$

Let $\tau = \tau_n\sigma_P$ and $C = C_n[\underline{\ldots}, f(\underline{\ldots})]$. We have that

$$
r_\tau n\sigma_P
= \text{state}_n\sigma_P, \text{att}(u_0\sigma_P, v_0) \rightarrow \text{Add}_n\sigma_P; \\
\text{Del}_n\sigma_P.
$$

It only remains to establish that $u_0\sigma_P = C[u_1\sigma_P, \ldots, u_m\sigma_P, x_1\sigma_P, x_2\sigma_P]$ but $v_0$ does not unify with $C$. We have that:

$$
\begin{align*}
u_0\sigma_P
= C_n[u_1, \ldots, u_m]\sigma_P \\
= C_n[u_1\sigma_P, \ldots, u_m\sigma_P, f(x_1, x_2)\sigma_P] \\
= C[u_1\sigma_P, \ldots, u_m\sigma_P, x_1\sigma_P, x_2\sigma_P]
\end{align*}
$$

and $v_0$ does not unify with $C_n$, so it does not unify with $C$, which concludes the proof of this subcase, and hence of the lemma.

**Lemma 14:** Let $r$ be an abstract rule, $r = \text{Pre}, \text{att}(u, v) \rightarrow \text{Add}; \text{Del}$. Let $\sigma$ be a substitution grounding for $\text{vars}_{\text{left}}(r)$ such that $\delta_P(x\sigma) \preceq \delta_P(x)$ for any $x \in \text{vars}_{\text{left}}(r)$, and $f$ a constructor function symbol such that $u\sigma = f(u_1, v_2)$. Assume that $\text{Add} = \text{bad}$ or $v$ does not unify with $f(\underline{\ldots})$.

Then $u$ is decomposable. Let $r'$ defined by

$$
r' = \text{decomp}(r, \text{att}(u, v))
= \text{Pre}', \text{att}(u'_1, v'_1), \text{att}(u'_2, v'_2) \rightarrow \text{bad}
$$

If $\text{Add} \neq \text{bad}$ then there is a substitution $\sigma'$ grounding for $\text{vars}_{\text{left}}(r')$ such that $\delta_P(x\sigma') \preceq \delta_P(x)$ for any $x \in \text{vars}_{\text{left}}(r')$.

$\text{Pre}'\sigma' = \text{Pre}_\sigma, f(u'_1, u'_2)\sigma' = u\sigma$ and $v'_1, v'_2$ are distinct variables that do not occur elsewhere in $r'$.

If $\text{Add} = \text{bad}$ and $\text{Del} = \emptyset$, the result still holds except that nothing is required on $v'_1$ and $v'_2$ if $v$ unifies with $f(\underline{\ldots})$.

**Proof.** $u\sigma = f(u_1, u_2)$ so either $u$ is not atomic or $u$ is a variable and $\delta_P(u\sigma) \preceq \delta_P(u) = f(\delta_P(u_1), \delta_P(u_2))$. So in both cases $u$ is decomposable.

We first consider the case where $\text{Add} \neq \text{bad}$.

As $v$ does not unify with $f(v_1, v_2)$ for any $v_1, v_2$.

$$\text{split}(\text{att}(u, v)) = (f, \text{att}(x_1, y_1), \text{att}(x_2, y_2), \sigma_P, \bot)$$

where $\sigma_P = \text{mgu}(u, f(x_1, x_2))$ and $\delta_P(x_1) = \delta_P(u_i)$.

**First case.** $u$ is a variable. We have $\sigma_P = \{u \mapsto f(x_1, x_2)\}$.

Let $r'$ defined by

$$
r' = \text{decomp}(r, \text{att}(u, v))
= \text{Pre}_\sigma, \text{att}(x_1\sigma_P, y_1), \text{att}(x_2\sigma_P, y_2) \rightarrow \text{bad}
$$

and $\sigma'$ defined by $\sigma' = \{x_1 \mapsto u_1; x_2 \mapsto u_2\} \cup \sigma$.

As $x_1, x_2 \notin \text{dom}(\sigma)$, $\sigma'$ is well-defined. We have $\delta_P(x\sigma') \preceq \delta_P(x)$ for any $x \in \text{vars}_{\text{left}}(\sigma')$. Moreover, $\text{Pre}'\sigma' = \text{Pre}_\sigma\sigma' = \text{Pre}_\sigma$ as $\sigma'$ coincide with $\sigma$ on $\text{dom}(\sigma)$ and $u\sigma, \sigma' = u\sigma$.

Finally, $y_1, y_2$ are variables that do not occur elsewhere in $r'$. So it concludes the proof for this case.

**Second case.** $u = f(a_1, a_2)$ for some $a_i$ with $a_i\sigma = u_i$ for each $i \in \{1, 2\}$. We have $\sigma_P = \{x_1 \mapsto a_1; x_2 \mapsto a_2\}$. Let $r'$ defined by

$$
r' = \text{decomp}(r, \text{att}(u, v))
= \text{Pre}_\sigma, \text{att}(x_1\sigma_P, y_1), \text{att}(x_2\sigma_P, y_2) \rightarrow \text{bad}
$$

and $\sigma'$ defined by $\sigma' = \sigma$.

