DEEPSEC: Deciding Equivalence Properties in Security Protocols - Theory and Practice

Vincent Cheval, Steve Kremer, Itsaka Rakotonirina

DEEPSEC: Deciding Equivalence Properties in Security Protocols Theory and Practice

Vincent Cheval, Steve Kremer, Itsaka Rakotonirina
Inria & LORIA

***Abstract*—Automated verification has become an essential part in the security evaluation of cryptographic protocols. Recently, there has been a considerable effort to lift the theory and tool support that existed for reachability properties to the more complex case of equivalence properties. In this paper we contribute both to the theory and practice of this verification problem. We establish new complexity results for static equivalence, trace equivalence and labelled bisimilarity and provide a decision procedure for these equivalences in the case of a bounded number of sessions. Our procedure is the first to decide trace equivalence and labelled bisimilarity exactly for a large variety of cryptographic primitives—those that can be represented by a subterm convergent destructor rewrite system. We implemented the procedure in a new tool, DEEPSEC. We showed through extensive experiments that it is significantly more efficient than other similar tools, while at the same time raises the scope of the protocols that can be analysed.**

1. Introduction

The use of automated, formal methods has become indispensable for analysing complex security protocols, such as those for authentication, key exchange and secure channel establishment. Nowadays there exist mature, fully automated such analysers; among others AVISPA [11], ProVerif [17], Scyther [33], Tamarin [45] or Maude-NPA [44]. These tools operate in so-called *symbolic* models, rooted in the seminal work by Dolev and Yao [35]: the attacker has full control of the communication network, unbounded computational power, but cryptography is idealised. This model is well suited for finding attacks in the protocol logic, and tools have indeed been extremely effective in discovering this kind of flaw or proving their absence.

While most works investigate *reachability* properties, a recent trend consists in adapting the tools—and the underlying theory—for the more complex *indistinguishability* properties. Such properties are generally modelled as a behavioural equivalence (bisimulation or trace equivalence) in a dedicated process calculus such as the Spi [6] or applied pi calculus [5]. A typical example is real-or-random secrecy: after interacting with a protocol, an adversary is unable to distinguish the *real* secret used in the protocol from a *random* value. Privacy-type properties can also be expressed as such: anonymity may be modeled as the adversary's inability to distinguish two instances of a protocol executed by different agents; vote privacy [34] has been expressed as indistinguishability of the situations where the votes of two agents have been swapped or not; unlinkability [8] is seen as indistinguishability of two sessions, either both executed by the same agent A, or by two different agents A and B.

Related work. The problem of analysing security protocols is undecidable in general but several decidable subclasses have been identified. While many complexity results are known for trace properties [36], [42], the case of behavioural equivalences remains mostly open. When the attacker is an eavesdropper and cannot interact with the protocol, the indistinguishability problem—*static equivalence*—has been shown PTIME for large classes of cryptographic primitives [3], [27], [29]. For active attackers, bounding the number of protocol sessions is often sufficient to obtain decidability [42] and is of practical interest: most real-life attacks indeed only require a small number of sessions. In this context Baudet [14], and later Chevalier and Rusinowtich [24], showed that real-or-random secrecy was coNP for cryptographic primitives that can be modelled as subterm convergent rewrite systems, by checking whether two constraint systems admit the same set of solutions. These procedures do however not allow for else branches, nor do they verify trace equivalence in full generality. In [23], Cheval et al. have used Baudet's procedure as a black box to verify trace equivalence of *determinate* processes. This class of processes is however insufficient for most anonymity properties. Finally, decidability results for an unbounded number of sessions were proposed in [26], [25], but with severe restrictions on processes and equational theories.

Tool support also exists for verifying equivalence properties. We start discussing tools that are limited to a bounded number of sessions. The SPEC tool [46], [47] verifies a sound symbolic bisimulation, but is restricted to particular cryptographic primitives (pairing, encryption, signatures and hash functions) and does not allow for else branches. The APTE tool [20] covers the same primitives but allows else branches and decides trace equivalence exactly. On the contrary, the AKISS tool [19] allows for user-defined cryptographic primitives. Partial correctness of AKISS is shown for primitives modelled by an arbitrary convergent rewrite system that has the finite variant property [28]. Termination is additionally shown for subterm convergent rewrite systems. However, AKISS does only decide trace equivalence

for a class of determinate processes; for other processes trace equivalence can be both over- and under-approximated which proved to be sufficient on many examples. The recent SAT-EQUIV tool [30] uses a different approach: it relies on Graph Planning and SAT solving to verify trace equivalence, rather than a dedicated procedure. The tool is extremely efficient and several orders of magnitude faster than other tools. It does however not guarantee termination and is currently restricted to pairing and symmetric encryption and only considers a class of *simple processes* (a subclass of determinate processes) that satisfy a type-compliance condition. These restrictions severely limit its scope.

Other tools support verification of equivalence properties, even for an unbounded number of sessions. This is the case of ProVerif [15], Tamarin [13] and Maude NPA [44] which all allow for user-defined cryptographic primitives. However, given that the underlying problem is undecidable, these tools may not terminate. Moreover, they only approximate trace equivalence by verifying the more fine-grained *diff-equivalence*. This equivalence is too fine-grained on many examples. While some recent improvements on ProVerif [21], [16] helps covering more protocols, general verification of trace equivalence is still out of scope. For instance, the verification by Arapinis et al. [10] of unlinkability in the 3G mobile phone protocols required some "tricks" and approximations of the protocol to avoid false attacks. In [31], Cortier et al. develop a type system and automated type checker for verifying equivalences. While extremely efficient, this tool only covers a fixed set of cryptographic primitives (the same as SPEC and APTE) and verifies an approximated equivalence, similar to the diff-equivalence. A different approach has been taken by Hirschi et al. [38], identifying sufficient conditions provable by ProVerif for verifying unlinkability properties, implemented in the tool Ukano, a front-end to the ProVerif tool. Ukano does however not verify equivalence properties in general.

Contributions. We significantly improve the theoretical understanding and the practical verification of equivalence when the number of protocol sessions is bounded. We emphasise that even in this setting, the system under study has an infinite state space due to the term algebra modelling cryptographic primitives. Our work targets the wide class of cryptographic primitives that can be represented by a subterm convergent rewriting system. Concretely, we provide

1) new tight complexity results for static equivalence ($\sim$), trace equivalence ($\approx_t$) and labelled bisimilarity ($\approx_\ell$);
2) a novel procedure for deciding trace equivalence and labelled bisimilarity for the class of cryptographic primitives modelled by a destructor subterm convergent rewrite system;
3) an implementation of our procedure for trace equivalence in a new tool called DEEPSEC (DEciding Equivalence Properties for SECurity protocols).

We detail the three contributions below.

Complexity. We provide the first complexity results for deciding trace equivalence and labelled bisimilarity in the applied pi calculus, without any restriction on the class of protocols (other than bounding the number of sessions). In particular, our results are not restricted to determinate processes, allow for else branches and do not approximate equivalence. Let us also highlight one small, yet substantial difference with existing work: we do not consider cryptographic primitives (rewrite systems) as constants of the problem. As most modern verification tools allow for user-specified primitives [17], [45], [44], [19], our approach seems to better fit this reality. Typically, all existing procedures for static equivalence can only be claimed PTIME because of this difference and are actually exponential in the sizes of signature or equational theory. Our complexity results are summarised in fig. 1. All our lower bounds hold for subterm convergent rewrite systems (destructor or not) and even for the positive fragment (without else branches). *En passant*, we present results for the pi calculus[1]: although investigated in [18], complexity was unknown when restricted to a bounded number of sessions. Still, our main result is the coNEXP completeness (and in particular, the decidability) of trace equivalence and labelled bisimilarity for destructor subterm convergent rewrite systems.

	Pure pi calculus	Applied pi calculus (destr.) subterm convergent
$\sim$	LOGSPACE	coNP complete
$\approx_t$	Π_2 complete	coNEXP complete
$\approx_\ell$	PSPACE complete	coNEXP complete

Figure 1: Summary of complexity results.

Decision procedure. We present a novel procedure based on a symbolic semantics and constraint solving. Unlike most other work, our procedure decides equivalences *exactly*, i.e. without approximations. Moreover, it does not restrict the class of processes (except for replication), nor the use of else branches, and is correct for any cryptographic primitives that can be modelled by a subterm convergent destructor rewrite system (see section 2). The design of the procedure did greatly benefit from our complexity study, and was developed in order to obtain tight complexity upper bounds.

Tool implementation. We implemented our procedure for trace equivalence in a new tool, DEEPSEC. While still a prototype, DEEPSEC was carefully engineered. The tool output is available in pretty printed html format and allows to step through an attack, if any is found. DEEPSEC can also distribute the computation, thus exploiting multicore architectures or clusters of computers to their fullest. Finally, we integrated several classical optimisations for trace-equivalence analysis, e.g. *partial order reductions* (POR) [12]. This has appeared to reduce the search space dramatically, making the tool scale well in practice despite the high theoretical complexity (coNEXP).

Through extensive benchmarks, we compare DEEPSEC to other tools limited to a bounded number of protocol sessions: APTE, SPEC, AKISS and SAT-EQUIV. Our tool is sig-

1. These results are detailed in the full version [1].

nificantly more efficient—by several orders of magnitude—than APTE, SPEC and AKISS, even though DEEPSEC covers a strictly larger class of protocols than APTE and SPEC. Besides, its performance are comparable to SAT-EQUIV, which still outperforms DEEPSEC when the number of parallel processes significantly increase. This gap in performance seems unavoidable as DEEPSEC operates on a much larger class of protocols (more primitives, else branches, no limitation to simple processes, termination guaranteed).

Part of the benchmarks consists of classical authentication protocols and focuses on demonstrating scalability of the tool when augmenting the number of parallel protocol sessions. The other examples include more complex protocols, such as Abadi and Fournet's anonymous authentication protocol [5], the protocols implemented in the European passport [37], the AKA protocol used in 3G mobile telephony, as well as the Prêt-à-Voter [43] and the Helios [7] e-voting protocols.

Additional details and proofs are given in the companion technical report [2]. Implementation-related files are freely available at [1].

2. Model

First, we present our model of cryptographic protocols which is mostly inspired from the applied pi calculus [4].

2.1. Messages and cryptographic primitives

Data as terms. Cryptographic operations are modelled by symbols of fixed arity $\mathcal{F} = \{\mathsf{f}/n, \mathsf{g}/m, \dots\}$ forming a finite *signature*. We partition $\mathcal{F}$ in two sets:

- *constructors* $\mathcal{F}_c$*:* model cryptographic constructions (encryption, signature, hash, ...);
- *destructors* $\mathcal{F}_d$*:* model inversions or operations that may fail depending on the structure of their argument (decryption, signature verification, ...).

Example 1. The signature $\mathcal{F} = \mathcal{F}_c \cup \mathcal{F}_d$ defined below models standard cryptographic primitives: symmetric encryption (senc and sdec), asymmetric encryption (pk, aenc and adec), concatenation ($\langle\,\rangle$, proj_1 and proj_2) and hash (h).

$$\mathcal{F}_c = \{\mathsf{senc}/2,\ \mathsf{aenc}/2,\ \mathsf{pk}/1,\ \langle\cdot,\cdot\rangle/2,\ \mathsf{h}/1\}$$
$$\mathcal{F}_d = \{\mathsf{sdec}/2,\ \mathsf{adec}/2,\ \mathsf{proj}_1/1,\ \mathsf{proj}_2/1\}$$

Function symbols are naturally intended to be applied to some arguments. Atomic data—typically communicating channels, randomness, keys—are modelled by an infinite set of so-called *names* $\mathcal{N} = \{a, b, c, \dots\}$. This provides an abstraction of low-level data whose structure is not relevant at the protocol level. To separate *public* from *secret* data, we partition names into two sets $\mathcal{N} = \mathcal{N}_{\mathsf{pub}} \uplus \mathcal{N}_{\mathsf{prv}}$. As usual we define *terms* as the smallest set containing $\mathcal{N}$ and closed under application of symbols to other terms. E.g. if $k \in \mathcal{N}_{\mathsf{prv}}$ models a decryption key, $\mathsf{aenc}(m, \mathsf{pk}(k))$ models the ciphertext obtained after encrypting m with the corresponding public key. The set of terms built from atoms in N by applying functions of F is denoted by $\mathcal{T}(\mathsf{F}, \mathsf{N})$.

Behaviours as rewriting. The behaviour of symbols is modelled by rewriting. For that, we assume an infinite set of variables $\mathcal{X} = \{x, y, z, \dots\}$ whose elements may be used as atoms in terms. A *substitution* σ is a mapping from variables to terms, homomorphically extended to a mapping from terms to terms. Postfix convention $t\sigma$ instead of $\sigma(t)$ and set notation $\sigma = \{x_1 \mapsto \sigma(x_1); \dots; x_n \mapsto \sigma(x_n)\}$ are the norm; in particular we use set operators $\cup$ and $\subseteq$ for domain extension and the extension ordering, respectively.

A *rewriting system* $\mathcal{R}$ is then a finite binary relation on terms. A pair $(\ell, r) \in \mathcal{R}$ is called a *rewriting rule*, written $\ell \to r$ and assumed to verify $\ell \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and $r \in \mathcal{T}(\mathcal{F}_c, vars(\ell))$. By extension, we also use notation $t \to s$ ("t rewrites to s") when t and s are related by the closure of $\mathcal{R}$ under substitution and term context. The reflexive transitive closure of this relation is written $\to^\star$.

Example 2. This rewriting system defines the behaviors of the previously-introduced primitives:

$$\mathsf{sdec}(\mathsf{senc}(x, y), y) \to x \qquad \mathsf{proj}_1(\langle x, y\rangle) \to x$$
$$\mathsf{adec}(\mathsf{aenc}(x, \mathsf{pk}(y)), y) \to x \qquad \mathsf{proj}_2(\langle x, y\rangle) \to y$$

The absence of rules for hash h models one-wayness.

Rewriting is however Turing-complete and restrictions are needed to get decidability results. The first natural limitation is to consider only *convergent* systems—ensuring existence and uniqueness of an irreducible term reachable from t, called its *normal form* and written $t{\downarrow}$. Most of the time, we will also work under the assumptions that $\mathcal{R}$ is

- *subterm*, meaning that for all $\ell \to r \in \mathcal{R}$, r is either a strict subterm of ℓ or a *ground* term—that is a term without variables—in normal form;
- *destructor*, meaning that for all $\ell \to r \in \mathcal{R}$, ℓ is of the form $\mathsf{g}(u_1, \dots, u_n)$ where $\mathsf{g} \in \mathcal{F}_d$ and u_i is a constructor term—that is a term whose function symbols are all constructors.

Subterm convergent rewriting systems has been introduced in [3] and is classical in protocol analysis. It indeed includes a lot of standard primitives (see previous examples).

Sizes. The size of term t—its number of symbols—is written $|t|$. Some of our complexity results are also stated w.r.t. a succinct representation of terms as DAGs with maximal sharing—which may be exponentially more concise. If $st(t)$ is the set of subterms of t, $|t|_{\mathsf{dag}} = |st(t)|$ is thus the size of its DAG representation. This definition is easily lifted to sets and sequences of terms (with common sharing).

2.2. Processes

We model protocols as parallel processes that may exchange messages, modelled as terms. *Plain processes* are defined by the following grammar

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

where u, v are terms and $x \in \mathcal{X}$. Sending message v on channel u is performed by $\overline{u}\langle v\rangle$, and receiving on channel u—and binding the input to variable x—is performed by $u(x)$. The main difference with the calculus of [4] is the absence of replication. This restriction does *not* make protocol analysis trivially decidable. Indeed, although processes are finite, we study them in presence of an active, unbounded attacker. This will become clear in section 2.3 with the semantics of the calculus.

