A procedure for deciding symbolic equivalence between sets of constraint systems[☆]

Vincent Cheval[a], Hubert Comon-Lundh[b], Stéphanie Delaune[c]

[a]*LORIA, CNRS, France*
[b]*LSV, ENS Cachan & Université Paris Saclay, France*
[c]*CNRS & IRISA, France*

Abstract

We consider security properties of cryptographic protocols that can be modeled using the notion of trace equivalence. The notion of equivalence is crucial when specifying privacy-type properties, like anonymity, vote-privacy, and unlinkability.

Infinite sets of possible traces are symbolically represented using deducibility constraints. We describe an algorithm that decides trace equivalence for protocols that use standard primitives (*e.g.,* signatures, symmetric and asymmetric encryptions) and that can be represented using such constraints. More precisely, we consider symbolic equivalence between *sets* of constraint systems, and we also consider disequations. Considering sets and disequations is actually crucial to decide trace equivalence for a general class of processes that may involve else branches and/or private channels (for a bounded number of sessions). Our algorithm for deciding symbolic equivalence between sets of constraint systems is implemented and performs well in practice. Unfortunately, it does not scale up well for deciding trace equivalence between processes. This is however the first implemented algorithm deciding trace equivalence on such a large class of processes.

Keywords: formal methods, verification, security protocols, privacy-type properties, symbolic model.

1. Introduction

The present work is motivated by the decision of security properties of cryptographic protocols. Such protocols are proliferating, because of the expansion of digital communications and the increasing concern on security issues. Finding attacks/proving the security of such protocols is challenging and has a strong societal impact.

In our work, we assume perfect cryptographic primitives: we consider a formal, symbolic, model of execution. Such an assumption may prevent from finding some attacks; the relevance of symbolic models is studied in other research papers (see *e.g.,* [1, 2]), but it is beyond the scope of the present work.

[☆]The research leading to these results has received funding from the ANR project Sequoia.

In this context, the protocols are described in some process algebra, using function symbols to represent the cryptographic primitives and symbolic terms to represent messages. We use the applied pi-calculus [3] in this paper. Many attacks on several protocols have been found during the last 20 years. For example, a flaw has been discovered (see [4]) in the Single-Sign-On protocol used *e.g.,* by Google Apps. These attacks on formal models of protocols can of course be reproduced on the concrete versions of the protocols. Several techniques and tools have been designed for the formal verification of cryptographic protocols. For instance CSP/FdR [5], ProVerif [6], Scyther [7], Avispa [8] and others.

Most results and tools only consider security properties that can be expressed as the (un)reachability of some bad state. For instance, the (weak) secrecy of s is the non-reachability of a state, in which s is known by the attacker. Authentication is also expressed as the impossibility to reach a state, in which two honest parties hold different values for a variable on which they are supposed to agree. In our work, we are interested in more general properties, typically strong secrecy, anonymity, or more generally any privacy-type property that cannot be expressed as the (non) reachability of a given state, but rather requires the *indistinguishability* of two processes. For instance, the strong secrecy of s is specified as the indistinguishability of $P(s)$ from $P(s')$, where s' is a new name. It expresses that the attacker cannot learn any piece of the secret s. Formally, these properties, as well as many other interesting security properties, can be expressed using *trace equivalence*: roughly, two processes P and Q are trace equivalent if any sequence of attacker's actions yields indistinguishable outputs of P and Q.

Some related work. The automated verification of equivalence properties for security protocols was first considered in [9] (within the spi-calculus). ProVerif also checks some equivalence properties (so-called diff-equivalence) [10], which is a stronger equivalence, often too strong, as we will see below with a simple example. More recently, the approach behind the Tamarin verification tool [11] has been extended to check equivalence-based properties [12]. Actually, the equivalence notion is quite similar to the notion of diff-equivalence used in ProVerif, and therefore suffers from the same drawbacks. A few other procedures have been published:

- In [13, 14] a decision procedure for the trace equivalence of bounded deterministic processes is proposed. Their procedure relies on an other procedure for deciding the equivalence of constraint systems such as the one developed by [15] or [16]. In particular, the processes are restricted to be determinate and do not contain (non trivial) conditional branching. Furthermore, the procedure seems to be not well-suited for an implementation. Regarding primitives, these works allow any primitives that are defined using a subterm convergent rewriting system.

- [17] gives a decision procedure for open-bisimulation for bounded processes in the spi-calculus. This procedure has been implemented. The scope is however limited: open-bisimulation is a stronger equivalence notion, and the procedure assumes a fixed set of primitives (in particular no asymmetric encryption) and no conditional branching.

- [18] designs a procedure based on Horn clauses for the class of optimally reducing theories, which encompasses subterm convergent theories. The procedure is sound and complete but its termination is not guaranteed. It applies to determinate processes without replication nor else branches. Moreover, when processes are not determinate, the procedure can be used for both under- and over-approximations of trace equivalence.