We have $\delta_P(x\sigma') \preceq \delta_P(x)$ for any $x \in \text{vars}_{\text{left}}(\sigma')$. Moreover, $\text{Pre}'\sigma' = \text{Pre}_\sigma\sigma' = \text{Pre}_\sigma$ as $\sigma'$ coincide with $\sigma$ on $\text{dom}(\sigma)$ and $\text{dom}(\sigma) \cap \text{vars}(\text{Pre}) = \emptyset$. Finally, $y_1, y_2$ are variables that do not occur elsewhere in $r'$. So it concludes the proof of the main result.

We now consider the case where $\text{Add} = \text{bad}$.

Now, if $v$ does unify with $f(\underline{\ldots})$, then, the hypotheses of Lemma 12 are satisfied, so we get its conclusion, which implies the desired result.

For the next lemmas, we need to define the relation $=_{\text{left}}$ on facts as $f_1 =_{\text{left}} f_2$ iff $f_1 = \text{att}(u, v), f_2 = \text{att}(u', v')$ with $u = u'$, or $f_1 = \text{state}_P\sigma_Q(\sigma_P, \sigma_Q), f_2 = \text{state}_P\sigma_Q'(\sigma_P, \sigma_Q')$ with $c = c'$, $P = P'$ and $\sigma_P = \sigma_P'$. We extend this definition to $\text{Pre}$.

**Lemma 15:** Let $r = \text{Pre}, \text{att}(u_1, v_1), \ldots, \text{att}(u_n, v_n) \rightarrow \text{bad}$. Let $\sigma$ be a substitution grounding for $\text{vars}_{\text{left}}(r)$ such that $\delta_P(x\sigma) \preceq \delta_P(x)$ for each $x \in \text{vars}_{\text{left}}(r)$.

Let $C_1, \ldots, C_n$ be constructor contexts. We assume that $u_1\sigma = C_1[u_1^1, \ldots, u_k^1]$. Then there exists $r' \in \text{Flat}(r)$ such that $r' = \text{Pre}', \text{att}(s_1, t_1), \ldots, \text{att}(s_k^n, t_k^n) \rightarrow \text{bad}$ and $\sigma'$ such that $u_i^1 = s_i^1\sigma'$ and $\text{Pre}'\sigma' =_{\text{left}} \text{Pre}_\sigma$ and the $t_i^n$ are any terms.

**Proof.** We first prove the following claim:

**Claim.** Let $r = \text{Pre}, \text{att}(u, v) \rightarrow \text{bad}$. Let $\sigma$ be a substitution grounding for $\text{vars}_{\text{left}}(r)$ and such that $\delta_P(x\sigma) \preceq \delta_P(x)$ for each $x \in \text{vars}_{\text{left}}(r)$.

Let $C$ be a constructor context. We assume that $u\sigma = C[u_1, \ldots, u_n]$. Then there exists $r' \in \text{Flat}(r)$ such that $r' = \text{Pre}', \text{att}(u_1, v_1), \ldots, \text{att}(u_n, v_n) \rightarrow \text{bad}$ and $\sigma'$ such that $u_i = u_i^1\sigma'$ and $\text{Pre}'\sigma' =_{\text{left}} \text{Pre}_\sigma$.

**Proof of the claim.** We proceed by induction on $C$. The base case is obvious. We assume $C = f(C_1, C_2)$. By Lemma 14, we have that $u$ is decomposable, and $r_1 = \text{decomp}(r, \text{att}(u, v)) = \text{Pre}_1, \text{att}(u_1, v_1), \text{att}(u_2, v_2) \rightarrow \text{bad}$. There exists a $\sigma_1$ such that $\sigma_1$ is grounding for $\text{vars}_{\text{left}}(r_1)$ and $\text{Pre}_1\sigma_1 = \text{Pre}_\sigma, f(u_1^1, u_2^1)\sigma_1 = u\sigma$.

Recall that $u\sigma = C[u_1, \ldots, u_n] = f(C_1[u_1, \ldots, u_m], C_2[u_m+1, \ldots, u_n])$. By induction hypothesis on $C_1$ with $r_1 = \text{Pre}_1', \text{att}(u_1^1, v_1) \rightarrow \text{bad}$, we get that there exists $r_2 \in \text{Flat}(r_1) \subset \text{Flat}(r)$ such that
\[ r_2 = \text{Pre}_2, \text{att}(u_1^2, v_1'), \ldots, \text{att}(u_m^2, v_m') \rightarrow \text{bad} \text{ and a } \sigma_2 \text{ such that } u_i^2 \sigma_2 = u_i \text{ for } i \leq m, \text{ and } \text{Pre}_2 \sigma_2 = \left\langle \text{Pre}_3 \sigma_2 \right\rangle. \]