Example 3. Consider the protocol for private authentication [5], described informally using Alice-and-Bob notation:

$$\begin{aligned} X \to B &: \mathsf{aenc}(\langle N_X, \mathsf{pk}(sk_X)\rangle, \mathsf{pk}(sk_B)) \\ B \to X &: \mathsf{aenc}(\langle N_X, \langle N_B, \mathsf{pk}(sk_B)\rangle\rangle, \mathsf{pk}(sk_A)) \quad \text{if X=A} \\ &\quad\ \mathsf{aenc}(N_B, \mathsf{pk}(sk_B)) \quad \text{otherwise} \end{aligned}$$

B accepts authentication requests from A but not from other parties. However, the protocol should hide to any outsider that B is willing to engage with A—which explains the decoy message sent when B is contacted by a different party. The role of B can be specified in the applied pi calculus, writing $t = \mathsf{adec}(x, sk_B)$, $t_1 = \mathsf{proj}_1(t)$ and $t_2 = \mathsf{proj}_2(t)$:

$$\begin{aligned} B = \quad & c(x). \\ & \mathsf{if}\ t_2 = \mathsf{pk}(sk_A)\ \mathsf{then} \\ & \quad \overline{c}\langle \mathsf{aenc}(\langle t_1, \langle N_B, \mathsf{pk}(sk_B)\rangle\rangle, \mathsf{pk}(sk_A))\rangle \\ & \mathsf{else}\ \overline{c}\langle \mathsf{aenc}(N_B, \mathsf{pk}(sk_B))\rangle \end{aligned}$$

where $sk_A, sk_B, N_A, N_B \in \mathcal{N}_{\mathsf{prv}}$, $c \in \mathcal{N}_{\mathsf{pub}}$. *Anonymity* can be stated as equivalence of B and $B' = B\{sk_A \mapsto sk_{A'}\}$, assuming that the attacker has access to all public keys involved. Indeed, this means that an attacker sees no difference between B willing to engage with A or A'.

Inputs bind variables and define their scope: a process is said to be *closed* if it does not have free variables, i.e. all variables are bound. We require that all variables are bound at most once. A plain process P is *positive* when each of its conditionals is of the form $\mathsf{if}\ u = v\ \mathsf{then}\ Q\ \mathsf{else}\ 0$ (written $\mathsf{if}\ u = v\ \mathsf{then}\ Q$ for short). We denote by $|P|_{\mathsf{dag}}$ the size of the process—syntax tree plus the dag size of the terms.

2.3. Concrete semantics

Attacker knowledge. Semantics of processes defines their behaviours in presence of an active attacker, capable of

- *eavesdropping* messages, i.e., the attacker can learn outputs sent on public channels;
- performing *deductions*, i.e., computations on messages. For example, after observing an encryption $\mathsf{aenc}(m, \mathsf{pk}(k))$ and, later on, the decryption key k, he can deduce m;
- *controlling* public channels: the attacker can remove messages, as well as insert messages deducible from previous observations.

Attacker's observations are seen as sequences of terms that can be used by reference. For that, we refine variables as $\mathcal{X} = \mathcal{X}^1 \uplus \mathcal{AX}$, variables of $\mathcal{AX} = \{\mathsf{ax}_1, \mathsf{ax}_2, \ldots\}$ (*axioms*) acting as pointers. A term $\xi \in \mathcal{T}(\mathcal{F}, \mathcal{N}_{\mathsf{pub}} \cup \mathcal{AX})$ is called a *recipe*: typically, an attacker observing sequentially $\mathsf{aenc}(m, \mathsf{pk}(k))$ and k can use recipe $\xi = \mathsf{adec}(\mathsf{ax}_1, \mathsf{ax}_2)$ to construct m. Variables of $\mathcal{X}^1$ (*first-order variables*) stick to the initial role of variables, namely being used as binders for protocol inputs. A term $t \in \mathcal{T}(\mathcal{F}, \mathcal{N} \cup \mathcal{X}^1)$ is therefore called a *protocol term*.

All of this finally leads to the notion of *extended processes*, representing a set of plain processes executed in parallel, together with the knowledge aggregated by an attacker interacting with the protocol:

Definition 1. An extended process is a pair $A = (\mathcal{P}, \Phi)$ s.t.

- $\mathcal{P}$ is a multiset of closed plain processes (we use a double-bracket notation $\{\!\!\{\cdot\}\!\!\}$ for multisets).
- $\Phi = \{\mathsf{ax}_1 \mapsto u_1, \ldots, \mathsf{ax}_n \mapsto u_n\}$, called the *frame* and written $\Phi(A)$, is a substitution from axioms to ground constructor terms.

Φ embodies the attacker's knowledge: typically $\xi\Phi{\downarrow}$ is the result obtained from the recipe ξ w.r.t. the attacker's observations recorded in Φ.

Operational semantics. We operate under the assumption that protocols only exchange meaningful data. For example, the decryption of a term that is not a ciphertext is supposed to fail and shall not be input nor output. While this assumption is realistic for authenticated encryption, it may not hold for schemes with weaker security guarantees. We model this using the following notion of *message*:

Definition 2. A protocol term t is a *message* when its destructors operate smoothly, i.e. when $u{\downarrow}$ is a constructor term for all subterms u of t. We define the predicate $Msg(\cdot)$ such that $Msg(t)$ iff the term t is a message.

The semantics will ensure that the network only circulates messages. In practice, the semantics takes the form of a transition relation between extended processes labelled by so-called *actions*:

1. *input actions* $\xi(\zeta)$, where ξ and ζ are recipes, model an input from the attacker of a message (crafted from recipe ζ) on some channel (known through recipe ξ);
2. *output actions* $\overline{\xi}\langle \mathsf{ax}_n\rangle$, where ξ is a recipe, model an output on a channel known by the attacker (using recipe ξ), recorded into the frame (at pointer $\mathsf{ax}_n \in \mathcal{AX}$).

We call $\mathcal{A}$ the alphabet of actions, transitions $A \xrightarrow{a}_{\mathsf{c}} B$ being labelled by a the empty word ε or a letter of $\mathcal{A}$. More generally, given a word $w \in \mathcal{A}^*$, we write $A \overset{w}{\Rightarrow}_{\mathsf{c}} B$ when $A \xrightarrow{a_1}_{\mathsf{c}} \ldots \xrightarrow{a_n}_{\mathsf{c}} B$ with $w = a_1 \ldots a_n$. The transition relation is defined by the rules given in fig. 2.

Observe that the finite, pure π-calculus [39] is retrieved by imposing $\mathcal{F} = \emptyset$ and $\mathcal{R} = \emptyset$. This restriction makes the transition relation finitely branching up to renaming—which is *not* the case in general because the attacker can input infinitely-many terms to protocols using rule (IN).

Example 4. We introduce a toy example which will serve as a pedagogical running example. It was designed for its

$$(\mathcal{P} \cup \{\!\{0\}\!\}, \Phi) \xrightarrow{\varepsilon}_{\mathsf{c}} (\mathcal{P}, \Phi) \qquad \text{(NULL)}$$

$$(\mathcal{P} \cup \{\!\{\mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q\}\!\}, \Phi) \xrightarrow{\varepsilon}_{\mathsf{c}} (\mathcal{P} \cup \{\!\{P\}\!\}, \Phi) \quad \text{if } Msg(u),\ Msg(v) \text{ and } u{\downarrow} = v{\downarrow} \qquad \text{(THEN)}$$

$$(\mathcal{P} \cup \{\!\{\mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q\}\!\}, \Phi) \xrightarrow{\varepsilon}_{\mathsf{c}} (\mathcal{P} \cup \{\!\{Q\}\!\}, \Phi) \quad \text{if either } \neg Msg(u),\ \neg Msg(v) \text{ or } u{\downarrow} \neq v{\downarrow} \qquad \text{(ELSE)}$$

$$(\mathcal{P} \cup \{\!\{\overline{u}\langle t\rangle.P, v(x).Q\}\!\}, \Phi) \xrightarrow{\varepsilon}_{\mathsf{c}} (\mathcal{P} \cup \{\!\{P, Q\{x \mapsto t\}\}\!\}, \Phi) \quad \text{if } Msg(u),\ Msg(v),\ Msg(t) \text{ and } u{\downarrow} = v{\downarrow} \qquad \text{(COMM)}$$

$$(\mathcal{P} \cup \{\!\{P \mid Q\}\!\}, \Phi) \xrightarrow{\varepsilon}_{\mathsf{c}} (\mathcal{P} \cup \{\!\{P, Q\}\!\}, \Phi) \qquad \text{(PAR)}$$

$$(\mathcal{P} \cup \{\!\{u(x).P\}\!\}, \Phi) \xrightarrow{\xi(\zeta)}_{\mathsf{c}} (\mathcal{P} \cup \{\!\{P\{x \mapsto \zeta\Phi{\downarrow}\}\}\!\}, \Phi) \quad \text{if } \xi, \zeta \in \mathcal{T}(\mathcal{F}, \mathcal{N}_{\mathsf{pub}} \cup dom(\Phi)),\ Msg(u),\ Msg(\xi\Phi),\ Msg(\zeta\Phi) \text{ and } \xi\Phi{\downarrow} = u{\downarrow} \qquad \text{(IN)}$$

$$(\mathcal{P} \cup \{\!\{\overline{u}\langle t\rangle.P\}\!\}, \Phi) \xrightarrow{\overline{\xi}\langle \mathsf{ax}_n\rangle}_{\mathsf{c}} (\mathcal{P} \cup \{\!\{P\}\!\}, \Phi \cup \{\mathsf{ax}_n \mapsto t{\downarrow}\}) \quad \text{if } \xi \in \mathcal{T}(\mathcal{F}, \mathcal{N}_{\mathsf{pub}} \cup dom(\Phi)),\ Msg(u),\ Msg(\xi\Phi),\ Msg(t),\ \xi\Phi{\downarrow} = u{\downarrow}, \text{ and } n = |\Phi| + 1 \qquad \text{(OUT)}$$

Figure 2: Semantics of the calculus (w.r.t. an implicit rewrite system $\mathcal{R}$)

simplicity and capacity to illustrate the different notions defined in this paper. If $b \in \{0,1\} \subseteq \mathcal{N}_{\mathsf{pub}}$ and $c \in \mathcal{N}_{\mathsf{pub}}$:

$$P^b \triangleq c(x).\,\mathsf{if}\ \mathsf{proj}_2(x) = b\ \mathsf{then}\ \overline{c}\langle 0\rangle\ \mathsf{else}\ \overline{c}\langle \mathsf{proj}_2(x)\rangle$$
$$Q \triangleq c(x).\,\overline{c}\langle \mathsf{proj}_2(x)\rangle$$

Process Q forwards the second component of a term received through a public channel c. P^0 and P^1 have a similar behaviour as Q except that on input values $\langle t, 1\rangle$, P^1 outputs 0 rather than 1. We illustrate the semantics on P^1, e.g. by forwarding a hash $\mathsf{h}(n)$, $n \in \mathcal{N}_{\mathsf{pub}}$, sent by the attacker:

$$(\{\!\{P^1\}\!\}, \emptyset)$$
$$\xrightarrow{c(\langle 0, \mathsf{h}(n)\rangle)}_{\mathsf{c}} \left(\left\{\!\!\left\{ \begin{array}{l} \mathsf{if}\ \mathsf{proj}_2(\langle 0, \mathsf{h}(n)\rangle) = 1\ \mathsf{then}\ \overline{c}\langle 0\rangle \\ \mathsf{else}\ \overline{c}\langle \mathsf{proj}_2(\langle 0, \mathsf{h}(n)\rangle)\rangle \end{array} \right\}\!\!\right\}, \emptyset \right)$$
$$\xrightarrow{\overline{c}\langle \mathsf{ax}_1\rangle}_{\mathsf{c}} (\emptyset, \{\mathsf{ax}_1 \mapsto \mathsf{h}(n)\})$$

2.4. Equivalences

Process equivalences express indistinguishability between two situations and can be used to formalise many interesting security properties, as explained previously.

2.4.1. Static equivalence. Static equivalence expresses that the knowledge obtained in two different situations does not permit the attacker to distinguish them. This notion has been extensively studied in the literature (see e.g. [3]).

Definition 3. Frames Φ and Φ' are *statically equivalent*, written $\Phi \sim \Phi'$, when $dom(\Phi) = dom(\Phi')$ and for all ground recipes ξ, ζ such that $axioms(\xi, \zeta) \subseteq dom(\Phi)$:

1) $Msg(\xi\Phi)$ if and only if $Msg(\xi\Phi')$
2) if $Msg(\xi\Phi)$ and $Msg(\zeta\Phi)$, then $\xi\Phi{\downarrow}_{\mathcal{R}} = \zeta\Phi{\downarrow}_{\mathcal{R}}$ if and only if $\xi\Phi'{\downarrow}_{\mathcal{R}} = \zeta\Phi'{\downarrow}_{\mathcal{R}}$.

This definition is easily lifted to extended processes by writing $A \sim B$ instead of $\Phi(A) \sim \Phi(B)$.

Intuitively, this definition states that the observations recorded in Φ and Φ' cannot be distinguished by observing destructor failures or equality tests.

Example 5. The aim of encryption schemes is to make messages unintelligible to any agent not possessing the decryption key. This is modelled by static equivalence of

$$\Phi = \{\mathsf{ax}_1 \mapsto \mathsf{senc}(m, k)\} \quad \text{and} \quad \Phi' = \{\mathsf{ax}_1 \mapsto k'\}$$

where $m \in \mathcal{N}_{\mathsf{pub}}$ and $k, k' \in \mathcal{N}_{\mathsf{prv}}$. This means that the attacker cannot distinguish between an encrypted message (with unknown key) and an arbitrary, private nonce. Naturally this does not hold anymore once the decryption key is revealed, and indeed

$$\Phi_0 \cup \{\mathsf{ax}_2 \mapsto k\} \not\sim \Phi_1 \cup \{\mathsf{ax}_2 \mapsto k\}$$

(witnessed by recipes $\xi = \mathsf{sdec}(\mathsf{ax}_1, \mathsf{ax}_2)$ and $\zeta = m$).

2.4.2. Dynamic equivalences. Dynamic extensions of static equivalence consider distinguishability for an attacker interacting with protocols actively. Two classical equivalences are *trace equivalence* and *labelled bisimilarity*:

Definition 4. If A and B are extended processes, we write $A \sqsubseteq_t B$ when for all $A \overset{\mathsf{tr}}{\Rightarrow}_{\mathsf{c}} (P, \Phi)$, there exists (P', Φ') such that $B \overset{\mathsf{tr}}{\Rightarrow}_{\mathsf{c}} (P', \Phi')$ and $\Phi \sim \Phi'$. A and B are *trace equivalent*, denoted $A \approx_t B$, when $A \sqsubseteq_t B$ and $B \sqsubseteq_t A$.

Definition 5. Labelled bisimilarity $\approx_\ell$ is the largest symmetric relation containing $\sim$ such that $A \approx_\ell B$ and $A \xrightarrow{\alpha}_{\mathsf{c}} A'$ entails existence of B' such that $B \overset{\alpha}{\Rightarrow}_{\mathsf{c}} B'$ and $A' \approx_\ell B'$.

These two equivalences are well established as means to express security properties [6], [4].

Example 6. The statement that P^0 and Q have same behavior is expressed by $(P^0, \emptyset) \approx_\ell (Q, \emptyset)$. On the contrary $(P^1, \emptyset) \not\approx_t (Q, \emptyset)$ since the trace

$$(P^1, \emptyset) \xrightarrow{c(\langle 1,1\rangle).\,\overline{c}\langle \mathsf{ax}_1\rangle}_{\mathsf{c}} (0, \{\mathsf{ax}_1 \mapsto 0\})$$

cannot be matched in $(Q, \emptyset)$.

2.5. Decision problems for equivalences

We can now define the decision problems associated to these equivalences.

Definition 6 (parameterised equivalence problem)**.** We define the decision problem $\mathsf{Equiv}^{\approx}_{\mathcal{R},\mathcal{F}}$, with $\approx \in \{\sim, \approx_t, \approx_\ell\}$:

> CONSTANTS: A signature $\mathcal{F}$, a rewrite system $\mathcal{R}$
> INPUTS: two extended processes A, B
> QUESTION: $A \approx B$

In [3], $\mathsf{Equiv}^{\sim}_{\mathcal{R},\mathcal{F}}$ is proven undecidable in general, but PTIME if $\mathcal{R}$ is subterm convergent. However, it does not take the size of $|\mathcal{R}|$ into account: all procedures proposed so far [3], [27], [29] are actually exponential in $|\mathcal{R}|$ or $|\mathcal{F}|$. We argue that the size of $\mathcal{R}$ should be considered for complexity analyses as it can be specified by the user in many automated tools. We therefore focus on:

Definition 7 (general equivalence problem)**.** We define the decision problem $\mathsf{Equiv}^{\approx}_{\psi}$, with $\approx \in \{\sim, \approx_t, \approx_\ell\}$ and a predicate $\psi(\mathcal{F}, \mathcal{R}, A, B)$:

> INPUTS: a signature $\mathcal{F}$, a rewrite system $\mathcal{R}$, extended processes A, B such that $\psi(\mathcal{F}, \mathcal{R}, A, B)$ holds.
> QUESTION: $A \approx B$

The predicate ψ is necessary to avoid trivial undecidability and will be used for instance to restrict $\mathcal{R}$ to subterm convergent theories. For the sake of convenience, the predicate ψ will be specified in prose in theorem statements.

Remark 2.1. Input representation may influence complexity. We adress the strongest configurations: lower bounds in the tree representation of terms, upper bounds in DAG.