Our contribution. Our aim was to design a procedure, which is general enough and efficient enough, so as to automatically verify the security of some simple protocols, such as the private authentication protocol (see Example 1) or the e-passport protocol analysed *e.g.,* in [19]. Both protocols are beyond the scope of any above mentioned results. An extension of ProVerif has been developed allowing one to analyse the private authentication protocol [20]. However, ProVerif is still unable for instance to deal with the e-passport protocol.

Example 1. *We consider the protocol given in [21] designed for transmitting a secret, while not disclosing the identity of the sender. In this protocol, a is willing to engage in a communication with b. However, a does not want to disclose her identity (nor the identity of b) to the outside world. Consider for instance the following protocol:*

$$\begin{array}{lcl} A \to B & : & \mathsf{aenc}(\langle n_a, \mathsf{pub}(ska)\rangle, \mathsf{pub}(skb)) \\ B \to A & : & \mathsf{aenc}(\langle n_a, \langle n_b, \mathsf{pub}(skb)\rangle\rangle, \mathsf{pub}(ska)) \end{array}$$

In words, the agent a (playing the role A) generates a new name n_a and sends it, together with her identity (here public key), encrypted with the public key of b. The agent b (playing the role B) replies by generating a new name n_b, sending it, together with n_a and his identity $\mathsf{pub}(skb)$*, encrypted with the public key of a. More formally, using pattern-matching, and assuming that each agent a holds a private key ska and a public key* $\mathsf{pub}(ska)$*, which is publicly available, the protocol could be written as follows:*

$$\mathit{PrivAuth1} \left\{ \begin{array}{ll} A(ska, pkb): & \nu\, n_a.\, \mathsf{out}(\mathsf{aenc}(\langle n_a, \mathsf{pub}(ska)\rangle, pkb)) \\ B(skb, pka): & \mathsf{in}(\mathsf{aenc}(\langle x, pka\rangle, \mathsf{pub}(skb))). \\ & \quad\quad \nu\, n_b.\, \mathsf{out}(\mathsf{aenc}(\langle x, \langle n_b, \mathsf{pub}(skb)\rangle\rangle, pka)) \end{array} \right.$$

We will later write $A(a, b)$ for $A(ska, \mathsf{pub}(skb))$,$B(b, a)$ for $B(skb, \mathsf{pub}(ska))$, and $B(b, c)$ for $B(skb, \mathsf{pub}(skc))$.

This is fine, as long as only mutual authentication is concerned. Now, if we want to ensure in addition privacy, an attacker should not get any information on who is trying to set up the agreement: $B(b, a)$ and $B(b, c)$ must be indistinguishable. This is not the case in the above protocol. Indeed, an attacker can forge e.g., *the message* $\mathsf{aenc}(\langle \mathsf{pub}(ska), \mathsf{pub}(ska)\rangle, \mathsf{pub}(skb))$ *and find out whether $c = a$ or not by observing whether b replies or not.*

The solution proposed in [21] consists in modifying the process B in such a way that a "decoy" message: $\mathsf{aenc}(\langle n_b, n_b\rangle, \mathsf{pub}(ska))$ *is sent when the received message is not as expected. This message should look like B's other message from the point of view of an outsider. More formally, this can be modelled using the following process:*

$$\mathit{PrivAuth2} \left\{ \begin{array}{ll} A(ska, pkb): & \nu\, n_a.\, \mathsf{out}(\mathsf{aenc}(\langle n_a, \mathsf{pub}(ska)\rangle, pkb)) \\ B'(skb, pka): & \mathsf{in}(x).\, \nu\, n_b. \\ & \quad \mathsf{if}\ \mathsf{proj}_2(\mathsf{adec}(x, skb)) = pka \\ & \quad\quad \mathsf{then}\ \mathsf{out}(\mathsf{aenc}(\langle \mathsf{proj}_1(\mathsf{adec}(x, skb)), \langle n_b, \mathsf{pub}(skb)\rangle\rangle, pka)) \\ & \quad\quad \mathsf{else}\ \mathsf{out}(\mathsf{aenc}(\langle n_b, n_b\rangle, pka)) \end{array} \right.$$

Again, we will later write $B'(b, a)$ for $B'(skb, \mathsf{pub}(ska))$.

Still, this solution is not yet fully satisfactory since, for instance, an attacker could distinguish $B'(b, a)$ from $B'(b, c)$ by sending a message $\mathsf{aenc}(\langle H, \mathsf{pub}(ska)\rangle, \mathsf{pub}(skb))$, where H is a huge message, and observing the size of the output. This can be fixed, considering a process B'' that, additionally, checks the length of its input and sends the decoy message if the length does not match its expectation. In this paper, we will assume perfect cryptography, including length hiding. As shown in [22], the properties of our algorithm allow to extend the procedure to protocols that include length tests.

With length tests, the above protocol is secure in our model. It is actually also computationally secure, for an IND-CCA1 encryption scheme, which satisfies key-privacy (see [23] for instance). When the encryption scheme does not satisfy key-privacy, deciding the computational security would require to give explicitly to the attacker the capability of comparing the encryption keys. We do not include this test in our model. It is likely that it could be added without any significant modification of our decision procedure (extend the rule EQ-FRAME-FRAME*).*

This example shows that the conditional branching in the process B' is necessary. However, such a conditional branching is, in general, beyond the scope of any method that we mentioned so far (though, the extension [20] of ProVerif can handle the above example).