By induction hypothesis on \( C_2 \) with \( r_2 = \text{Pre}_2, \text{att}(u_2, v_2) \rightarrow \text{bad} \), we get that there exists \( r_3 \in \text{Flat}(r_2) \subseteq \text{Flat}(r) \) such that \( r_3 = \text{Pre}_3, \text{att}(u_{m+1}^3, v_{m+1}'), \ldots, \text{att}(u_n^3, v_n') \rightarrow \text{bad} \) and a \( \sigma_3 \) such that \( u_i^3 \sigma_3 = u_i \) for \( i > m \), and \( \text{Pre}_3 \sigma_3 = \left\langle \text{Pre}_4 \sigma_3 \right\rangle. \)

So we have that \( \text{Pre}_3 \sigma_3 = \left\langle \text{Pre}_4 \sigma_3 \right\rangle \) for some adequate \( \sigma'_3 \) such that \( u_i^3 \sigma'_3 = u_i^2 \sigma_2 \) for each \( i \leq m \). Hence we have that

\[ \text{Pre}_4 \sigma_3, \text{att}(u_1^3, v_1'), \ldots, \text{att}(u_{m+1}^3, v_{m+1}'), \text{att}(u_n^3, v_n') \rightarrow \text{bad}. \]

We deduce that \( \text{Pre}_3 \sigma_3 = \left\langle \text{Pre}_4 \sigma_3 \right\rangle \) and \( u_i = u_i^3 \sigma_3 \) for each \( i \). It concludes the proof of the claim.

Now we want to prove the main result. We proceed by induction on the number \( n \) of contexts. The base case \( n = 1 \) is our claim. Consider the inductive case.

We have \( r = \text{Pre}, \text{att}(u_1^1, v_1), \ldots, \text{att}(u_{n+1}^1, v_{n+1}) \rightarrow \text{bad}. \)

We define \( \text{Pre}' = \text{Pre}, \text{att}(u_{n+1}, v_{n+1}) \rightarrow \text{bad}. \)

We apply our induction hypothesis on \( r = \text{Pre}', \text{att}(u_1^1, v_1), \ldots, \text{att}(u_n, v_n) \rightarrow \text{bad} \) and we get a \( r_1 \in \text{Flat}(r) \) such that \( r_1 = \text{Pre}_1, \text{att}(u_1^1, v_1), \ldots, \text{att}(u_{n+1}^1, v_{n+1}) \rightarrow \text{bad} \) and \( \sigma_1 \) such that \( u_i^1 \sigma_1 = u_i^2 \sigma_2 \) and \( \text{Pre}_2 \sigma_1 = \left\langle \text{Pre}_3 \sigma_1 \right\rangle. \)

We can apply our claim on \( r_1 = \text{Pre}_1, \text{att}(u_{n+1}^1, v_{n+1}) \rightarrow \text{bad} \) for some adequate \( \text{Pre}_1, \text{att}(u_{n+1}, v_{n+1}) \rightarrow \text{bad} \) and \( \sigma_1 \) such that \( u_i^1 \sigma_1 = u_i^2 \sigma_2 \) and \( \text{Pre}_2 \sigma_1 = \left\langle \text{Pre}_3 \sigma_1 \right\rangle. \)

We get a rule \( r_2 = \text{Pre}_2, \text{att}(u_1^2, v_1'), \ldots, \text{att}(u_{n+1}^2, v_{n+1}') \rightarrow \text{bad} \) for any \( u_i^2 \sigma_2 = u_i^1 \sigma_1 \) and \( v_i^1, v_i^2 \) are distinct variables that do not occur elsewhere in \( r_2 \).

Now we write \( r_2 = \text{Pre}_2, \text{att}(u_1^2, v_1'), \ldots, \text{att}(u_{n+1}^2, v_{n+1}') \rightarrow \text{bad} \), where \( \alpha_i, \beta_i = u_i^2 \sigma_2 \) and \( \gamma_i, \delta_i = v_i^2 \sigma_2 \) for each \( i \geq 2 \).

We can apply Lemma 15 with contexts \( C_1, C_2, C_3, \ldots, C_n \). We get a rule

\[ r_3 = \text{Pre}_3, \text{att}(\gamma_1, \delta_1'), \ldots, \text{att}(\gamma_n, \delta_n'), \text{att}(\gamma_1', \delta_1'), \ldots, \text{att}(\gamma_n', \delta_n'). \]

and a substitution \( \sigma_3 \) where \( \text{Pre}_3 \sigma_3 = \left\langle \text{Pre}_2 \sigma_2 \right\rangle \) and \( u_i^2 \sigma_2 = C_i' \gamma_i^1, \ldots, \gamma_i^n \sigma_3 \) for each \( i \in \{1, 2\} \), and \( \alpha_i, \beta_i = C_i' \gamma_i^1, \ldots, \gamma_i^n \sigma_3 \) for each \( i \geq 2 \) and the \( \gamma_i, \delta_i' \) are any terms.