3. Complexity lower bounds

3.1. Static equivalence

We prove that, in our setting where the rewrite system is part of the input, static equivalence is coNP hard. By reduction from SAT, let $\varphi = \bigwedge_{i=1}^{p} C_i$ a boolean formula in CNF with n variables $x_1, \ldots, x_n$ and p clauses $C_1, \ldots, C_p$. Then we consider the constructors $\mathcal{F}_c = \{\mathsf{0},\ \mathsf{1},\ \mathsf{f}/2,\ \mathsf{g}/2\}$ and destructor $\mathcal{F}_d = \{\mathsf{eval}/n\}$ which are equipped with the rewrite system $\mathcal{R}$ defined by the following $p+1$ rules:

$$\mathsf{eval}(\mathsf{f}(x_1, y), \ldots, \mathsf{f}(x_n, y)) \to \mathsf{0}$$
$$\mathsf{eval}(\mathsf{g}(t_1^i, y), \ldots, \mathsf{g}(t_n^i, y)) \to \mathsf{0}$$

where $1 \leqslant i \leqslant p$ and

$$t_j^i = \begin{cases} x_j & \text{if } x_j \text{ does not appear in } C_i \\ \mathsf{0} & \text{if } x_j \text{ appears positively in } C_i \\ \mathsf{1} & \text{if } x_j \text{ appears negatively in } C_i \end{cases}$$

This assumes that no clause of φ contains both a litteral and its negation, but such clauses can be removed by a LOGSPACE preprocessing. Intuitively, if $t_1, \ldots, t_n \in \{\mathsf{0}, \mathsf{1}\}$, $\mathsf{eval}(\mathsf{g}(t_1, y), \ldots, \mathsf{g}(t_n, y))$ is a message and reduces to $\mathsf{0}$ iff the valuation $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ falsifies φ. Then it sufficies to observe that, for some $k \in \mathcal{N}_{\mathsf{prv}}$, $\{\mathsf{f}(\mathsf{0}, k),\ \mathsf{f}(\mathsf{1}, k)\} \sim \{\mathsf{g}(\mathsf{0}, k),\ \mathsf{g}(\mathsf{1}, k)\}$ iff φ is unsatisfiable.

Theorem 1. $\mathsf{Equiv}^{\sim}$ *is co*NP *hard for subterm destructor rewrite systems.*

3.2. Tools for reductions

We introduce tools for smoothing out lower bound proofs. They are presented as extensions of the syntax and semantics but can be encoded within the original calculus. These encodings, provided in the appendix, preserve positiveness of processes and are of polynomial size.

Non-deterministic choice. One first, classical feature is *non-deterministic choice* $P + Q$ defined by semantics

$$(\mathcal{P} \cup \{\!\{P + Q\}\!\}, \Phi) \xrightarrow{\varepsilon}_{\mathsf{c}} (\mathcal{P} \cup \{\!\{R\}\!\}, \Phi) \quad \text{if } R \in \{P, Q\}$$

Boolean guess. We assume that $\mathcal{F}_c$ contains two constant symbols $\mathsf{0}$ and $\mathsf{1}$ for booleans. One derivative of the non-deterministic choice is the boolean guess $\mathtt{Choose}(x).P$ defined by two rules, for $b \in \{\mathsf{0}, \mathsf{1}\}$:

$$\mathtt{Choose}(x).P \xrightarrow{\varepsilon}_{\mathsf{c}} P\{x \mapsto b\}$$

Boolean circuits. *Logical gates* are boolean functions with at most two inputs and two (identical) outputs. Typical examples are $\mathsf{0}$, $\mathsf{1}$, $\neg$, $\wedge$, $\vee$ or $=$. A *boolean circuit* is an acyclic graph of gates, each input (resp. output) of a gate being either isolated or connected to a unique output (resp. input) of another gate. Such a circuit Γ with m isolated inputs and n isolated outputs therefore models a function $\Gamma : \mathbb{B}^m \to \mathbb{B}^n$ (where $\mathbb{B} = \{\mathsf{0}, \mathsf{1}\}$). We integrate circuits into the calculus using syntax

$$x_1, \cdots, x_n \leftarrow \Gamma(b_1, \cdots, b_m).P$$

where $x_1, \ldots, x_n$ are variables and $b_1, \cdots, b_m$ terms, with

$$(\mathcal{P} \cup \{\!\{\vec{x} \leftarrow \Gamma(\vec{b}).P\}\!\}, \Phi) \xrightarrow{\varepsilon}_{\mathsf{c}} (\mathcal{P} \cup \{\!\{P\{\vec{x} \mapsto \Gamma(\vec{b}\downarrow)\}\}\!\}, \Phi) \quad \text{if } \mathit{Msg}(\vec{b}) \text{ and } \vec{b}\downarrow \subseteq \mathbb{B}$$

3.3. coNEXP hardness of dynamic equivalences

Let us consider a circuit $\Gamma : \{\mathsf{0}, \mathsf{1}\}^{m+2} \to \{\mathsf{0}, \mathsf{1}\}^{n+1}$. Using binary representation of integers, Γ can be interpreted as a function $[\![\Gamma]\!] : [\![0, 2^m - 1]\!] \times [\![1, 3]\!] \to \{0, 1\} \times [\![0, 2^n - 1]\!]$. This way, Γ encodes a CNF formula $[\![\Gamma]\!]_\varphi$ with 2^n variables $\vec{x} = x_0, \cdots, x_{2^n-1}$ and 2^m clauses:

$$[\![\Gamma]\!]_\varphi(\vec{x}) = \bigwedge_{i=0}^{2^m-1} \ell_i^1 \vee \ell_i^2 \vee \ell_i^3$$

$$\text{where } \begin{cases} \ell_i^j = x_k & \text{if } [\![\Gamma]\!](i, j) = (0, k) \\ \ell_i^j = \neg x_k & \text{if } [\![\Gamma]\!](i, j) = (1, k) \end{cases}$$

Rephrasing, $[\![\Gamma]\!](i, j)$ returns a sign bit and the j^{th} variable of the i^{th} clause of $[\![\Gamma]\!]_\varphi$.

Lemma 2 (SUCCINCT 3SAT [40])**.** *The following problem is* NEXP*-complete:*

> INPUT*: A circuit* Γ *with* $m+2$ *inputs and* $n+1$ *outputs.*
> QUESTION*: Is the* 3SAT*-formula* $[\![\Gamma]\!]_\varphi$ *satisfiable?*

Given an instance Γ of this problem, we design $\mathcal{F}$, $\mathcal{R}$ subterm destructor and A and B positive processes such that $A \not\approx_t B$ iff $A \not\approx_\ell B$ iff $[\![\Gamma]\!]_\varphi$ is satisfiable.

Term algebra. Terms are built over the following signature $\mathcal{F} = \mathcal{F}_c \uplus \mathcal{F}_d$ and rewrite system $\mathcal{R}$:

$$\begin{array}{rcl} \mathcal{F}_c & \triangleq & \{0\ ,1,\ \mathsf{Node}/2,\ \mathsf{h}/2,\ \mathsf{h}_\mathsf{N}/2,\ \mathsf{h}_\mathbb{B}/2\} \\ \mathcal{F}_d & \triangleq & \{\pi/2,\ \mathsf{Test}_\mathsf{N}/1,\ \mathsf{Test}_\mathsf{B}/1\} \end{array}$$

$$\begin{array}{cc} \pi(\mathsf{Node}(x,y),0) \to x & \pi(\mathsf{Node}(x,y),1) \to y \\ \mathsf{Test}_\mathsf{B}(\mathsf{h}_\mathbb{B}(0,z)) \to 1 & \mathsf{Test}_\mathsf{B}(\mathsf{h}_\mathbb{B}(1,z)) \to 1 \end{array}$$
$$\mathsf{Test}_\mathsf{N}(\mathsf{h}_\mathsf{N}(\mathsf{Node}(x,y),z)) \to 1$$

This models argument-testing mechanisms and a binary-tree datatype. We use syntactic sugar for recursive extractions: if $\ell \in \mathcal{T}(\mathcal{F}, \mathcal{N} \cup \mathcal{X}^1)^\star$ is a sequence of terms, the notation $t_{|\ell}$ is inductively defined by $t_{|\varepsilon} \triangleq t$ and $t_{|b\cdot\ell} \triangleq \pi(t,b)_{|\ell}$.

Reduction. We want to design processes A and B whose equivalence rephrases to $[\![\Gamma]\!]_\varphi$ being falsified by all valuations of its variables. We manage this using two processes:

1) $\mathsf{CheckTree}(x)$ checks whether x is a correct encoding of a valuation, that is, whether x is a complete binary tree of height n whose leaves are booleans;
2) $\mathsf{CheckSat}(x)$ checks whether the valuation encoded by x falsifies $[\![\Gamma]\!]_\varphi$.

A and B are defined in fig. 3 and their equivalence intuitively means: "*for all terms x, either x is not an encoding of a valuation or x falsifies a clause of $[\![\Gamma]\!]_\varphi$*". This is formalised by two lemmas, where $P_0 = \overline{c}\langle \mathsf{h}(0,s)\rangle.\overline{c}\langle \mathsf{h}(1,s)\rangle$:

Lemma 3. *Let x be a message which is not complete binary tree of height n with boolean leaves. Then there exists a reduction $\mathsf{CheckTree}(x) \overset{\varepsilon}{\Rightarrow}_\mathsf{c} C$ such that $C \approx_\ell (\{\!\{P_0\}\!\}, \emptyset)$.*

Lemma 4. *Let x be a complete binary tree of height n whose leaves are booleans, and $\mathtt{val}_x$ the valuation mapping the variable number $i = \sum_{k=1}^{m} p_k 2^{k-1}$ of $[\![\Gamma]\!]_\varphi$ to $x_{|p_1 \cdots p_m} \in \mathbb{B}$. If $\mathtt{val}_x$ does not satisfy $[\![\Gamma]\!]_\varphi$, there exists a reduction $\mathsf{CheckSat}(x) \overset{\varepsilon}{\Rightarrow}_\mathsf{c} C$ such that $C \approx_\ell (\{\!\{P_0\}\!\}, \emptyset)$.*

These two lemmas are the key ingredients needed to prove that $[\![\Gamma]\!]_\varphi$ is satisfiable iff $A \not\approx_t B$ iff $A \not\approx_\ell B$. Hence:

Theorem 5. $\mathsf{Equiv}^{\approx_t}$ *and* $\mathsf{Equiv}^{\approx_\ell}$ *are co*NEXP*-hard for subterm destructor rewrite systems and positive processes.*

4. A symbolic setting

The main difficulty to decidability is the attacker's ability to provide inputs with messages of its choice. One traditionally relies on symbolic techniques, only recording logical constraints characterising concrete actions. This is the main ingredient of our decision procedure (section 5).

4.1. Constraints as formulas

We formalise trace constraints by a first-order logic relying on two kinds of atomic formulas:

$$X \vdash^? u \ (\textit{deduction fact}) \quad \text{and} \quad u =^? v \ (\textit{equation})$$

where u, v are constructor protocol terms and $X \in \mathcal{X}^2$ is a new type of variable (*second-order variable*). They are used to model deductive capabilities: $X \vdash^? u$ intuitively means that the attacker is able to deduce u. Equations $u =^? v$ and their negations, *disequations* $u \neq^? v$, are typically imposed by conditionals (if $u = v$ then ... else ...). A first-order formula over such atoms is simply called a *formula*. We see sets of formulas as conjunctions of formulas and vice versa. Moreover, a substitution $\{x \mapsto t\}$ is also interpreted as $x =^? t$.

A formula is to be interpreted through the valuation of axioms, second-order variables and first-order variables. A *valuation* of a formula is therefore a triple (Φ, Σ, σ) with

$$\begin{array}{ll} \Phi : \mathcal{A}\mathcal{X} \to \mathcal{T}(\mathcal{F}, \mathcal{N}) & \text{(frame)} \\ \Sigma : \mathcal{X}^2 \to \mathcal{T}(\mathcal{F}, \mathcal{N}_\mathsf{pub} \cup \mathcal{A}\mathcal{X}) & \text{(second-order solution)} \\ \sigma : \mathcal{X}^1 \to \mathcal{T}(\mathcal{F}_c, \mathcal{N}) & \text{(first-order solution)} \end{array}$$

and the satisfiability relation $\models$ is defined as follows:

$$(\Phi, \Sigma, \sigma) \models (X \vdash^? u) \text{ iff } Msg(X\Sigma\Phi) \text{ and } X\Sigma\Phi{\downarrow} = u\sigma{\downarrow}$$
$$(\Phi, \Sigma, \sigma) \models (u =^? v) \text{ iff } u\sigma = v\sigma$$

Additionally, we use a form of bookkeeping to record which knowledge is available to the attacker when performing a deduction $X \vdash^? u$. For that we decompose $\mathcal{X}^2$ into

$$\mathcal{X}^2 = \biguplus_{n \in \mathbb{N}} \mathcal{X}^2_{:n} \qquad \mathcal{X}^2_n = \bigcup_{i=0}^{n} \mathcal{X}^2_{:i}$$

Variables $\mathcal{X}^2_n$ points to recipes that only use the first n protocol outputs. Formally speaking, this means that for all Σ we assume $img(\Sigma_{|\mathcal{X}^2_n}) \subseteq \mathcal{T}(\mathcal{F}, \mathcal{N}_\mathsf{pub} \cup \{\mathsf{ax}_i\}_{i=1}^n)$.

4.2. Semantics with symbolic inputs

In a nutshell, we abstract inputs by variables constrained by formulas. They are collected into *constraint systems*:

Definition 8. A constraint system is a tuple $\mathcal{C} = (\Phi, \mathsf{D}, \mathsf{E}^1)$:

- $\Phi = \Phi(\mathcal{C})$ is a frame of constructor protocol terms (not necessarily closed);
- $\mathsf{D} = \mathsf{D}(\mathcal{C})$ is a set of deduction facts $X \vdash^? t$ with $X \in \mathcal{X}^2_{|\Phi|}$ and t a constructor protocol term;
- $\mathsf{E}^1 = \mathsf{E}^1(\mathcal{C})$ is a set of formulas each of the form

$$u =^? v \qquad \text{or} \qquad \forall y_1 \ldots . \forall y_k . \bigvee_{j=1}^{p} u_j \neq^? v_j$$

where u_j, v_j, u, v are constructor protocol terms and $y_1, \ldots, y_k \in \mathcal{X}^1$. The restriction of E^1 to its equalities is written $\mathsf{E}^1|_=$.

We call $\mathcal{C}_\emptyset = (\emptyset, \emptyset, \emptyset)$ the empty constraint system.

Intuitively, Φ is a frame (with symbolic inputs), D collects the deductions an attacker has to perform, and E^1 gathers the (dis)equations of the trace. The form of disequations comes from the requirements that terms are messages in the concrete semantics. For example, consider

$$\text{if } \mathsf{proj}_1(x) = y \text{ then } P \text{ else } Q$$

The positive and the negative branches of the test will trigger

$$x =^? \langle y, x' \rangle \qquad \text{and} \qquad \forall x' .\ x \neq^? \langle y, x' \rangle$$

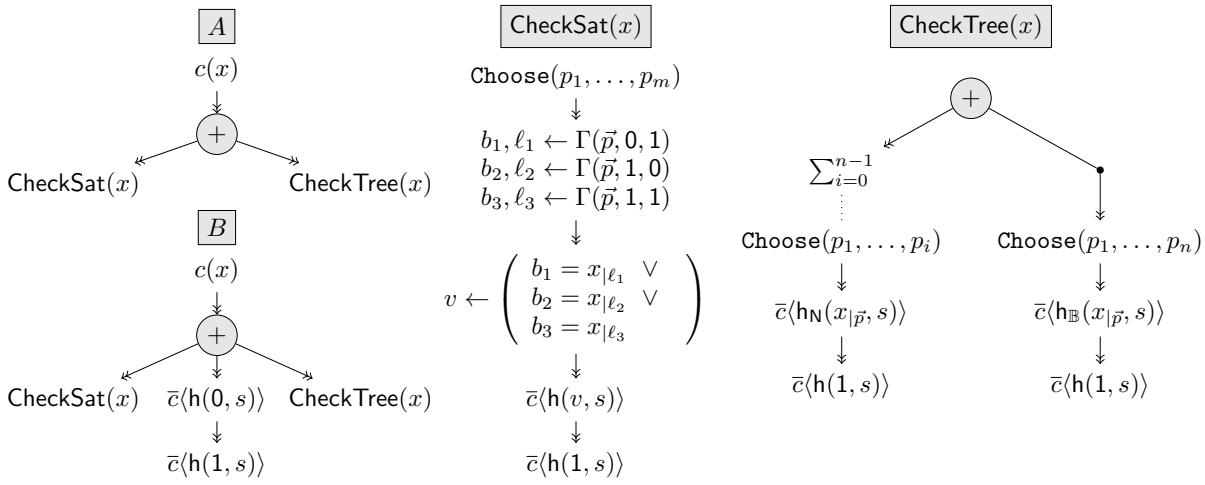


Figure 3: Definition of the extended processes A and B

Notice in particular that x' is implicitly quantified existentially in the first formula. As expected, such branching constraints are closely related to a notion of unification:

Definition 9. Let $T = \{s_i, t_i\}_{i\in I}$ be terms. A *unifier* of $E = \{s_i =^? t_i\}_{i\in I}$ modulo $\mathcal{R}$ is a substitution σ such that

1) $dom(\sigma) \subseteq vars(T)$;
2) for all $i \in I$, $s_i\sigma{\downarrow} = t_i\sigma{\downarrow}$, $Msg(s_i\sigma)$ and $Msg(t_i\sigma)$.