We have:

\[ u_i^1 \sigma_1 = \alpha_i \sigma_2 \text{ for each } i \geq 2, \text{ we have:} \]

\[ \text{Pre}_2 \sigma_2 = \left\langle \text{Pre}_1 \sigma_2 \right\rangle, \text{att}(u_2^2, v_2'), \ldots, \text{att}(u_n^2, v_n') \sigma_2 \sigma_1 = \left\langle \text{Pre}_2 \sigma_2 \right\rangle, \text{att}(\alpha_2, \beta_2) \sigma_2, \ldots, \text{att}(\alpha_n, \beta_n) \sigma_2. \]
so $\text{Pre}_1\sigma_1 = \text{left} \, \text{Pre}_2\sigma_2 = \text{left} \, \text{Pre}_3\sigma_3$. As $\text{Pre}_1\sigma_1 = \text{Pre}_2\sigma_2$ we get $\text{Pre}_3 = \text{left} \, \text{Pre}_3\sigma_3$. It concludes the proof. □

Lemma 2: Let $\mathcal{P}$ be a protocol type-compliant w.r.t. $(T_p, \delta_p)$, and $Q$ be another protocol. Let $\Theta$ be the following planning system:

$$\langle \text{Fact}_0, \text{Fact}(P, Q), \mathcal{R} \rangle$$

where $\mathcal{R} = \text{Concrete}^+(\text{Rule}_A \cup \text{Flat}(\text{Rule}(P, Q)))$.

Let $(tr, \phi) \in \text{trace}_2(P)$ for some $\phi$ and such that:

- $tr$ only contains simple recipes;
- $(tr, \phi)$ is well-typed w.r.t. $(T_p, \delta_p)$;
- $(tr, \psi) \in \text{trace}_3(Q)$ for some $\psi$.

Then, there exist a planning path $r_1, \ldots, r_n$ of some length $n$ from $\text{Fact}(P, Q)$ to some $S_n$ such that $\text{Fact}(K_p', K_q') \uparrow S_n$ where $K_p'$ (resp. $K_q'$) is the resulting configuration starting from $P$ (resp. $Q$) and executing $tr$.

Conversely, let $r_1, \ldots, r_n$ be a planning path from $\text{Fact}(P, Q)$ to $S_n$ such that $bad \notin S_n$. Then, there exist a trace $tr$, and frames $\phi$ and $\psi$ such that:

- $tr$ only contains simple recipes;
- $(tr, \phi)$ is well-typed w.r.t. $(T_p, \delta_p)$;
- $(tr, \psi) \in \text{trace}_3(Q)$ for some $\psi$; and
- $\text{Fact}(K_p', K_q') \uparrow S_n$ where $K_p'$ (resp. $K_q'$) is the resulting configuration starting from $P$ (resp. $Q$) and executing $tr$.

Proof. ($\Rightarrow$) We show this result by induction on the length of $tr$.

Base case. The trace is empty, and the empty planning path can be used to establish the result.

Inductive case. $tr = tr'.\alpha$ with $\alpha = \text{out}(c,w)$ or $\alpha = \text{in}(c,R)$. We apply our induction hypothesis on $tr'$ and we obtain a planning path $r_1, \ldots, r_n$ from $\text{Fact}(K_p', K_q')$ to some $S_n$. Let $K_p'$ (resp. $K_q'$) the resulting configuration starting from $P$ (resp. $Q$) and executing $tr'$. Similarly, let $K_p''$ (resp. $K_q''$) the resulting configuration starting from $P$ (resp. $Q$) and executing $tr'.\alpha$. Thanks to our induction hypothesis, we have that $\text{Fact}(K_p', K_q') \uparrow S_n$. We have also that $K_p' \xrightarrow{\tau_p} K_p''$ and $K_q' \xrightarrow{\tau_q} K_q''$, and therefore $K_p' = (P'; \sigma_p; \phi')$ contains a simple process of the form $P = \text{out}(c,u).P'$ and $K_q' = (Q'; \sigma_q' \psi')$ contains a simple process of the form $Q = \text{out}(c,v).Q'$ (and similarly in case of an input). Moreover, we know that $\sigma_p$ and $\sigma_q'$ are messages.

Let $R$ be the abstract protocol rule corresponding to this step.

We consider the case of an output. We have that $R \in \text{Rule}(P, Q)$ and this rule is of the form:

$$\text{St}(P, Q) \rightarrow \text{att}(u, v), \text{St}(P', Q'); \text{St}(P, Q)$$

Now, we consider the concrete instance that corresponds to the execution mentioned above, i.e. the one obtained by applying $\sigma_p' \cup \sigma_q'$. This will allow us to conclude.

We now consider the case of an input, i.e. $P = \text{in}(c,u).P'$ and $Q = \text{in}(c,v).Q'$. We have that $R \in \text{Rule}(P, Q)$ and this rule is of the form:

$$\text{St}(P, Q), \text{att}(u, v) \rightarrow \text{St}(P', Q'); \text{St}(P, Q)$$

We know that $R_\phi \downarrow = u\sigma_p$ and $R_\psi \downarrow = v\sigma_q'$ with $R$ a simple recipe. $R$ is a constructor on destructor recipe, so there is a constructor context $C$ such that $R = C[R_1, \ldots, R_n]$ where $R_1, \ldots, R_n$ are destructor-only recipes. So $u\sigma_p = C[R_1\phi \downarrow, \ldots, R_n\phi \downarrow]$ and $v\sigma_q' = C[R_1\psi' \downarrow, \ldots, R_n\psi' \downarrow]$. So it is possible to apply Lemma 12. There exists a rule $r'$ in $\text{Flat}(r)$:

$$r' = \text{Pre}', \text{att}(u_1', v_1'), \ldots, \text{att}(u_n', v_n') \rightarrow \text{Add}', \text{Del}'$$

and $\sigma'$ a grounding substitution for $r'$ such that:

1) $\delta_p(x\sigma') \leq \delta_p(x)$ for any $x \in \text{vars}(r')$;
2) $(\text{Pre}', \text{Add}', \text{Del}')\alpha = (\text{Pre}, \text{Add}, \text{Del})(\sigma_p \cup \sigma_q')$; and
3) $\text{att}(u_1', v_1'), \ldots, (u_n', v_n') = \text{att}(C[u_1', \ldots, u_n', C[v_1', \ldots, v_n'])\sigma'$.\n
$R_1, \ldots, R_n$ are destructor-only recipes such that $\text{att}(R_1\phi \downarrow, R_1\psi' \downarrow)$ unifies with $\text{Pre}(r')$. Thanks to Lemma 1, for each $R_i$ there is an associated plan $\pi_i$ containing only adversary rules such that $\text{att}(R_i\phi \downarrow, R_i\psi' \downarrow)$ is in the final state of $\pi_i$. As the adversary rules delete nothing, these plans are composable together. They give $\pi = \pi_1, \ldots, \pi_n$. As the $\text{att}(R_i\phi \downarrow, R_i\psi' \downarrow)$ unify with $\text{Pre}(r')$, there is a rule $r''$ in $\text{Concrete}^+(r')$ such that its preconditions are exactly the $\text{att}(R_i\phi \downarrow, R_i\psi' \downarrow)$.

This allows us to conclude.

($\Leftarrow$) We show this result by induction on the length of the planning path.

Base case. Obvious.

Inductive case. We have a planning path $r_1, \ldots, r_n$. Thanks to our induction hypothesis, we know that the result holds for $r_1, \ldots, r_{n-1}$ and therefore the existence of a trace $tr$. Then, we distinguish several cases depending on the rule $r_n$. In case $r_n$ is an instance of $\text{Concrete}^+(\text{Rule}_A)$, we consider $tr$ again. The case where $r_n$ is a rule that adds bad is impossible since $bad \notin S_n$. Now, if $r_n$ is an instance of an abstract rule in $\text{Flat}(\text{Rule}(P, Q))$. Let $R_f$ be the flattened abstract rule, and $R$ the abstract protocol rule.

In case $R$ is a rule corresponding to the case of an output, then $r_n$ is an instance of $R$ since the flattening does not produce any other rule. In such a case, we can mimic this step by considering $tr.\text{out}(c,w)$.

In case $R$ is a rule corresponding to an input, then $r_n$ is an instance of a rule $R_f \in \text{Flat}(R)$. We have that $R_f$ is of the form:

$$\text{St}'_{P,Q}(\theta_p, \theta_q), \text{att}(u_1, v_1), \ldots, \text{att}(u_k, v_k) \rightarrow \text{St}'_{P', Q'}(\theta_p, \theta_q)$$

and $r_n$ is an instance of $R_f(\sigma_p \cup \sigma_q)$ where $\sigma_p$ (resp. $\sigma_q$) is the substitution obtained after executing $tr$. Let $\tau_p$ and $\tau_q$ be grounding substitution such that $r_n = (R_f(\sigma_p \cup \sigma_q))(\tau_p \cup \tau_q)$.\n
29
We know that there exist destructor only recipes $R_1, \ldots, R_k$ such that $R_1 \phi_1 = u_1 \sigma \circ_P$ and $R_i \psi_i = v_i \sigma \circ_Q$ by Lemma 1. We can apply Lemma 13 on rule $r_n$ written as:

$$r_n = \text{state}, \text{att}(u_1, v_1), \ldots, \text{att}(u_n, v_n) \rightarrow \text{Add}; \text{Del}$$

In the case where bad-flat $\not\in \text{Add}$, there exists a constructor context $C$ and a substitution $\tau$ such that $R_1(\sigma \circ P \cup \sigma \circ Q) \tau = \text{state}, \text{att}(u, v) \rightarrow \text{Add}; \text{Del}$ where $u = C[u_1, \ldots, u_n]$ and $v = C[v_1, \ldots, v_n]$.

Therefore, consider the trace $\text{tr} \cdot \text{in}(c, C[R_1, \ldots, R_k])$. This step can be done on the $P$ side, as well as in the $Q$ side. Hence, the result.