A set S of unifiers modulo $\mathcal{R}$ is said *complete* when, for all unifiers θ of E, there exist substitutions $\sigma \in S$ and τ such that $\theta|_{vars(T)}{\downarrow} = (\sigma\tau){\downarrow}$.

We denote by $mgu_{\mathcal{R}}(E)$ a complete set of unifiers of E, or simply $mgu_{\mathcal{R}}(s,t)$ for singletons $E = \{s =^? t\}$. They are known computable for (destructor) subterm convergent rewrite systems using narrowing. This notion is reminiscent to the classical, syntactic most general unifier $mgu(E)$ (known computable and unique up to variable renaming).

Example 7. Sticking to our running example, if $a \in \mathcal{N}$

$$mgu_{\mathcal{R}}(a, \mathsf{sdec}(\mathsf{proj}_1(x), y)) = \{\{x \mapsto \langle \mathsf{senc}(a, y), z\rangle\}\}$$

However, when unifying $t = \mathsf{proj}_1(x)$ with itself, as destructors shall not fail, we do not get the identity substitution but

$$mgu_{\mathcal{R}}(t, t) = \{\{x \mapsto \langle y, z\rangle\}\}$$

We now have all ingredients to specify the symbolic semantics. A *symbolic process* is a pair $(\mathcal{P}, \mathcal{C})$, $\mathcal{P}$ multiset of plain processes, $\mathcal{C}$ constraint system. Like concrete processes, their semantics is given by a transition relation $\xrightarrow{a}_{\mathsf{s}}$ where a is either ε or a *symbolic action* $a \in \mathcal{A}_{\mathsf{s}}$ with

$$\mathcal{A}_{\mathsf{s}} \triangleq \{Y(X), \overline{Y}\langle \mathsf{ax}_n\rangle \mid X, Y \in \mathcal{X}^2, n \in \mathbb{N}\}$$

Again $\xrightarrow{a}_{\mathsf{s}}$ is extended to a closure $\overset{w}{\Rightarrow}_{\mathsf{s}}$ where w is a word of symbolic actions. The most important rules defining $\xrightarrow{a}_{\mathsf{s}}$ are given in fig. 4 (remaining ones in the appendix). The execution tree of running example P^b is given in fig. 5.

4.3. Link with concrete semantics

Each symbolic trace abstracts several concrete traces. The possible concretisations of a symbolic trace are therefore the concrete traces compatible with the constraints of the symbolic process. We model this by the notion of *solution* of a constraint system:

Definition 10. We say that (Σ, σ) is a solution of a constraint system $\mathcal{C} = (\Phi, \mathsf{D}, \mathsf{E}^1)$ when $dom(\Sigma) = vars(\mathsf{D}) \cap \mathcal{X}^2$, $dom(\sigma) = vars(\mathsf{D}) \cap \mathcal{X}^1$ and

$$(\Phi\sigma, \Sigma, \sigma) \models \mathsf{D} \wedge \mathsf{E}^1$$

The substitution σ (resp. Σ) is called the first-order (resp. second-order) solution of $\mathcal{C}$ and the set of solutions of $\mathcal{C}$ is denoted $Sol(\mathcal{C})$. $\mathcal{C}$ is said satisfiable when $Sol(C) \neq \emptyset$.

In practice, σ is entirely determined by Σ. Indeed an invariant—called *origination property*—verified by $\mathcal{C}_\emptyset$ and preserved by symbolic transitions is that all variables $x \in \mathcal{X}^1$ of $\Phi(\mathcal{C})$ have been priorly determined by an input. Formally, for all $\mathsf{ax}_k \in dom(\Phi(\mathcal{C}))$ and $x \in vars(\mathsf{ax}_k\Phi(\mathcal{C}))$:

$$\exists X \in \mathcal{X}^2_{k-1},\ (X \vdash^? x) \in \mathsf{D}$$

Using the notion of solution, we can then express the link between the two semantics: all symbolic traces are concretisable (*soundness*) and all concrete traces are represented by a symbolic trace (*completeness*).

Lemma 6. *Let $(\mathcal{P}_{\mathsf{s}}, \mathcal{C})$ be a symbolic process. We have*

Soundness : *if* $(\mathcal{P}_{\mathsf{s}}, \mathcal{C}) \overset{\mathsf{tr}_{\mathsf{s}}}{\Rightarrow}_{\mathsf{s}} (\mathcal{P}'_{\mathsf{s}}, \mathcal{C}')$ *and* $(\Sigma', \sigma') \in Sol(\mathcal{C}')$, *then* $(\mathcal{P}_{\mathsf{s}}\sigma', \Phi(\mathcal{C})\sigma'{\downarrow}) \overset{\mathsf{tr}_{\mathsf{s}}\Sigma'}{\Longrightarrow}_{\mathsf{c}} (\mathcal{P}'_{\mathsf{s}}\sigma', \Phi(\mathcal{C}')\sigma'{\downarrow})$

Completeness : *if* $(\Sigma, \sigma) \in Sol(\mathcal{C})$ *and* $(\mathcal{P}_{\mathsf{s}}\sigma, \Phi(\mathcal{C})\sigma) \overset{\mathsf{tr}_{\mathsf{c}}}{\Rightarrow}_{\mathsf{c}} (\mathcal{P}'_{\mathsf{c}}, \Phi')$, *then there exist* $(\mathcal{P}_{\mathsf{s}}, \mathcal{C}) \overset{\mathsf{tr}_{\mathsf{s}}}{\Rightarrow}_{\mathsf{s}} (\mathcal{P}'_{\mathsf{s}}, \mathcal{C}')$ *and* $(\Sigma', \sigma') \in Sol(\mathcal{C}')$ *such that* $\Sigma \subseteq \Sigma'$, $\mathcal{P}'_{\mathsf{c}} = \mathcal{P}'_{\mathsf{s}}\sigma'$, $\mathsf{tr}_{\mathsf{c}} = \mathsf{tr}_{\mathsf{s}}\Sigma'$ *and* $\Phi' = \Phi(\mathcal{C}')\sigma'{\downarrow}$

$$(\mathcal{P} \cup \{\!\{\mathsf{if}\ u = v\ \mathsf{then}\ Q_1\ \mathsf{else}\ Q_2\}\!\}, (\Phi, \mathsf{D}, \mathsf{E}^1)) \xrightarrow{\varepsilon}_{\mathsf{s}} (\mathcal{P} \cup \{\!\{Q_1\}\!\}, (\Phi, \mathsf{D}, \mathsf{E}^1 \wedge \sigma)) \quad \text{(S-THEN)}$$
$$\text{if } \sigma \in mgu_{\mathcal{R}}(u\mu\downarrow =^? v\mu\downarrow)$$

$$(\mathcal{P} \cup \{\!\{u(x).Q\}\!\}, (\Phi, \mathsf{D}, \mathsf{E}^1)) \xrightarrow{Y(X)}_{\mathsf{s}} (\mathcal{P} \cup \{\!\{Q\}\!\}, (\Phi, \mathsf{D} \wedge X \vdash^? x \wedge Y \vdash^? y, \mathsf{E}^1 \wedge \sigma)) \quad \text{(S-IN)}$$
$$\text{if } \sigma \in mgu_{\mathcal{R}}(y =^? u\mu\downarrow) \text{ and } X, Y \in \mathcal{X}^2_{:|\Phi|}$$

$$(\mathcal{P} \cup \{\!\{\overline{u}\langle t\rangle.Q\}\!\}, (\Phi, \mathsf{D}, \mathsf{E}^1)) \xrightarrow{\overline{Y}\langle \mathsf{ax}_n\rangle}_{\mathsf{s}} (\mathcal{P} \cup \{\!\{Q\}\!\}, (\Phi \cup \{\mathsf{ax}_n \mapsto t\sigma\downarrow\}, \mathsf{D} \wedge Y \vdash^? y, \mathsf{E}^1 \wedge \sigma)) \quad \text{(S-OUT)}$$
$$\text{if } \sigma \in mgu_{\mathcal{R}}(y =^? u\mu\downarrow, t\mu\downarrow =^? t\mu\downarrow),\ y \text{ is fresh, } Y \in \mathcal{X}^2_{:n} \text{ and } n = |\Phi| + 1$$

Figure 4: Some rules of the symbolic semantics (where $\mu = mgu(\mathsf{E}^1|_=)$)

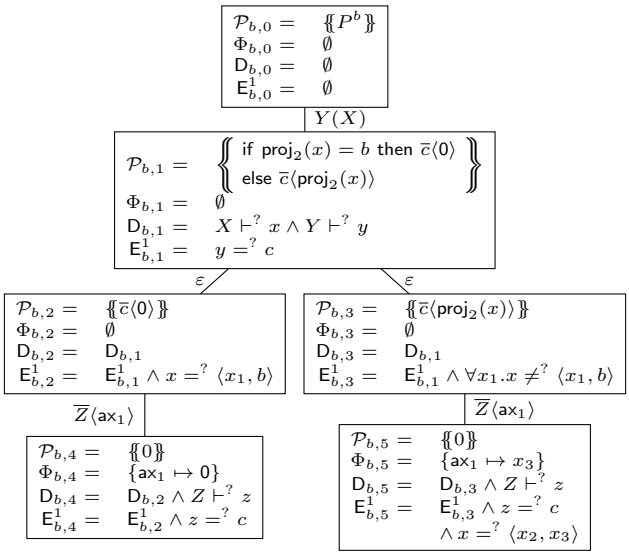


Figure 5: Symbolic execution tree of P^b, $b \in \{0, 1\}$

5. Complexity upper bounds

As non-equivalence is not a simple structural property, symbolic trees alone are not sufficient to decide dynamic equivalences. This is the motivation of our novel proof tool, *partition trees*. Our proof scenario is the following:

1) we show that equivalence of two processes rephrases to a simple condition on their partition tree T;
2) we construct T so that its nodes—which contain symbolic processes—have solutions of exponential size.

1 and 2 justify that, whenever two processes are not equivalent, there exists a concrete witness of non-equivalence of exponential size. This naturally leads to a coNEXP procedure for trace equivalence and labelled bisimilarity.

5.1. Most general solutions

Similarly to *mgu*'s, we define *mgs*'—most general solutions—of a constraint system, acting as solutions of minimal size. For that, if π is a predicate on second-order substitutions stable under domain extension—meaning that $\pi(\Sigma)$ and $\Sigma \subseteq \Sigma'$ entails $\pi(\Sigma')$—we consider the set of solutions of a constraint system $\mathcal{C}$ satisfying π:

$$Sol^{\pi}(\mathcal{C}) = \{(\Sigma, \sigma) \in Sol(\mathcal{C}) \mid \pi(\Sigma)\}$$

Definition 11. The *most general solutions of $\mathcal{C}$ satisfying* π, denoted $mgs^{\pi}(\mathcal{C})$, is the set of substitutions Σ such that:

1) $img(\Sigma) \subseteq \mathcal{T}(\mathcal{F}, \mathcal{N}_{\mathsf{pub}} \cup \mathcal{AX} \cup \mathcal{X}^2)$ and $dom(\Sigma) \subseteq vars(\mathcal{C}) \cap \mathcal{X}^2$;
2) *instantiating by fresh public names yields a solution*:
   for all bijective mapping Σ' from the second-order variables of $vars(\mathcal{C}, \Sigma) \setminus dom(\Sigma)$ to fresh public names, there exists σ such that $(\Sigma\Sigma'|_{vars(\mathcal{C})}, \sigma) \in Sol^{\pi}(\mathcal{C})$
3) *all solutions are instances of a mgs:*
   for all $(\Sigma, \sigma) \in Sol^{\pi}(\mathcal{C})$, there exist $\Sigma' \in mgs^{\pi}(\mathcal{C})$ and Σ'' such that $\Sigma = \Sigma'\Sigma''|_{vars(\mathcal{C})}$.

We suppose that all substitutions in $mgs^{\pi}(\mathcal{C})$ are distinct modulo renaming of variables.

This notion is very natural in that it follows in the steps of the definition of *mgu*'s. The computation of *mgs'* is however more involved—it is in the same vein, though more general than other work on constraint solving for security protocols [14], [22]. The approach is twofold:

1) constraint systems $\mathcal{C}$ are *extended*, in particular with a set of second-order constraints $\mathsf{E}^2(\mathcal{C})$. This set is completely analogous to E^1 but gathers (dis)equalities on second-order terms;
2) we use a (terminating) constraint-solving relation $\rightarrow$ over extended constraint systems.

A dedicated notation $\bot$ is used to denote a constraint system without solution. $mgs(\mathcal{C})$ is then obtained by taking the second-order *mgu's* of solved systems reachable from $\mathcal{C}$:

$$mgs(\mathcal{C}) = \left\{ mgu(\mathsf{E}^2(\mathcal{C}')|_=) \;\middle|\; \begin{array}{l} \mathcal{C} \rightarrow^{\star} \mathcal{C}',\ \mathcal{C}' \neq \bot, \\ \mathcal{C}' \text{ irreducible} \end{array} \right\}$$

Because of lack of space we omit the description of this constraint solving procedure. The details can be found in the technical report [2] but are not necessary to understand the overall decision procedure.

5.2. Partition tree

Partition trees are key to our decision procedure. Structurally speaking, a partition tree is similar to a symbolic execution tree where children of each node are refined to guide the decision of dynamic equivalences. Typically its nodes—so-called *configurations*—contain symbolic processes sharing statically-equivalent solutions:

Definition 12. A configuration is a triple (Γ, π, ℓ) where Γ is a set of symbolic processes, $\ell \in \mathcal{A}_s \cup \{\varepsilon\}$ and π is a predicate on second-order solutions. We also assume that, for all $(\mathcal{P}, \mathcal{C}) \in \Gamma$:

1) $Sol^{\pi}(\mathcal{C}) \neq \emptyset$ and $|mgs^{\pi}(\mathcal{C})| = 1$;
2) if $(\Sigma, \sigma) \in Sol^{\pi}(\mathcal{C})$ and $(\mathcal{P}', \mathcal{C}') \in \Gamma$, then for some σ', $(\Sigma, \sigma') \in Sol^{\pi}(\mathcal{C}')$ and $\Phi(\mathcal{C})\sigma \sim \Phi(\mathcal{C}')\sigma'$.