**Theorem 3:** Let $\mathcal{P}$ a protocol type-compliant w.r.t. $(\mathcal{T}_P, \delta_P)$, and $\mathcal{Q}$ be another protocol. We consider the following set $\mathcal{R}$ of concrete rules:

$$\mathcal{R} = \text{Concrete}(\text{Rule}_A \cup \text{flat}(\text{Rule}(\mathcal{P}, \mathcal{Q}))) \cup \mathcal{R}_\text{test} \cup \mathcal{R}_\text{atom}$$

Let $\Theta = \langle \text{Fact}_A, \text{Fact}(\mathcal{P}, \mathcal{Q}), \mathcal{R} \rangle$ and $\Pi = \langle \Theta, \{\text{bad}\} \rangle$. We have that $\mathcal{P} \not\subseteq \mathcal{Q}$ if, and only if, $\Pi$ has a solution.

**Proof.** We show the two directions separately.

$(\Rightarrow)$ In case $\mathcal{P} \not\subseteq \mathcal{Q}$, we know thanks to Theorem 2 that there exists a witness of this fact such that $(\text{tr}, \phi)$ exists in $\text{trace}(\mathcal{P})$ is quasi-well-typed, only involves the constants we consider here. Moreover, $\text{tr}$ is made of simple recipes. We consider such a witness of minimal length, and we distinguish two cases depending on the fact that $(\text{tr}, \psi)$ exists in $\text{trace}(\mathcal{Q})$ for some $\psi$ or not. If not, we let $\text{tr}^{-1}$ the trace $\text{tr}$ without its last element, otherwise $\text{tr}^{-1} = \text{tr}$.

Lemma 2 allows us to conclude that there exists a planning path $r_1, \ldots, r_n$ from $\text{Fact}(\mathcal{P}, \mathcal{Q})$ to $S_n$ such that $\text{Fact}(K', K'_Q) \uparrow S_n$ where $K'$ (resp. $K'_Q$) is the resulting configuration starting from $\mathcal{P}$ (resp. $\mathcal{Q}$) and executing $\text{tr}^{-1}$.

In case $(\text{tr}, \psi)$ exists in $\text{trace}(\mathcal{Q})$, we know that $\phi \not\subseteq \mathcal{Q}$, and thanks to Proposition 1, we will obtain a planning path that we can concatenate to $r_1, \ldots, r_n$ to conclude.

Otherwise, we have that $(\text{tr}^{-1}, \psi^{-1})$ exists in $\text{trace}(\mathcal{Q})$ but the last action $\alpha$ can not be performed. In case $\alpha = \text{out}(c, w)$. If such an action can not be performed, it means that this action is not available in the process or would lead to output a term that is not a message. In the first case, we have an abstract protocol rule $\mathcal{R}$ that can be instantiated to mimick this step. In the second case, we have to consider the instance in $\text{Concrete}^{-}(\mathcal{R})$. Note that for such a rule $\text{Flat}(\mathcal{R}) = \mathcal{R}$.

Now, in case $\alpha = \text{in}(c, \mathcal{R})$. If such an action can not be performed, it means that either this action is not syntactically available in the process or the term in the $Q$ side does not match. In both case, we have an abstract protocol rule in $\mathcal{R}$ that corresponds to this step.

First of all, remind that $\mathcal{R}$ is a simple recipe: $R = C[R_1, \ldots, R_k]$ where $R_1, \ldots, R_k$ are destructor-only recipes and $C$ a constructor context built on $\Sigma_c$ only. Thanks to Lemma 1, for each $R_i$ there is an associated plan $\pi_i$ containing only adversary rules such that $\text{att}(R_i \phi_i, R_i \psi_i)$ is in the final state of $\pi_i$. As the adversary rules delete nothing, these plans are composable together. They give $\pi = \pi_1, \ldots, \pi_k$. We call $S_n'$ the resulting state.

If the input is not syntactically available in the process, then $\mathcal{R}_f$ is of the form $\text{St}(\mathcal{P}, \mathcal{Q}), \text{att}(u, y) \rightarrow \text{bad-protod}$ where $y$ is a fresh variable. $\text{St}(\mathcal{P}, \mathcal{Q})$ unifies with some fact of $S_n'$ by induction hypothesis (we arrived at this step). So we call $\sigma$ the substitution such that:

- $\text{St}(\mathcal{P}, \mathcal{Q}) \sigma \in S_n'$
- $R_\phi = u \sigma$
- $\delta_P(x \sigma) \leq \delta_P(x)$ for any $x \in \text{vars}_{\text{left}}(\mathcal{R}_f)$
- $R_\psi = y \sigma$

There exists such a $\sigma$ because we arrived at step (induction hypothesis: item 1), the input passes in the $P$ side (item 2,3) and $y$ is a fresh variable.