When a node n of a partition tree is labelled by (Γ, π, ℓ) we denote by $\Gamma(n)$, $\pi(n)$, and $\ell(n)$ the corresponding elements of the configuration. Note an important consequence of the definition: all constraint systems in $\Gamma(n)$ have a unique *mgs* (by point 1) and this *mgs* is common to all of them (by point 2). We denote it $mgs(n)$. The definition of partition trees is then the following:

Definition 13. Let P_1 and P_2 be two closed plain processes. A *partition tree T of P_1 and P_2* is a finite tree whose nodes are labelled by configurations. It also verifies, for all nodes n and $(\mathcal{P}, \mathcal{C}) \in \Gamma(n)$:

1) *the initial processes are in the root:*

   if n is the root of the tree then $\ell(n) = \varepsilon$, $\pi(n) = \top$ and $\Gamma(n)$ contains $(\{\!\{P_1\}\!\}, \mathcal{C}_\emptyset)$ and $(\{\!\{P_2\}\!\}, \mathcal{C}_\emptyset)$;

2) *nodes are closed under ε-transition*:

   if $(\mathcal{P}, \mathcal{C}) \overset{\varepsilon}{\Rightarrow}_{\mathsf{s}} (\mathcal{P}', \mathcal{C}')$ and $Sol^{\pi(n)}(\mathcal{C}') \neq \emptyset$ then $(\mathcal{P}', \mathcal{C}') \in \Gamma$;

3) *completeness of the partition tree:*

   if $(\mathcal{P}, \mathcal{C}) \overset{\ell}{\Rightarrow}_{\mathsf{s}} (\mathcal{P}', \mathcal{C}')$ and $(\Sigma, \sigma) \in Sol^{\pi(n)}(\mathcal{C}')$ then there exists n' child of n s.t. $(\mathcal{P}', \mathcal{C}') \in \Gamma(n')$, $\ell(n') = \ell$ and $(\Sigma', \sigma) \in Sol^{\pi(n')}(\mathcal{C}')$ for some Σ';

Besides, if n_c is a child node of n and $(\mathcal{P}_c, \mathcal{C}_c) \in \Gamma(n_c)$:

4) *predicates are refined along branches:* $\pi(n_c) \subseteq \pi(n)$;
5) *soundness of the partition tree:*

   if $(\Sigma, \sigma) \in Sol^{\pi(n)}(\mathcal{C})$, $(\Sigma_c, \sigma_c) \in Sol^{\pi(n_c)}(\mathcal{C}_c)$ and $\Sigma \subseteq \Sigma_c$, then $\Gamma(n_c)$ contains all $(\mathcal{P}', \mathcal{C}')$ such that $(\mathcal{P}, \mathcal{C}) \overset{\ell(n_c)}{\Longrightarrow}_{\mathsf{s}} (\mathcal{P}', \mathcal{C}')$ and $\Phi(\mathcal{C}_c)\sigma_c \sim \Phi(\mathcal{C}')\sigma'$ for some σ' such that $(\Sigma_c, \sigma') \in Sol(\mathcal{C}')$.

We denote by $\mathsf{PTree}(P_1, P_2)$ the (infinite) set of all partition trees of P_1 and P_2.

The nodes of $T \in \mathsf{PTree}(P_1, P_2)$ gather statically-equivalent processes, while second-order predicates π describe the recipes the attacker has to use to reach a given node. Symbolic traces are then embedded into the tree as edges (item 3) and the edge relation encompasses all symbolic traces leading to statically-equivalent processes (item 5). Typically, trace equivalence of P_1 and P_2 will be stated as a condition on the branches of T.

Example 8. Two partition trees are presented in fig. 6. They use notations of fig. 5, $\mathcal{E}_i^b = (\{\!\{\mathcal{P}_{b,i}\}\!\}, (\Phi_{b,i}, \mathsf{D}_{b,i}, \mathsf{E}^1_{b,i}))$, $\mathcal{E}_Q^b = (\overline{c}\langle \mathsf{proj}_2(x)\rangle, \mathcal{C}_{b,1})$ and $\mathcal{E}_Q^f = (\{\!\{0\}\!\}, \mathcal{C}_f)$ with

$$\mathcal{C}^f = (\Phi_f, \mathsf{D}_f, \mathsf{E}^1_f) \quad \mathsf{D}_f = \{X \vdash^? x, Y \vdash^? y, Z \vdash^? z\}$$
$$\Phi_f = \{\mathsf{ax}_1 \mapsto x_2\} \quad \mathsf{E}^1_f = \{y =^? c, z =^? c, x = \langle x_1, x_2\rangle\}$$

In this example, second-order predicates π are described by second-order formulas φ^2 (casting the satisfiability relation of our first-order logic in the natural way).

A criterion for equivalence. Each branch a partition tree T intuitively encompasses equivalent trace scenarios. Typically, trace equivalence will be interpreted as a simple condition on these branches. Assuming $(\mathcal{P}, \mathcal{C}) \overset{a}{\Rightarrow}_{\mathsf{s}} (\mathcal{P}_c, \mathcal{C}_c)$, $a \in \mathcal{A}_s \cup \{\varepsilon\}$, we write $(\mathcal{P}, \mathcal{C}), n \overset{a}{\rightarrow}_T (\mathcal{P}_c, \mathcal{C}_c), n_c$ when n, n_c are nodes of T with n_c is a child of n (if $a \in \mathcal{A}_s$) or $n = n_c$ (if $a = \varepsilon$), $(\mathcal{P}, \mathcal{C}) \in \Gamma(n)$ and $(\mathcal{P}_c, \mathcal{C}_c) \in \Gamma(n_c)$. It is extended into the closure $\overset{\mathsf{tr}}{\Rightarrow}_T$ as usual. The core of our procedure for trace equivalence is then stated by:

Lemma 7. *Let P_1, P_2 be two ground plain processes, a partition tree $T \in \mathsf{PTree}(P_1, P_2)$ and n_0 the root of T.*

$$(\{\!\{P_1\}\!\}, \emptyset) \sqsubseteq_t (\{\!\{P_2\}\!\}, \emptyset)$$
iff
for all reductions $(\{\!\{P_1\}\!\}, \mathcal{C}_\emptyset), n_0 \overset{\mathsf{tr}}{\Rightarrow}_T (\mathcal{P}, \mathcal{C}), n$
there exists a reduction $(\{\!\{P_2\}\!\}, \mathcal{C}_\emptyset), n_0 \overset{\mathsf{tr}}{\Rightarrow}_T (\mathcal{P}', \mathcal{C}'), n$

Proof (sketch). We prove both directions separately.
($\Rightarrow$) We proceed by contraposition. The key argument is the following claim ($\star$), provable by induction on $|\mathsf{tr}|$:

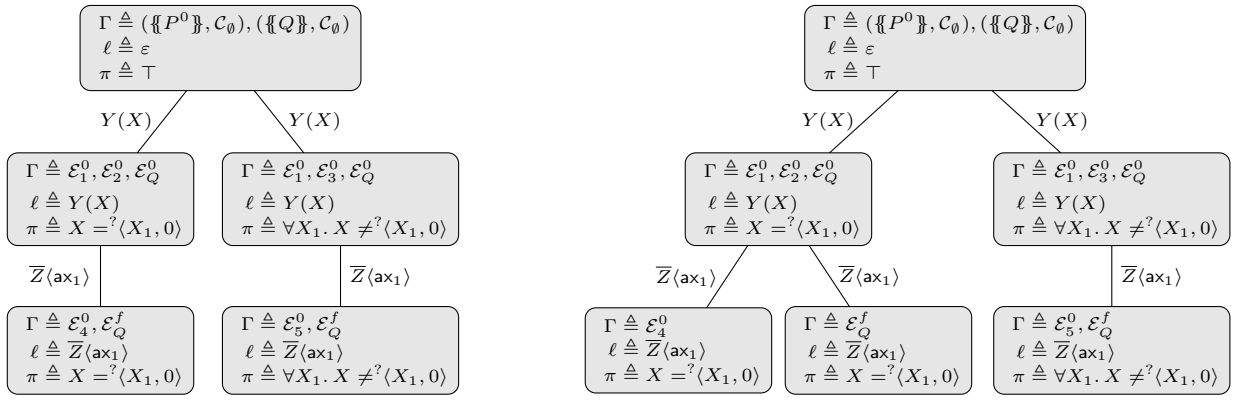
> Let n node of T and $(\mathcal{P}_1, \mathcal{C}_1), (\mathcal{P}_2, \mathcal{C}_2) \in \Gamma(n)$. If
> - $(\mathcal{P}_1, \mathcal{C}_1), n \overset{\mathsf{tr}}{\Rightarrow}_T (\mathcal{P}'_1, \mathcal{C}'_1), n'$
> - for all $(\mathcal{P}'_2, \mathcal{C}'_2)$, $(\mathcal{P}_2, \mathcal{C}_2), n \overset{\mathsf{tr}}{\not\Rightarrow}_T (\mathcal{P}'_2, \mathcal{C}'_2), n'$
> - $(\Sigma, \sigma_1) \in Sol^{\pi(n')}(\mathcal{C}'_1)$
> - $(\Sigma, \sigma_2) \in Sol(\mathcal{C}_2)$
>
> then we have $\Phi(\mathcal{C}'_1)\sigma_1 \not\sim \Phi$ for all concrete reductions $(\mathcal{P}_2\sigma_2, \Phi(\mathcal{C}_2)\sigma_2\downarrow) \overset{\mathsf{tr}\Sigma'}{\Longrightarrow}_{\mathsf{c}} (P, \Phi)$ s.t. $\Sigma \subseteq \Sigma'$.

Then, assume $(\{\!\{P_1\}\!\}, \mathcal{C}_\emptyset), n_0 \overset{\mathsf{tr}}{\Rightarrow}_T (\mathcal{P}, \mathcal{C}), n$ and, for all $(\mathcal{P}', \mathcal{C}')$, $(\{\!\{P_2\}\!\}, \mathcal{C}_\emptyset), n_0 \overset{\mathsf{tr}}{\not\Rightarrow}_T (\mathcal{P}', \mathcal{C}'), n$. We know by soundness of the symbolic semantics (theorem 6) that $(\{\!\{P_1\}\!\}, \emptyset) \overset{\mathsf{tr}\Sigma}{\Longrightarrow}_{\mathsf{c}} (\mathcal{P}\sigma, \Phi(\mathcal{C})\sigma\downarrow)$ for an arbitrary $(\Sigma, \sigma) \in Sol^{\pi(n)}(\mathcal{C})$. We thus obtain $(\{\!\{P_1\}\!\}, \emptyset) \not\sqsubseteq_t (\{\!\{P_2\}\!\}, \emptyset)$ by ($\star$) with $\mathcal{P}_1 = \{\!\{P_1\}\!\}$, $\mathcal{P}_2 = \{\!\{P_2\}\!\}$ and $\mathcal{C}_1 = \mathcal{C}_2 = \mathcal{C}_\emptyset$.

($\Leftarrow$) Suppose $(\{\!\{P_1\}\!\}, \emptyset) \overset{\mathsf{tr}_c}{\Longrightarrow}_{\mathsf{c}} (\mathcal{P}_1, \Phi_1)$. We have to show that there exists $(\mathcal{P}_2, \Phi_2)$ s.t. $(\{\!\{P_2\}\!\}, \emptyset) \overset{\mathsf{tr}_c}{\Longrightarrow}_{\mathsf{c}} (\mathcal{P}_2, \Phi_2)$ and $\Phi_1 \sim \Phi_2$. By completeness of the symbolic semantics (theorem 6), we have tr, n, $(\mathcal{P}, \mathcal{C}) \in \Gamma(n)$ and $(\Sigma, \sigma) \in Sol(\mathcal{C}_A)$ s.t. $\Phi = \Phi(\mathcal{C})\sigma\downarrow$, $\mathsf{tr}_c = \mathsf{tr}\Sigma$ and

$$(\{\!\{P_1\}\!\}, \mathcal{C}_\emptyset), n_0 \overset{\mathsf{tr}}{\Rightarrow}_T (\mathcal{P}, \mathcal{C}), n$$

By hypothesis, there exists $(\mathcal{P}', \mathcal{C}') \in \Gamma(n)$ s.t. $(\{\!\{P_2\}\!\}, \mathcal{C}_\emptyset), n_0 \overset{\mathsf{tr}}{\Rightarrow}_T (\mathcal{P}', \mathcal{C}'), n$.By definition 12, there is σ' s.t. $(\Sigma, \sigma') \in Sol(\mathcal{C}')$ and $\Phi = \Phi(\mathcal{C})\sigma\downarrow \sim \Phi(\mathcal{C}')\sigma'\downarrow$. Therefore, we obtain by theorem 6, $(\{\!\{P_2\}\!\}, \emptyset) \overset{\mathsf{tr}\Sigma}{\Longrightarrow}_{\mathsf{c}} (\mathcal{P}'\sigma', \Phi(\mathcal{C}')\sigma'\downarrow)$. As $\mathsf{tr}_c\Phi\downarrow = \mathsf{tr}\Sigma\Phi\downarrow$ and $\Phi \sim \Phi(\mathcal{C}')\sigma'\downarrow$ we have that $\mathsf{tr}_c\Phi(\mathcal{C}')\sigma'\downarrow = \mathsf{tr}\Sigma\Phi(\mathcal{C}')\sigma'\downarrow$. Hence, we conclude that $(\{\!\{P_2\}\!\}, \emptyset) \overset{\mathsf{tr}_c}{\Longrightarrow}_{\mathsf{c}} (\mathcal{P}'\sigma', \Phi(\mathcal{C}')\sigma'\downarrow)$. □



(a) Partition tree $T_0 \in \mathsf{PTree}(P^0, Q)$

(b) Partition tree $T_1 \in \mathsf{PTree}(P^1, Q)$

Figure 6: Example of partition trees

This lemma justifies decidability of trace equivalence given a partition tree—whose construction is outlined in the next paragraph. We investigate complexity afterwards.

Remark 5.1. A similar result can be stated for labelled bisimilarity, omitted here due to lack of space. In spirit, we cope with the more complex notion of equivalence by considering subtrees of T rather than simple nodes. The procedure is outlined in the appendix and described extensively in the technical report [2].

Generation of a partition tree. Let us describe, in broad lines, the ingredients to compute $T \in \mathsf{PTree}(P_1, P_2)$.

1) First, *constraint systems are extended* with additional components, e.g. logical constraints on second-order terms or deduction facts modelling the attacker knowledge, and symbolic semantics are lifted to extended constraint systems. The goal is to carry additional information through the procedure to help with the generation of the predicates π. Typically, initial processes $\mathcal{P}_i = \{\!\{P_i\}\!\}$, $i \in \{1, 2\}$, are put under the form $(\mathcal{P}_i, \mathcal{C}_\emptyset, \mathcal{C}^e_\emptyset)$ with $\mathcal{C}^e_\emptyset$ the empty extended constraint system.
2) We define *simplification rules* putting extended constraint systems in a form where their satisfiability is trivial to decide. Typically, the set $\mathcal{S}_{\mathsf{root}}$ of symbolic processes at the root of the partition tree is obtained by saturating $\{(\mathcal{P}_1, \mathcal{C}_\emptyset, \mathcal{C}^e_\emptyset), (\mathcal{P}_2, \mathcal{C}_\emptyset, \mathcal{C}^e_\emptyset)\}$ under ε-transitions and using simplification rules to get rid of unsatisfiable constraint systems.
3) From the processes of the root, two sets of children processes are then defined: $\mathcal{S}_{\mathsf{in}}$ (processes ε-reachable after an input transition) and $\mathcal{S}_{\mathsf{out}}$ (processes ε-reachable after an output transition). $\mathcal{S}_{\mathsf{in}}$ and $\mathcal{S}_{\mathsf{out}}$ may need to be partitioned to satisfy the requirement that processes of a same configuration are statically equivalent. This is achieved by *case-distinction rules*, resulting into
$$\mathcal{S}_{\mathsf{in}} = \mathcal{S}^1_{\mathsf{in}} \uplus \ldots \uplus \mathcal{S}^p_{\mathsf{in}} \qquad \mathcal{S}_{\mathsf{out}} = \mathcal{S}^1_{\mathsf{out}} \uplus \ldots \uplus \mathcal{S}^q_{\mathsf{out}}$$
This gives $p + q$ children in the partition tree. For example, the first child node is defined by
   - $\Gamma = \{(\mathcal{P}, \mathcal{C}) \mid (\mathcal{P}, \mathcal{C}, \mathcal{C}^e) \in \mathcal{S}^1_{\mathsf{in}}\}$
   - $\ell = Y(X)$
   - $\pi(\Sigma)$ iff $\exists \sigma.\, (\Sigma, \sigma) \in Sol(\mathcal{C}^e)$ (where $\mathcal{C}^e$ is an arbitrary extended constraint system appearing in $\mathcal{S}^1_{\mathsf{in}}$)
4) The construction is then kept up *top-down*: for all current leaves of the partition tree for which visible actions are still reachable, we generate the sets $\mathcal{S}_{\mathsf{in}}$ and $\mathcal{S}_{\mathsf{out}}$ and partition them using case-distinction rules, giving birth to new leaves. This process naturally terminates as the transition relation $\xrightarrow{a}_{\mathsf{s}}$ is strongly terminating.

We prove in the technical report [2] that this procedure indeed generates a partition tree $T \in \mathsf{PTree}(P_1, P_2)$.

Final result. Using the procedure above to generate a partition tree and lemma 7, we obtain a decision procedure for trace equivalence. This procedure is actually the one implemented in our automated tool, DEEPSEC. As mentionned earlier, we also have a decision procedure for labelled bisimilarity based on partition trees, outlined in the appendix.

However, in the end, we use a different argument to obtain the theoretical complexity upper bound expected (coNEXP). By careful analysis of the sizes of the extended constraint systems carried out during the procedure and by bounding the number of applications of case-distinction rules, we can prove that the partition tree we generate has solutions of exponential size. Hence:

Theorem 8. *There exists $p \in \mathbb{N}$ such that for all convergent subterm destructor rewriting system $\mathcal{R}$, closed plain processes P_1 and P_2 there exists a partition tree $T \in \mathsf{PTree}(P_1, P_2)$ such that for all nodes n in T, for all $\Sigma \in mgs(n)$, $|\Sigma|_{\mathsf{dag}} < 2^{(|P_1, P_2, \mathcal{R}|_{\mathsf{dag}} + |\mathcal{F}|)^p}$.*

The combination of this theorem and lemma 7 straightforwardly justifies the existence, whenever $P_1 \not\approx_t P_2$, of a concrete trace of exponential size in P_1 or P_2 which is not matched in the other process. This easily leads to a coNEXP decision procedure for trace equivalence, using the fact that static equivalence is NP—this result is easily obtained from existing procedures, e.g. [3], [27], [29]. As we explained, the procedure can be adapted for labelled bisimilarity. We refer to appendix for an outline or the technical report [2]

for details. Hence:

Theorem 9. $\mathsf{Equiv}^{\approx_t}$ *and* $\mathsf{Equiv}^{\approx_\ell}$ *are co*NEXP *for subterm convergent destructor rewriting systems.*

6. Implementation

Building on the previous section we have implemented a prototype for verifying trace equivalence in OCaml, called DEEPSEC (DEciding Equivalence Properties in SECurity protocols), publicly available at [1]. The tool's specification language extends the grammar presented in section 2.2: in particular, we define a non-deterministic choice operator $P + Q$, a let operator for variable assignment $\mathsf{let}\ x = u\ \mathsf{in}\ P\ \mathsf{else}\ Q$, as well as bounded replication $!^n P$ defining n copies of P in parallel. These additional primitives are only here for modelling convinience—and the native integration allowed specific optimisations compared to encoding within the initial calculus. The syntax and structure of DEEPSEC's input files are similar to the ones of the widely used ProVerif [17] tool. We hope this will make it easier for new users to discover and handle our tool.

Partial order reductions. The tool also implements *partial order reductions* (POR), an optimisation technique for protocol analysis developed by Baelde et al. [12]. The basic idea is to discard part of the state space that is redundant. This optimisation is sound when processes are *action determinate*, as defined in [12]. Assigning a different channel name to each parallel process is a simple, syntactic way to ensure this property although this is not always possible—typically when looking at anonymity or unlinkability properties. In practice, DEEPSEC automatically detects action-determinate processes and activates the POR, which drastically reduces the number of symbolic executions that need be considered.

Distributing the computation. Following the outline of section 5, the main task of DEEPSEC is to generate a partition tree. This task can be distributed: computing a given node of the tree can be done independently of its sibling nodes. However, some engineering is needed to avoid heavy communication overhead due to task scheduling. Indeed, the partition tree is indeed not a balanced tree and it is impossible to know which branches will be larger than others. Hence, in practice we do not directly compute and return the children of each node in the most straightforward manner, but proceed in two steps:

1) We start with a breadth-first generation of the partition tree. The number of pending nodes will gradually grow until, potentially, exceeding a threshold parameter n.
2) Each available core focuses on one of these nodes, computes the whole subtree rooted by this node (depth-first manner), and is then assigned a new node. If at some point cores become idle—because all nodes generated at step 1 are either completed or currently assigned to an active core—we restart this two-step procedure on incomplete nodes.

While parallelisation is also supported by the AKISS tool, DEEPSEC goes one step further as it is also able to distribute the computation through clusters of computers.

Benchmarks. We performed extensive benchmarks to compare our tool against other tools that verify equivalence properties for a bounded number of sessions: AKISS [19], APTE [20], SAT-EQUIV [30] and SPEC [47]. Experiments are carried out on Intel Xeon 3.10GHz cores, with 50Go of memory. AKISS and DEEPSEC use 35 cores as they support parallelisation—unlike the others which therefore use a single core. The results are summarised in fig. 7.

We analysed strong secrecy, an equivalence based version of secrecy, for several classical authentication protocols. These benchmarks are mainly used for measuring scalability when increasing the number of sessions (fig. 7 indicates the number of roles in parallel, as depending on the exact scenario a session may require more or less roles). The DEEPSEC tool clearly outperforms AKISS, APTE, and SPEC. The SAT-EQUIV tool becomes more efficient, when the number of sessions significantly increases. However, the Otway-Rees protocol cannot be analysed by SAT-EQUIV as it does not satisfy their type compliance condition and the Needham-Schroeder-Lowe protocol is out of its scope, as SAT-EQUIV does not support asymmetric encryption. DEEPSEC is even able to verify a higher number of roles than those reported in fig. 7. Setting a timout of 12 hours, e.g., the Denning-Sacco and Yahalom-Lowe protocols can be verified for 52 roles in 11h09m, resp. 25 roles in 10h04m.

To illustrate the broad scope of the tool we analyse unlinkability and anonymity properties for a number of other protocols: Abadi and Fournet's anonymous authentication protocol [5], the AKA protocol deployed in 3G telephony networks [10], the Passive Authentication and Basic Access Control (BAC) protocols implemented in the European passport [37], as well as the Prêt-à-Voter (PaV) [43] and several variants of the mixnet based Helios [7] voting protocols. We comment a bit more on the voting protocol examples. Relying on the reduction result of Arapinis et al. [9], we know that it is sufficient to consider three voters, two honest and one dishonest one, to conclude vote privacy for an arbitrary number of voters. Moreover, when revoting is allowed, which is the case for Helios, but not for PaV, we only need to consider a server which accepts seven ballots that may come from any of the three voters. For the Helios protocol we consider several versions. The *vanilla* Helios version, which does not allow revoting, is known to be vulnerable to a ballot-copy attack [32]—the attacker simply copies the ballot of a honest voter in order to bias the outcome. Two countermeasures have been proposed to thwart this attack: one applies a ballot weeding procedure (W), while the other is based on a zero-knowledge proof (ZKP) that links the identity of the voter to the ballot. When no revote (NR) is allowed these two versions are indeed shown to be secure. When allowing revoting we consider the case where seven ballots can be accepted [9], under two different scenarios. When only the dishonest voter revotes (dR) we can show the security of the weeding mechanism. When however one honest voter re-votes twice (the same vote), a variant of the ballot-copy, pointed out to us by Rønne [41], is possible in the weeding version. The attacker intercepts or delays the first honest vote, and casts this

ballot in his name. The same ballot by the honest voter is than removed through weeding. However, as the honest voter casts a second (differently randomised) ballot the tally is biased by containing an additional vote for the honest voter's candidate. This attack is found by DEEPSEC. We can show that the ZKP version does not suffer from this attack as ballots are linked to voter identities and cannot be cast on behalf of someone else. Besides note that, while PaV is a priori in the scope of AKISS, it failed to produce a proof: AKISS only approximates trace equivalence of non-determinate processes and finds a false attack here.

Finally we note that the BAC, Prêt-à-Voter and Helios protocols are not action-determinate and therefore do not benefit from the POR optimisation, which explains the much higher verification times when increasing the sessions. Nevertheless, as exemplified by the Helios hR-W example, attacks may be found very efficiently, as it generally does not require to explore the entire state space.

7. Conclusion and future work

In this paper we have studied automated verification of equivalence properties, encompassing both theoretical and practical aspects. We provide tight complexity results for static equivalence, trace equivalence and labelled bisimilarity, summarised in fig. 1. In particular we show that deciding trace equivalence and labelled bisimilarity for a bounded number of sessions is coNEXP complete for subterm convergent destructor rewrite systems. Finally, we implement our procedure in the DEEPSEC prototype. As demonstrated through an extensive benchmark (fig. 7), our tool is broad in scope and efficient compared to other tools.

Our work opens several directions for future work. First, we wish to lift the restriction of subterm convergent equational theories. Even though the problem becomes quickly undecidable for more general rewrite theories, we plan to design a partially correct, i.e., sound, complete, but not necessarily terminating, procedure, as the procedure underlying the AKISS tool [19]. Second, we plan to avoid the restriction to destructor rewrite systems to more general ones. From the complexity point of view we envision to study *parametrised* complexity (taking the rewrite system, or the degree of non-determinism as a parameter). This may increase our understanding of which parts of the input are responsible of the high complexity, and guide further optimisations. We have seen that the POR techniques have dramatically increased the tool's performances on action-determinate processes. We wish to develop similar techniques for more general classes of processes.

Acknowledgment

The research leading to these results has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation program (grant agreements No 645865-SPOOC), as well as from the French National Research Agency (ANR) under the project TECAP (ANR-17-CE39-0004-01).

	Protocol (# of roles)		Akiss	APTE	SPEC	Sat-Eq	DeepSec
Strong secrecy	Denning-Sacco	3	✓ <1s	✓ <1s	✓ 11s	✓ <1s	✓ <1s
		6	✓ <1s	✓ 1s	OM	✓ <1s	✓ <1s
		7	✓ 6s	✓ 3s		✓ <1s	✓ <1s
		10	OM	✓ 9m49		✓ <1s	✓ <1s
		12		⏱		✓ <1s	✓ <1s
		14				✓ <1s	✓ <1s
		17				✓ <1s	✓ 1s
		29				✓ <1s	✓ 6s
	Wide Mouth Frog	3	✓ <1s	✓ <1s	✓ 5s	✓ <1s	✓ <1s
		6	✓ <1s	✓ <1s	✓1h11m	✓ <1s	✓ <1s
		7	✓ <1s	✓ 1s	OM	✓ <1s	✓ <1s
		10	✓ 10s	✓ 3m35		✓ <1s	✓ 1s
		12	✓22m16s	⏱		✓ <1s	✓ <1s
		14	OM			✓ <1s	✓ <1s
		17				✓ <1s	✓ 1s
		23				✓ <1s	✓ 3s
	Yahalom-Lowe	3	✓ <1s	✓ <1s	✓ 7s	✓ <1s	✓ <1s
		6	✓ 2s	✓ 41s	OM	✓ <1s	✓ <1s
		7	✓ 42s	✓34m38s		✓ 1s	✓ <1s
		10	OM	⏱		✓ 1s	✓ <1s
		12				✓ 4s	✓ 2s
		14				✓ 7s	✓ 2s
		17				✓ 12s	✓ 8s
	Needham-Schroeder-Lowe	2	✓ <1s	✓ <1s	✓ 30s		✓ <1s
		4	✓ 3s	BUG	⏱	✗	✓ <1s
		8	OM				✓ 2s
		12					✓ 21s
	Otway-Rees	3	✓ 28s	✓ 2s	✓58m9s		✓ <1s
		6	OM	OM	⏱		✓ <1s
		7				✗	✓ <1s
		10					✓ 3s
		14					✓ 5m28s
Anonymity	Private Authentication	2	✓ <1s	✓ <1s			✓ <1s
		4	✓ <1s	✓ 1s			✓ <1s
		6	✓ 21s	✓ 4m18s	✗	✗	✓ <1s
		8	OM	⏱			✓ 1s
		10					✓ 2s
		15					✓ 32s
	3G-AKA	4	✓ 1m34s	✓ 1h38m	✗	✗	✓ 0s
		6	OM	⏱			✓ 3s
	Passive Authentication	2	✓ <1s	✓ <1s			✓ <1s
		4	✓ <1s	✓ 1s			✓ <1s
		6	✓ 2m22s	✓ 1m26s			✓ <1s
		7	✓ 1h42m	✓ 1m40s	✗	✗	✓ 1s
		9	⏱	✓ 1h55m			✓ <1s
		15		⏱			✓ 4s
		21					✓ 8s
Unlinkability	3G-AKA	4	✓ 1m35s	✓ 1h23m	✗	✗	✓ <1s
		6	OM	⏱			✓ 2s
	Passive Authentication	4	✓ <1s	✓ 1s			✓ <1s
		6	✓ 2m15s	✓ 1m27s			✓ <1s
		7	✓ 1h40m	✓ 1m44s	✗	✗	✓ 1s
		9	⏱	✓ 2h08m			✓ <1s
		15		⏱			✓ 9s
		21					✓ 15s
	BAC	4	OM	↯ 38m56s	✗	✗	↯ 1s
		6		⏱			⏱
Vote privacy	Prêt-à-Voter	6	✗	✗	✗	✗	✓ 2s
	Helios Vanilla	6	↯ 47s	↯ <1s	✗	✗	↯ <1s
	Helios NR-W	6					✓ 1s
	Helios NR-ZKP	6					✓ 2s
	Helios dR-W	10					✓30m 24s
	Helios dR-ZKP	10	OM	✗	✗	✗	✓ 9m 26s
	Helios hR-W	11					↯ 2s
	Helios hR-ZKP	11					✓ 2h 42m

✓ successful verification ↯ attack found ✗ out of scope
OM out of memory/stack overflow ⏱ timeout (12 hours)

Figure 7: Benchmark results

References

[1] Deepsec: Deciding equivalence properties in security protocols. https://deepsec-prover.github.io, Jan. 2018.

[2] Deepsec: technical report. https://hal.inria.fr/hal-01698177/document, Jan. 2018.

[3] M. Abadi and V. Cortier. Deciding knowledge in security protocols under equational theories. *Theoretical Computer Science*, 387(1-2):2–32, Nov. 2006.

[4] M. Abadi and C. Fournet. Mobile Values, New Names, and Secure Communication. In *28th Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115. ACM Press, 2001.

[5] M. Abadi and C. Fournet. Private authentication. *Theoretical Computer Science*, 322(3):427–476, Sept. 2004.

[6] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Inf. Comput.*, 148(1):1–70, 1999.

[7] B. Adida. Helios: web-based open-audit voting. In *17th conference on Security symposium (SS'08)*, pages 335–348. USENIX Association, 2008.

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

[9] M. Arapinis, V. Cortier, and S. Kremer. When are three voters enough for privacy properties? In *Proceedings of the 21st European Symposium on Research in Computer Security (ESORICS'16)*, volume 9879 of *Lecture Notes in Computer Science*, pages 241–260. Springer, Sept. 2016.

[10] M. Arapinis, L. Mancini, E. Ritter, M. Ryan, N. Golde, K. Redon, and R. Borgaonkar. New privacy issues in mobile telephony: fix and verification. In *19th Conference on Computer and Communications Security (CCS'12)*, pages 205–216. ACM Press, 2012.

[11] A. Armando, D. A. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P. H. Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA tool for the automated validation of internet security protocols and applications. In *17th International Conference on Computer Aided Verification (CAV'05)*, Lecture Notes in Computer Science, pages 281–285. Springer, 2005.

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

[13] D. A. Basin, J. Dreier, and R. Sasse. Automated symbolic proofs of observational equivalence. In *22nd Conference on Computer and Communications Security (CCS'15)*, pages 1144–1155. ACM Press, 2015.

[14] M. Baudet. Deciding security of protocols against off-line guessing attacks. In *12th Conference on Computer and Communications Security (CCS'05)*, pages 16–25. ACM Press, 2005.

[15] B. Blanchet, M. Abadi, and C. Fournet. Automated Verification of Selected Equivalences for Security Protocols. In *Symposium on Logic in Computer Science (LICS'05)*, pages 331–340. IEEE Comp. Soc. Press, June 2005.

[16] B. Blanchet and B. Smyth. Automated reasoning for equivalences in the applied pi calculus with barriers. In *29th Computer Security Foundations Symposium (CSF'16)*, pages 310–324. IEEE Comp. Soc. Press, 2016.

[17] B. Blanchet, B. Smyth, and V. Cheval. *Automatic Cryptographic Protocol Verifier, User Manual and Tutorial*, 2016.

[18] M. Boreale and L. Trevisan. A complexity analysis of bisimilarity for value-passing processes. *Theor. Comput. Sci.*, 238(1-2):313–345, 2000.

[19] R. Chadha, V. Cheval, Ş. Ciobâcă, and S. Kremer. Automated verification of equivalence properties of cryptographic protocol. *ACM Transactions on Computational Logic*, 23(4):1–32, 2016.

[20] V. Cheval. Apte: an algorithm for proving trace equivalence. In *20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14)*, volume 8413 of *Lecture Notes in Computer Science*, pages 587–592. Springer, 2014.

[21] V. Cheval and B. Blanchet. Proving more observational equivalences with ProVerif. In *Proc. 2nd Conference on Principles of Security and Trust (POST'13)*, volume 7796 of *Lecture Notes in Computer Science*, pages 226–246. Springer, 2013.

[22] V. Cheval, H. Comon-Lundh, and S. Delaune. Trace equivalence decision: Negative tests and non-determinism. In *18th Conference on Computer and Communications Security (CCS'11)*, pages 321–330. ACM Press, 2011.

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

[24] Y. Chevalier and M. Rusinowitch. Decidability of equivalence of symbolic derivations. *Journal of Automated Reasoning*, 48(2):263–292, 2010.