We can apply Lemma 12. We get a rule $r' \in \text{Flat}(\mathcal{R}_f)$:

$$r' = \text{Pre}', \text{att}(u_1', v_1'), \ldots, \text{att}(u_k', v_k') \rightarrow \text{Add}', \text{Del}'$$

and a grounding substitution $\sigma'$ such that:

- $\delta_P(x \sigma') \leq \delta_P(x)$ for any $x \in \text{vars}_{\text{left}}(r')$
- $(\text{Pre}', \text{Add}', \text{Del}') \sigma' = (\text{Pre}, \text{Add}, \text{Del}) \sigma'$

and $\sigma'$ such that:

1. $\delta_P(x \sigma') \leq \delta_P(x)$ for any $x \in \text{vars}_{\text{left}}(r')$
2. $(\text{Pre}', \text{Add}', \text{Del}') \sigma' = (\text{Pre}, \text{Add}, \text{Del}) \sigma'$; and
3) \(\text{att}(u, v)\sigma = \text{att}(C[u_1', \ldots, u_n'], C[v_1', \ldots, v_n'])\sigma'.\)

We have that there is some fact unifying at left with \(Pr'\sigma' = Pr\sigma\) in \(S_n\) by induction hypothesis, and the \(R_i\phi\downarrow\) unify with the \(u_i\), but the \(R_i\psi\downarrow\) do not unify with the \(v_i\) by hypothesis. So there is a \(r'\in\text{Concrete}^{-}(r')\) whose precondition are true. It concludes this case and therefore the proof.

\((\Leftarrow)\) We show this result by induction on the length of planning path leading to bad. We consider one of minimal length. We have that \(\pi = r_1, \ldots, r_n\), and we can apply Lemma 2 on \(r_1, \ldots, r_{n-1}\), we obtain \(tr\). This trace allows us to reach the configurations \((P', \sigma_{P'}, \phi')\) and \((Q', \sigma_{Q'}, \psi')\). We call \(S_{n-1}\) the set of fluents resulting from this planning path, and \(S_n\) the set resulting from \(r_1, \ldots, r_n\). Now, we distinguish several cases depending on whether \(r_n\) is in \(\text{Concrete}^{-}(\text{Rule}_n)\cup\mathcal{R}_{\text{test}}\cup\mathcal{R}_{\text{atom}}\) or in \(\text{Concrete}^{-}((\text{Rule}(P, Q))\). In the first case, we conclude relying on Proposition 1, and therefore we obtain that the frames resulting from the execution of \(tr\) are not in static inclusion.

The second case occurs when \(r_n\in\text{Concrete}^{-}(\text{Rule}(P, Q))\). So there is a \(R_f\) such that \(r_n\in\text{Concrete}^{-}(\text{Rule}(R_f))\).

This rule cannot come from the first item of the protocol rules definition, so it comes either from the second or from the third.

We first consider the case where it comes from the second. Then, there is no flattening on those rules. Therefore: \(r_n\in\text{Concrete}^{-}(R_f)\). We write:

\[R_f = St(P, Q) \rightarrow \text{Add}; \text{Del}\]

\[r_n = f_0 \rightarrow \text{Add}_n; \text{Del}_n\]

for some \(\text{Add}, \text{Del}, \text{Add}_n, \text{Del}_n\) sets of facts and \(f_0\in S_{n-1}\). As \(r_n\in\text{Concrete}^{-}(R_f)\) is applicable, \(St(P, Q)\) unifies with \(f_0\in S_{n-1}\) by induction hypothesis: call \(\sigma\) a substitution such that \(St(P, Q)\sigma = f_0\). \(\text{Add}, \text{Del}\) unifies at left with \((\text{Add}_n, \text{Del}_n)\) through \(\sigma\). If it unifies at right, then \(r_n\in\text{Concrete}^{-}(R_f)\). Therefore \(Del = \emptyset\) and \(Add = \text{att}(u, c_0')\), bad-proto. \(u\) must be instanciated by a message as \(r_n\) exists and is applicable. So in \(P'\) there is a process \(P\) on some channel \(c\) that begins with an output but the process in \(Q'\) on channel \(c\) does not begin by an output. Moreover, it is possible to make this output, so \(tr.\text{out}(c, w)\) is a witness of non-inclusion.

If it does not unify at right, then \(r_n\in\text{Concrete}^{-}(R_f)\). Assume \(R_f = St(P, Q) \rightarrow \text{att}(u, c_0'), \text{bad-proto}\). Then by definition of \(\text{Concrete}^{-}\) we deduce that \(u\sigma\) is a message but \(c_0'\sigma\) is not. So this case never happens. We get that \(R_f = St(P, Q) \rightarrow \text{att}(u, v), St(P', Q'); St(P, Q)\). As \(St(P, Q)\sigma\in S_{n-1}\), we must have that \(u\sigma\) is a message but \(v\sigma\) is not. So the output is possible in the \(P\) side, but not in the \(Q\) side. This concludes the case of the second item of protocol rules definition.

Only the third item remains to be considered. We write:

\[R_f = St(P, Q), \text{att}(u, v) \rightarrow \text{Add}; \text{Del}\]

\[r_n = f_0, \text{att}(t_1, t_1'), \ldots, \text{att}(t_k, t_k') \rightarrow \text{Add}_n; \text{Del}_n\]

where \(f_0\) unifies at left with \(St(P, Q)\) and the \(t_i, t_i'\) are messages. We have that \(bad\in Add_n\). As \(r_n\) is applicable, there are recipes \(R_1, \ldots, R_k\) such that \(R_i\phi\downarrow\in t_i\) and \(R_i\psi\downarrow = t_i'\) for each \(i\) by Lemma 1.

Moreover, the form of the rule indicates that there is some process \(P = in(c, u). P' = P'\in P'\) and \(Q\) is the process on channel \(c\) in \(Q'\). We distinguish three cases according to the origin of this bad:

**First case.** It is bad,proto. Then the only possibility is that \(R_f = St(P, Q), \text{att}(u, x) \rightarrow \text{bad-proto where } x\) is a fresh variable. We have \(r_n\in\text{Concrete}^{-}(\text{Flat}(R_f))\) (\(\text{Concrete}^{-}\) would give bad-concrete) so there is a rule \(r' = f_0', \text{att}(u_1, v_1), \ldots, \text{att}(u_k, v_k) \rightarrow \text{Add}; \text{Del}' \in \text{Flat}(R_f)\) such that \(r_n\in\text{Concrete}^{-}(r')\). In particular there is a \(\sigma\) such that \(r_n = \sigma r_0\) and \(f_0' \sigma_0 = f_0, u \sigma_0 = t_i\) and \(v \sigma_0 = t_i'\) for each \(i\). So \(bad-flat \notin \text{Add}'\). We apply Lemma 13, there is a constructor context \(C\) and a substitution \(\tau\) such that \(R_f\tau = f_0', \text{att}(u, v) \rightarrow \text{Add}; \text{Del} and u = C[u_1, \ldots, u_k]\) and \(v = C[v_1, \ldots, v_k]\).

The \((t_i, t_i')\) are in \(S_{n-1}\). So by Lemma 1, there are destructor only recipes \(R_1, \ldots, R_k\) such that \(R_i\phi\downarrow = t_i\) and \(R_i\psi\downarrow = t_i'\).

So it means that \(tr.in(c, C[R_1, \ldots, R_k])\) is a trace of \(P\) but it is not a trace of \(Q\) because there is no input in \(Q'\).

**Second case.** It is bad-flat. We have \(r_n\in\text{Concrete}^{-}(\text{Flat}(R_f))\) (\(\text{Concrete}^{-}\) would give bad-concrete) So there is a rule

\[r' = f_0', \text{att}(u_1, v_1), \ldots, \text{att}(u_k, v_k) \rightarrow \text{bad-flat}; \emptyset \in \text{Flat}(R_f)\]

and a substitution \(\sigma\) such that \(r_n = r'\sigma\). So we apply Lemma 13, and we get that there exists a constructor context \(C\) a substitution \(\tau\), a term \(v'\) and two sets \(\text{Add}_0\) and \(\text{Del}_0\) such that

\[R_f\tau = f_0', \text{att}(C[u_1, \ldots, u_n], v') \rightarrow \text{Add}_0; \text{Del}_0\]

but \(v'\) does not unify with \(C\).

So \(tr.in(c, C[R_1, \ldots, R_k])\) is a trace of \(P\), but it is not a trace of \(Q\) (either there was no input in \(Q\) or there is an input but it does not unify with \(C[R_1, \ldots, R_k]v'\)). It gives a witness of non-inclusion.

**Third case.** It is bad-concrete. We have \(r_n\in\text{Concrete}^{-}(\text{Flat}(R_f))\) (\(\text{Concrete}^{-}\) would not give bad-concrete). So there is a rule

\[r' = f_0', \text{att}(u_1, v_1), \ldots, \text{att}(u_k, v_k) \rightarrow \text{Add}; \text{Del}' \in \text{Flat}(R_f)\]

for some \(\text{Add}', \text{Del}'\) and a substitution \(\sigma\) such that \(r_n = r'\sigma\), but \(r_n\) and \(r'\) do not unify at right.

In both cases of Lemma 13, we get a constructor context \(C\), a substitution \(\tau\) and two sets \(\text{Add}_0, \text{Del}_0\) such that
\[ R_{fT} = St(P, Q), \text{att}(u', v') \rightarrow Add_0; Del_0 \text{ where } u = C[u_1, \ldots, u_k]. \]

So \( \text{tr.in}(c, C[R_1, \ldots, R_k]) \) is a valid trace of \( P \) (we have
\( C[R_1, \ldots, R_k] \phi' \downarrow = u' = u \tau). \)

Moreover, \( C[R_1 \psi' \downarrow, \ldots, R_k \psi' \downarrow] \) does not unify with \( v \), (as \( r_n \in \text{Concrete}^{-}(r') \)). So \( \text{tr.in}(c, C[R_1, \ldots, R_k]) \) is not a trace of \( Q \), which gives a witness of non-inclusion.

It concludes the proof. \( \square \)