[25] R. Chrétien, V. Cortier, and S. Delaune. Decidability of trace equivalence for protocols with nonces. In *Proceedings of the 28th IEEE Computer Security Foundations Symposium (CSF'15)*, pages 170–184. IEEE Computer Society Press, 2015.

[26] R. Chrétien, V. Cortier, and S. Delaune. From security protocols to pushdown automata. *ACM Transactions on Computational Logic*, 17(3):1–45, November 2015.

[27] Ş. Ciobâcă, S. Delaune, and S. Kremer. Computing knowledge in security protocols under convergent equational theories. *Journal of Automated Reasoning*, 48(2):219–262, 2012.

[28] H. Comon-Lundh and S. Delaune. The finite variant property: How to get rid of some algebraic properties. In *16th International Conference on Rewriting Techniques and Applications (RTA'05)*, volume 3467 of *Lecture Notes in Computer Science*, pages 294–307. Springer, 2005.

[29] B. Conchinha, D. A. Basin, and C. Caleiro. Efficient decision procedures for message deducibility and static equivalence. In *Proc. 7th International Workshop on Formal Aspects of Security and Trust (FAST'10)*, volume 6561 of *Lecture Notes in Computer Science*, pages 34–49. Springer, 2010.

[30] V. Cortier, S. Delaune, and A. Dallon. Sat-equiv: an efficient tool for equivalence properties. In *Proceedings of the 30th IEEE Computer Security Foundations Symposium (CSF'17)*, pages 481–494. IEEE Computer Society Press, 2017.

[31] V. Cortier, N. Grimm, J. Lallemand, and M. Maffei. A type system for privacy properties. In *24th ACM Conference on Computer and Communications Security (CCS'17)*, Dallas, USA, October 2017. ACM. To appear.

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

[33] C. J. F. Cremers. The scyther tool: Verification, falsification, and analysis of security protocols. In *Proc. 20th International Conference on Computer Aided Verification (CAV'08)*, volume 5123 of *Lecture Notes in Computer Science*, pages 414–418. Springer, 2008.

[34] S. Delaune, S. Kremer, and M. D. Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17(4):435–487, July 2009.

[35] D. Dolev and A. Yao. On the security of public key protocols. In *Proc. of the 22nd Symp. on Foundations of Computer Science (FOCS'81)*, pages 350–357. IEEE Comp. Soc. Press, 1981.

[36] N. A. Durgin, P. Lincoln, and J. C. Mitchell. Multiset rewriting and the complexity of bounded security protocols. *Journal of Computer Security*, 12(2):247–311, 2004.

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

[38] L. Hirschi, D. Baelde, and S. Delaune. A method for verifying privacy-type properties: the unbounded case. In *37th IEEE Symposium on Security and Privacy (S&P'16)*, pages 564–581, San Jose, California, USA, 2016. IEEE Comp. Soc. Press.

[39] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I. *Inf. Comput.*, 100(1):1–40, 1992.

[40] C. Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003.

[41] P. Rønne. Personal communication, 2016.

[42] M. Rusinowitch and M. Turuani. Protocol insecurity with a finite number of sessions, composed keys is np-complete. *Theor. Comput. Sci.*, 299(1-3):451–475, 2003.

[43] P. Y. A. Ryan and S. A. Schneider. Prêt-à-voter with re-encryption mixes. In *11th European Symp. On Research In Computer Security (ESORICS'06)*, volume 4189 of *Lecture Notes in Computer Science*, pages 313–326. Springer, 2006.

[44] S. Santiago, S. Escobar, C. Meadows, and J. Meseguer. A formal definition of protocol indistinguishability and its verification using Maude-NPA. In *10th International Workshop on Security and Trust Management STM'14*, volume 8743 of *Lecture Notes in Computer Science*, pages 162–177. Springer, 2014.

[45] B. Schmidt, S. Meier, C. Cremers, and D. Basin. The TAMARIN prover for the symbolic analysis of security protocols. In *25th International Conference on Computer Aided Verification (CAV'13)*, volume 8044 of *Lecture Notes in Computer Science*, pages 696–701. Springer, 2013.

[46] A. Tiu and J. Dawson. Automating open bisimulation checking for the spi-calculus. In *23rd Computer Security Foundations Symposium (CSF'10)*, pages 307–321. IEEE Comp. Soc. Press, 2010.

[47] A. Tiu, N. Nguyen, and R. Horne. SPEC: an equivalence checker for security protocols. In *14th Asian Symposium on Programming Languages and Systems (APLAS'16)*, volume 10017 of *Lecture Notes in Computer Science*, pages 87–95. Springer, 2016.

Appendix A. Encoding calculus extensions

In section 3.2 we introduced calculus extensions—sum, guess, circuit—to make establishment of complexity lower bounds easier. Here we present how the new operators can be encoded within the original calculus. More precisely we construct a translation $[\![\cdot]\!]$ with the following properties:

Lemma 10. *If P is a closed plain process in the extended syntax, $[\![P]\!]$ is a closed plain process in the original syntax computable in polynomial time in $|P|$. Besides, after casting $\approx \in \{\approx_t, \approx_\ell\}$ to the extended semantics, we have*

$$(\{\!\{[\![P]\!]\}\!\}, \emptyset) \approx (\{\!\{P\}\!\}, \emptyset)$$

Since the extended semantics does not affect processes in the original syntax, this lemma justifies that equivalence in the extended calculus reduces in polynomial time to equivalence in the original calculus. The construction of $[\![\cdot]\!]$ is detailed below and the proof of the above lemma can be found in the technical report [2].

Sums. The non-deterministic choice can be implemented by a standard encoding, using an internal communication on a fresh private channel:

$$[\![P + Q]\!] \triangleq \overline{s}\langle s\rangle \mid s(x).[\![P]\!] \mid s(y).[\![Q]\!]$$

where $s \in \mathcal{N}_{\mathsf{prv}}$ and $x, y \in \mathcal{X}^1$ are fresh.

Guesses. Using the same mechanism as sums, we can encode the binary guess.

$$[\![\mathtt{Choose}(x).P]\!] \triangleq \overline{s}\langle \mathsf{0}\rangle \mid \overline{s}\langle \mathsf{1}\rangle \mid s(x).[\![P]\!]$$

where $s \in \mathcal{N}_{\mathsf{prv}}$ is fresh. We recall that $\mathsf{0}$ and $\mathsf{1}$ are two constructor symbols of the signature.

Circuits. Seeing a circuit edge as a private channel, a circuit computation can easily be simulated by internal communication. Let $\Gamma : \mathbb{B}^m \to \mathbb{B}^n$ be a circuit. For simplicity we only consider the case where gates have two inputs and two outputs: handling lower arities is straightforward. If $(c_1, c_2, g, c_3, c_4) \in \Gamma$ is such a gate, we first define notation:

$$[\![c_1, c_2, g, c_3, c_4]\!] \triangleq c_1(x).c_2(y). \prod_{b,b' \in \mathbb{B}} P^{c_3,c_4}_{g,b,b'}$$

$$\text{with } P^{c,c'}_{g,b,b'} \triangleq \mathsf{if}\ x = b\ \mathsf{then}\ \mathsf{if}\ y = b'\ \mathsf{then}\ (\overline{c}\langle g(b,b')\rangle \mid \overline{c'}\langle g(b,b')\rangle)$$

where $c_1, c_2, c_3, c_4 \in \mathcal{N}_{\mathsf{prv}}$. Note that $g(b, b')$ denotes the boolean obtained from the truth table of g, for b, b' fixed: in particular g is *not* a function symbol of the signature $\mathcal{F}$. It finally leads to

$$[\![\vec{x} \leftarrow \Gamma(\vec{b}).P]\!] \triangleq P_{\mathsf{args}} \mid P_{\mathsf{compute}} \mid P_{\mathsf{return}}$$

$$\text{where } P_{\mathsf{args}} = \prod_{k=1}^{m} \overline{c_{i_k}}\langle b_k\rangle$$

$$\text{and } P_{\mathsf{compute}} = \prod_{(c_1,c_2,g,c_3,c_4)\in\Gamma} [\![c_1, c_2, g, c_3, c_4]\!]$$

$$\text{and } P_{\mathsf{return}} = c_{o_1}(x_1) \dots c_{o_n}(x_n).[\![P]\!]$$

where $(c_{i_k})_{k=1}^m$ (resp. $(c_{o_k})_{k=1}^n$) are the isolated input (resp. output) edges of Γ.

Appendix B. Full symbolic semantics

Now we present the remaining rules of the symbolic semantics, partly presented in fig. 4. Note the formula introduced by the rule S-ELSE. It has to take into account that an equality test fails if the compared terms are not messages—even if they are syntactically equal. For that we define, if σ is a substitution, the formula

$$\neg\sigma \quad \triangleq \quad \forall \tilde{z}_\delta. \bigvee_{x \in dom(\sigma)} x \neq^? x\sigma$$

where $\tilde{z}_\delta = vars(\sigma) \setminus dom(\sigma)$. The rules can be found in fig. 8.

$$(\mathcal{P} \cup \{\!\{0\}\!\}, \mathcal{C}) \xrightarrow{\varepsilon}_{\mathsf{s}} (\mathcal{P}, \mathcal{C}) \qquad \text{(S-NULL)}$$

$$(\mathcal{P} \cup \{\!\{\mathsf{if}\ u = v\ \mathsf{then}\ Q_1\ \mathsf{else}\ Q_2\}\!\}, (\Phi, \mathsf{D}, \mathsf{E}^1)) \xrightarrow{\varepsilon}_{\mathsf{s}} (\mathcal{P} \cup \{\!\{Q_1\}\!\}, (\Phi, \mathsf{D}, \mathsf{E}^1 \wedge \sigma)) \quad \text{if } \sigma \in mgu_{\mathcal{R}}(u\mu\downarrow =^? v\mu\downarrow) \qquad \text{(S-THEN)}$$

$$(\mathcal{P} \cup \{\!\{\mathsf{if}\ u = v\ \mathsf{then}\ Q_1\ \mathsf{else}\ Q_2\}\!\}, (\Phi, \mathsf{D}, \mathsf{E}^1)) \xrightarrow{\varepsilon}_{\mathsf{s}} (\mathcal{P} \cup \{\!\{Q_2\}\!\}, (\Phi, \mathsf{D}, \mathsf{E}^1 \wedge \bigwedge_{\delta \in mgu_{\mathcal{R}}(u\mu\downarrow, v\mu\downarrow)} \neg\delta)) \qquad \text{(S-ELSE)}$$

$$(\mathcal{P} \cup \{\!\{\overline{u}\langle t\rangle.Q_1, v(x).Q_2\}\!\}, (\Phi, \mathsf{D}, \mathsf{E}^1)) \xrightarrow{\varepsilon}_{\mathsf{s}} (\mathcal{P} \cup \{\!\{Q_1, Q_2\{x \to t\}\}\!\}, (\Phi, \mathsf{D}, \mathsf{E}^1 \wedge \sigma)) \quad \text{if } \sigma \in mgu_{\mathcal{R}}(u\mu\downarrow =^? v\mu\downarrow, t\mu\downarrow =^? t\mu\downarrow) \qquad \text{(S-COMM)}$$

$$(\mathcal{P} \cup \{\!\{P \mid Q\}\!\}, (\Phi, \mathsf{D}, \mathsf{E}^1)) \xrightarrow{\varepsilon}_{\mathsf{s}} (\mathcal{P} \cup \{\!\{P, Q\}\!\}, (\Phi, \mathsf{D}, \mathsf{E}^1)) \qquad \text{(S-PAR)}$$

$$(\mathcal{P} \cup \{\!\{u(x).Q\}\!\}, (\Phi, \mathsf{D}, \mathsf{E}^1)) \xrightarrow{Y(X)}_{\mathsf{s}} (\mathcal{P} \cup \{\!\{Q\}\!\}, (\Phi, \mathsf{D} \wedge X \vdash^? x \wedge Y \vdash^? y, \mathsf{E}^1 \wedge \sigma)) \quad \text{if } \sigma \in mgu_{\mathcal{R}}(y =^? u\mu\downarrow) \text{ and } X, Y \in \mathcal{X}^2_{:|\Phi|} \qquad \text{(S-IN)}$$

$$(\mathcal{P} \cup \{\!\{\overline{u}\langle t\rangle.Q\}\!\}, (\Phi, \mathsf{D}, \mathsf{E}^1)) \xrightarrow{\overline{Y}\langle \mathsf{ax}_n\rangle}_{\mathsf{s}} (\mathcal{P} \cup \{\!\{Q\}\!\}, (\Phi \cup \{\mathsf{ax}_n \mapsto t\sigma\downarrow\}, \mathsf{D} \wedge Y \vdash^? y, \mathsf{E}^1 \wedge \sigma)) \quad \text{if } \sigma \in mgu_{\mathcal{R}}(y =^? u\mu\downarrow, t\mu\downarrow =^? t\mu\downarrow),\ y \text{ is fresh, } X \in \mathcal{X}^2_{:n} \text{ and } n = |\Phi| + 1 \qquad \text{(S-OUT)}$$

Figure 8: Symbolic semantics (where $\mu = mgu(\mathsf{E}^1|_=)$)

Appendix C. Overview of case-distinction rules

At the end of section 5.2, we mentioned the use of case-distinction rules to generate the partition tree. We give here more details about these rules. Their purpose is to transform a set of extended symbolic process into sets of extended symbolic process where the constraint systems in each set have statically equivalence solutions. For this reason, case-distinction rules are presented as (three) transition rules operating on sets of sets of extended symbolic process.

Extended constraint systems. We need more formalism on extended constraint systems. First, we consider that recipes can now contain second-order variables; that is, the set of recipes is $\mathcal{T}(\mathcal{F}, \mathcal{N}_{\mathsf{pub}} \cup \mathcal{AX} \cup \mathcal{X}^2)$. Besides, we extend the notion of formula in several ways. Deduction fact are generalised to the form $\xi \vdash^? u$ where ξ is a recipe. Besides, we introduce two new atomic formulas:

$$\xi =^? \xi' \text{ (\textit{recipe equation})} \quad \text{and} \quad \xi =^?_f \xi' \text{ (\textit{equality fact})}$$

where ξ, ξ' are recipes. Recipe equations are similar to (first-order) equations whereas equality facts state that two recipes deduce the same term; namely

$$(\Phi, \Sigma, \sigma) \models \xi =^?_f \xi' \text{ iff } Msg(\xi\Sigma\Phi), Msg(\xi'\Sigma\Phi), \text{ and } \xi\Sigma\Phi\downarrow = \xi'\Sigma\Phi\downarrow$$

Moreover, we distinguish two forms of first-order formulas called *deduction* (resp. *equality*) *formulas*:

$$\forall S.H \Leftarrow C_1 \wedge \ldots \wedge C_n$$

where

- S is a set of (first- and second-order) variables;
- H is a deduction (resp. equality) fact;
- for all $i \in \mathbb{N}_n$, C_i is either a deduction fact of the form $X \vdash^? u$ or an equation.

This intuitively states that the statement H holds under premisses $C_1, \ldots, C_n$. We finally get to the notion of *extended constraint systems*.

Definition 14. An extended constraint system $\mathcal{C}$ is a tuple $(\Phi, \mathsf{D}, \mathsf{E}^1, \mathsf{E}^2, \mathsf{K}, \mathsf{F})$ where

- $(\Phi, \mathsf{D}, \mathsf{E}^1)$ is a constraint system;
- E^2 is a conjunction of recipe equations or formulas of the form $\forall Y_1, \ldots, Y_k. \bigvee_{j=1}^{p} \xi_j \neq^? \zeta_j$;
- K is a conjunction of deduction facts;
- F is a conjunction of deduction or equality formulas.

E^2 is similar to E^1 but gathers constraints on recipes. On the contrary, K and F are not constraints but represent the *attacker knowledge*:

- deduction facts in K materialise the deduction capabilities of the attacker;
- deduction formulas in F reason about potentially deducible terms.

After an output transition, our decision procedure distinguishes whether the new output term can increase the deduction capabilities of the attacker. This is done by adding a formula to F which is then simplified into a deduction fact, so that we can easily decide whether

a) it induces a new deducible term (in which case it is added to K), or
b) it is a *consequence of the deduction facts already in* K.

More generally we say that (ξ, u)—or simply u—is a consequence of a set of deduction facts S when u can be deduced from S (using recipe ξ). The notion of solution of extended constraint systems carries out this idea: we say that (Σ, σ) is a solution of $(\Phi, \mathsf{D}, \mathsf{E}^1, \mathsf{E}^2, \mathsf{K}, \mathsf{F})$ when $(\Phi\sigma, \Sigma, \sigma) \models \mathsf{D} \wedge \mathsf{E}^1 \wedge \mathsf{E}^2$ with the two requirements that

1) all recipes in Σ must be a consequence of the attacker knowledge K;
2) recipes in Σ must be chosen in a *uniform* way: two different recipes in the solution must deduce different protocol terms.

Rule SAT. Le us now present the three case-distinction rules. We recall that these rules operate on sets of sets of constraint systems $\mathcal{S}$ representing the children partition in the partition tree. The first rule, SAT, helps obtaining the requirement that $n \in \mathcal{S}$ shall contain constraint systems with the same second-order solutions. Thus, the rule will perform

a case distinction on whether solutions of all constraint systems in the same set $n \in \mathcal{S}$ are instance of a common mgs Σ. This is formalised by the rule

$$\mathcal{S} \cup \{\{(\mathcal{P}_i, \mathcal{C}_i, \mathcal{C}_i^e)\}_{i=1}^p\} \to \mathcal{S} \cup \{\mathcal{E}_{\mathsf{pos}}, \mathcal{E}_{\mathsf{neg}}\} \qquad \text{(SAT)}$$
$$\text{with } \mathcal{E}_{\mathsf{pos}} = \{(\mathcal{P}_i, \mathcal{C}_i, \mathcal{C}_i^e{:}\Sigma)\}_{i=1}^p$$
$$\text{and } \mathcal{E}_{\mathsf{neg}} = \{(\mathcal{P}_i, \mathcal{C}_i, \mathcal{C}_i^e[\mathsf{E}^2 \wedge \neg\Sigma])\}_{i=1}^p$$

where $\Sigma \in mgs(\mathcal{C}_j^e)$ for some j, and $\mathcal{C}{:}\Sigma$ is the application of Σ to $\mathcal{C}$. The definition of $\mathcal{C}{:}\Sigma$ is actually a little more involved than a plain application of substitution—as some structural invariants have to be maintained for the sake of the procedure—but we omit details for readability.

This rule is also applied to solve syntactic disequations and formulas in F. For this usage, Σ is not computed as a simple mgs of $\mathcal{C}_j^e$ but embodies additionial constraints. For exemple, if $\mathsf{E}^1(\mathcal{C}_j^e)$ contains the disequation $x \neq^? v$, the rule will partition the solutions depending on whether they satisfy or such disequations. This is done by considering the mgs of $\mathcal{C}_j^e\sigma$ where $\sigma = \{x \to v\}$.

Rule EQ. The second rule, EQ, focuses on static equivalence between constraint systems of a same node of the partition tree. For instance, given two deduction facts $\xi \vdash^? u, \zeta \vdash^? v$ in some attacker knowledge K, the rule distinguishes solutions in which the two deduction facts deduce the same protocol term.

$$\mathcal{S} \cup \{\{(\mathcal{P}_i, \mathcal{C}_i, \mathcal{C}_i^e)\}_{i=1}^p\} \to \mathcal{S} \cup \{\mathcal{E}_{\mathsf{pos}}, \mathcal{E}_{\mathsf{neg}}\} \qquad \text{(EQ)}$$
$$\text{with } \mathcal{E}_{\mathsf{pos}} = \{(\mathcal{P}_i, \mathcal{C}_i, \mathcal{C}_i^e{:}\Sigma[\mathsf{F} \cup \{\psi_i\}])\}_{i=1}^p$$
$$\text{and } \mathcal{E}_{\mathsf{neg}} = \{(\mathcal{P}_i, \mathcal{C}_i, \mathcal{C}_i^e[\mathsf{E}^2 \wedge \neg\Sigma])\}_{i=1}^p$$

where $\Sigma \in mgs(\mathcal{C}_j^e mgu(u, v))$ for some j and deduction facts $(\xi \vdash^? u), (\zeta \vdash^? v) \in \mathsf{K}(\mathcal{C}_j^e)$. The formulas ψ_i state that recipes ξ and ζ should deduce the same protocol terms:

$$\psi_i = (\xi\Sigma =_f^? \zeta\Sigma \Leftarrow u_i =^? v_i)$$

where $(\xi \vdash^? u_i), (\zeta \vdash^? v_i) \in \mathsf{K}(\mathcal{C}_i^e)$.

Rule REW. The third rule, REW, focuses on intruder knowledge, that is K. Typically, when the last symbolic transition is an output, the rule applies rewrite rules on this output to determine whether the attacker can deduce new messages from it. For example, $\mathsf{sdec}(\mathsf{senc}(x, y), y) \to x$ leads to

$$\mathcal{S} \cup \{\{(\mathcal{P}_i, \mathcal{C}_i, \mathcal{C}_i^e)\}_{i=1}^p\} \to \mathcal{S} \cup \{\mathcal{E}_{\mathsf{pos}}, \mathcal{E}_{\mathsf{neg}}\} \qquad \text{(REW)}$$
$$\text{with } \mathcal{E}_{\mathsf{pos}} = \{(\mathcal{P}_i, \mathcal{C}_i, \mathcal{C}_i^e{:}\Sigma[\mathsf{F} \cup \{\psi_i\}])\}_{i=1}^p$$
$$\text{and } \mathcal{E}_{\mathsf{neg}} = \{(\mathcal{P}_i, \mathcal{C}_i, \mathcal{C}_i^e[\mathsf{E}^2 \wedge \neg\Sigma])\}_{i=1}^p$$

where for some j, $(\xi \vdash^? u) \in \mathsf{K}(\mathcal{C}_j^e)$, $\Sigma = mgs(\mathcal{C}_j^{e\prime})$ with

$$\mathcal{C}_j^{e\prime} = \mathcal{C}_j^e[\mathsf{D} \wedge X \vdash^? y][\mathsf{E}^1 \wedge \mathsf{senc}(x, y) =^? u]$$

with X, x, y fresh. Here the solutions $\mathcal{C}_j^{e\prime}$ represents the solutions of $\mathcal{C}_j^e$ in which we can apply a decryption on ξ. Hence the formula ψ_i, stating that $\mathsf{sdec}(\xi\Sigma, X\Sigma)$ should deduce a message:

$$\psi_i = \forall X, x, y.(\mathsf{sdec}(\xi\Sigma, X\Sigma) \vdash^? x) \Leftarrow (y =^? v_i \wedge \mathsf{senc}(x, y) =^? u_i)$$

where $\xi \vdash^? u_i \in \mathsf{K}(\mathcal{C}_i^e)$ and $(X\Sigma, v_i)$ is a consequence of $\mathsf{K}(\mathcal{C}_i^e{:}\Sigma)$ and $\mathsf{D}(\mathcal{C}_i^e{:}\Sigma)$.

Appendix D. Bounding the size of solutions

One key argument to obtain the coNEXP complexity is the bound on the sizes of mgs' of partition trees (theorem 8). Let us give some insight about this result. In all case-distinction rules, we apply mgs' Σ of constraint systems. Therefore we need to bound

1) the size of such substitutions Σ
2) the number of applications of the three rules.

We only detail the first (polynomial) bound here and omit the second (exponential) bound due to lack of space. Technically, we use a measure counting the protocol subterms not already deduced by a recipe in E^2, K and D:

$$\mathcal{M}(\mathcal{C}^e) = \left| \left\{ t \in PT(\mathcal{C}^e) \setminus \mathcal{X}^1 \;\middle|\; \begin{array}{l} \text{for all } \xi \in st_c(\mathcal{C}^e) \\ \xi \notin \mathcal{X}^2,\ (\xi, t) \text{ is} \\ \text{not consequence of} \\ \mathsf{K}(\mathcal{C}^e) \text{ and } \mathsf{D}(\mathcal{C}^e) \end{array} \right\} \right|$$

where

- $\mu^i = mgu(\mathsf{E}^i(\mathcal{C}^e)|_=)$ for $i \in \{1, 2\}$;
- $PT(\mathcal{C}^e)$ is the set of protocol subterms in $\Phi(\mathcal{C}^e)\mu^1$, $\mathsf{K}(\mathcal{C}^e)\mu^1$, $\mathsf{D}(\mathcal{C}^e)\mu^1$ and μ^1;
- $st_c(\mathcal{C}^e)$ is a set of recipes subterms of $img(\mu^2)$ directly consequence of $\mathsf{K}(\mathcal{C}^e)\mu^2$ and $\mathsf{D}(\mathcal{C}^e)$.

The important argument is that this measure decreases by application of mgs, namely

$$\forall \Sigma \in mgs(\mathcal{C}^e),\ \mathcal{M}(\mathcal{C}^e{:}\Sigma) \leqslant \mathcal{M}(\mathcal{C}^e)$$

This relation also holds for any constraint systems with the same *recipe structure* as $\mathcal{C}^e$, i.e. constraint systems that differs from $\mathcal{C}^e$ by its protocol terms but not its recipes. In particular, in all case-distinction rules described in the previous appendix, we have

$$\forall i \in \mathbb{N}_p,\ \mathcal{M}(\mathcal{C}_i^e{:}\Sigma) \leqslant \mathcal{M}(\mathcal{C}_i^e)$$

Additionally, the growth of $st_c(\mathcal{C}^e)$ after application of mgs can also be bounded—with a dependency in $|\mathcal{F}|$. More precisely, if α the maximal arity of $\mathcal{F}$, we proved that for all $\Sigma \in mgs(\mathcal{C}^e)$:

$$|R(\mathcal{C}^e{:}\Sigma)| - |R(\mathcal{C}^e)| \leqslant \alpha(\mathcal{M}(\mathcal{C}^e) - \mathcal{M}(\mathcal{C}^e{:}\Sigma))$$

Let us then consider the two initial processes P_1 and P_2 for which the procedure is intended to generate a partition tree. Let us also consider a constraint system $\mathcal{C}$ in some node of T. Seeing $\mathcal{C}$ as an extended constraint system where $\mathsf{K}(\mathcal{C}) = \emptyset$ and $\mathsf{E}^2(\mathcal{C}) = \top$, we easily obtain

$$\mathcal{M}(\mathcal{C}^e) \leqslant \mathcal{M}(\mathcal{C}) \qquad \text{and} \qquad \mathcal{M}(\mathcal{C}) \leqslant |PT(\mathcal{C})|$$

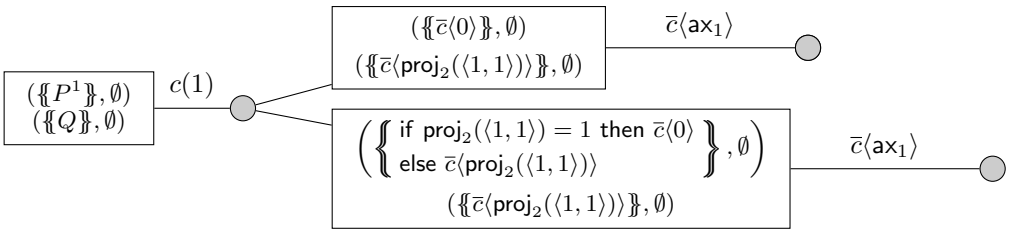


Figure 9: A concrete witness of $(\{\!\{P^1\}\!\},\emptyset) \not\approx_\ell (\{\!\{Q\}\!\},\emptyset)$.

Finally, from the the symbolic semantics (fig. 4), we can see that a protocol term in $\mathcal{C}$ is a protocol term in P_1 or P_2 on which a mgu σ has been applied. This lead to

$$|PT(\mathcal{C})| \leqslant 2|P_1,P_2|_{\mathsf{dag}}(1+|\mathcal{R}|_{\mathsf{dag}})$$

Above, we showed the relation between $|R(\mathcal{C}^e{:}\Sigma)|$ and $|R(\mathcal{C}^e)|$ when $\Sigma \in mgs(\mathcal{C}^e)$. However, for case distinction rules, we also compute most general solutions of the constraint system with additional deduction facts and protocol term equations. Hence, to compute $|R(\mathcal{C}^e{:}\Sigma)|$, we need to also consider the terms introduced by rewrite rules and disequations. For instance, with the rule REW, we have that for all i, $|R(\mathcal{C}_i^e{:}\Sigma)| - |R(\mathcal{C}_i^e)| \leqslant |\mathcal{F}|(2|P_1,P_2|_{\mathsf{dag}}(1+|\mathcal{R}|_{\mathsf{dag}})+|\mathcal{R}|_{\mathsf{dag}}(2+|\mathcal{F}|))$. This shows that the DAG size of recipes in constraint systems increases at most polynomially in P_1,P_2, $\mathcal{F}$ and $\mathcal{R}$.

Appendix E. Decidability of labelled bisimilarity

In the body of the paper, we outlined quite precisely the decision procedure for trace equivalence through lemma 7: we explain here how to adapt this approach to labelled bisimilarity. Typically, a witness of $A \not\approx_t B$ is a sequence of actions tr, and a set of pairs of recipes—witnessing violations of static equivalence. The case of bisimilarity is more involved. Borrowing vocabulary from game's theory, a witness that $A \not\approx_\ell B$ is an adversary's winning strategy in the bisimulation game. Such a strategy can be modelled as a tree whose nodes are labelled by pairs of processes:

1) for each node n labelled (A,B) such that $A \sim B$, the attacker shall be able to choose an action $a \in \mathcal{A}$ and either a transition $A \xrightarrow{a}_{\mathsf{c}} A'$ or $B \xrightarrow{a}_{\mathsf{c}} B'$;
2) say, for example, that $A \xrightarrow{a}_{\mathsf{c}} A'$ has been chosen. Then the labels of the children of n are all (A',B') where $B \stackrel{a}{\Rightarrow}_{\mathsf{c}} B'$ and $A' \sim B'$.

Example 9. Consider the running examples P^1 and Q. A witness of $(\{\!\{P_1\}\!\},\emptyset) \not\approx_\ell (\{\!\{Q\}\!\},\emptyset)$ is depicted in fig. 9. First, the adversary inputs $\langle 1,1\rangle$, selecting the transition

$$(Q,\emptyset) \xrightarrow{c(\langle 1,1\rangle)}_{\mathsf{c}} (\{\!\{\overline{c}\langle \mathsf{proj}_2(\langle 1,1\rangle)\rangle\}\!\},\emptyset)$$

Whatever the answer of the defender, the adversary can then reach a leave of the tree by choosing the transition

$$(\{\!\{\overline{c}\langle \mathsf{proj}_2(\langle 1,1\rangle)\rangle\}\!\},\emptyset) \xrightarrow{\overline{c}\langle \mathsf{ax}_1\rangle}_{\mathsf{c}} (\{\!\{0\}\!\},\{\mathsf{ax}_1 \mapsto 1\})$$

The defender can indeed not answer to this move without violating static equivalence.

As for trace equivalence we can show that we can decide labelled bisimulation in coNEXP.

Theorem 11. $\mathsf{Equiv}^{\approx_\ell}$ *for convergent subterm destructor rewriting systems is in co*NEXP.

Proof overview. Our first goal is to show that there exists p such that whenever $(\{\!\{P_1\}\!\},\emptyset) \not\approx_\ell (\{\!\{P_2\}\!\},\emptyset)$ then there exists a witness w such that $|\mathsf{w}|_{\mathsf{dag}} < 2^{|P_1,P_2,\mathcal{R}|^p_{\mathsf{dag}}}$. We therefore introduce the notion of *symbolic witness*: a symbolic witness are similar to subtrees of the partition tree $\mathsf{PTree}(P_1,P_2)$ where

- internal nodes n contain a pair of processes of $\Gamma(n)$;
- leaves contain a single process (witnessing impossibility for the defender to answer to the attacker's move).

A *solution* of a symbolic witness is a function that maps each node n to a second-order solution of $\Gamma(n)$. To ensure coherence of a solution we require that the solution of child nodes extend the solution of their parent. We can then show that $(\{\!\{P_1\}\!\},\emptyset) \not\approx_\ell (\{\!\{P_2\}\!\},\emptyset)$ iff there exists a symbolic witness with non-empty solution.

Then, relying on theorem 8 just as for trace equivalence, we want to show that any symbolic witness with non-empty solution entails existence of a concrete witness w such that $|\mathsf{w}|_{\mathsf{dag}} < 2^{|P_1,P_2,\mathcal{R}|^p_{\mathsf{dag}}}$. However, a solution of a symbolic witness requires that the solution of each child extends the solution of the parent—which is not guaranteed by the definition of partition trees in general. Fortunately, our construction appears to ensure the existence of such a solution of the parent node while preserving the exponential size.

Given the concrete witness we obtain a straightforward procedure for non-bisimilarity running in NEXP:

1) We first guess the witness w (structure only). As mentionned above it sufficies to guess w of exponential size.
2) Then we check validity of w. This requires to explore the possibly-exponential number of branches of w and to verify an exponential number of static equivalences (whose frames are at most of exponential size). Therefore, checking non-validity of w can indeed be performed in non-deterministic exponential time—since static equivalence is coNP. □