Another example is the e-passport protocol, that was analysed in [19], for which, also, conditional branchings are essential for privacy purposes. Another limitation of the existing works is the determinacy condition: for each attacker's message, there is at most one possible move of the protocol. This condition forces each message to contain the recipient's name, which is a natural restriction, but it also prevents from using private channels (which occur in some natural formalisations).

The results presented in the current paper yield a decision procedure for bounded processes, with conditional branching and non-determinism. It has been implemented and the above examples were automatically analysed.

Some difficulties. One of the main difficulties in the automated analysis of cryptographic protocols is the unbounded possible actions of an attacker: the transition system defined by a protocol is infinitely branching (and also infinite in depth when the protocols under study contain replications - which is not the case here). One of the solutions consists in symbolically representing this infinite set of possible transitions, using *symbolic constraint systems.* More precisely, *deducibility constraints* [24, 25, 26] allow one to split the possible attacker's actions in finitely many sets of actions yielding the same output of the protocol. Each of these sets is represented by a set of deducibility constraints. In this framework, attackers inputs are represented by variables, that must be deducible from the messages available at the time the input is generated and satisfying the conditions that trigger a new message output.

Example 2. *Consider the protocol PrivAuth1 and PrivAuth2given in Example 1. Assume a has sent her message. The message $\mathsf{aenc}(\langle n_a, \langle n_b, \mathsf{pub}(skb)\rangle\rangle, \mathsf{pub}(ska))$ is output only if the attacker's input x can be computed from the messages available and the attacker's initial knowledge $\{\mathsf{pub}(ska), \mathsf{pub}(skb)\}$ and satisfies the test. Formally, x is a solution of* *the constraint system:*

$$\left\{\begin{array}{l}\mathsf{pub}(\mathit{ska}),\ \mathsf{pub}(\mathit{skb}),\ \mathsf{aenc}(\langle n_a, \mathsf{pub}(\mathit{ska})\rangle, \mathsf{pub}(\mathit{skb}))) \vdash^? x \\ \mathsf{proj}_2(\mathsf{adec}(x, \mathit{skb})) =^? \mathsf{pub}(\mathit{ska})\end{array}\right.$$

The symbol $\vdash^?$ is interpreted as the attacker's computing capability. In our case (perfect cryptography), the attacker may only apply function symbols to known messages. This is followed by a normalisation step, in which, for instance, the second projection of a pair gives back the second component, according to the rule $\mathsf{proj}_2(\langle x, y\rangle) \to y$. Similarly, in the protocol PrivAuth2, *the message $\mathsf{aenc}(\langle n_b, n_b\rangle, \mathsf{pub}(\mathit{ska}))$ is output if x is a solution of the constraint system:*

$$\left\{\begin{array}{l}\mathsf{pub}(\mathit{ska}),\ \mathsf{pub}(\mathit{skb}),\ \mathsf{aenc}(\langle n_a, \mathsf{pub}(\mathit{ska})\rangle, \mathsf{pub}(\mathit{skb}))) \vdash^? x \\ \mathsf{proj}_2(\mathsf{adec}(x, \mathit{skb})) \neq^? \mathsf{pub}(\mathit{ska})\end{array}\right.$$

Hence, though the variable x may take infinitely many values, only two relevant sets of messages have to be considered, that are respectively the solutions of the first and the second constraint systems.

Now, let us consider the trace equivalence problem. Given two processes P and Q, we have to decide whether or not, for every attacker's sequences of actions, the sequences of outputs of P and Q respectively are indistinguishable. Again, since there are infinitely many possible attacker's actions, we split them into sets that are symbolically represented using constraint systems, in such a way that the operations that are performed by, say, the process P are the same for any two solutions of the same constraint system $\mathcal{C}_P$. Assume first that there is a constraint system $\mathcal{C}_Q$ that represents the same set of attacker's actions and for which Q performs the same operations. Then P and Q are trace equivalent if and only if (at each output step) $\mathcal{C}_P$ and $\mathcal{C}_Q$ are *equivalent constraint systems*: $\mathcal{C}_P$ and $\mathcal{C}_Q$ have the same solutions and, for each solution of $\mathcal{C}_P$ the output messages of P are indistinguishable from the output messages of Q. This indistinguishability property on sequences of messages is formalised using *static equivalence.*

Example 3. *Let us come back to the private authentication protocol presented in Example 1. As explained in [21], the privacy property can be formally expressed as the (trace) equivalence of the two processes $B(b, a)$ and $B(b, c)$ that formalise the role B, in which, respectively, b is willing to talk to a and b is willing to talk to c (assuming $a, b,$ and c are honest and their public keys are known by the attacker).*

In the protocol PrivAuth1, *the traces of $B(b, a)$ are represented, as explained in Example 2, by the constraint system $\mathcal{C}_P$:*

$$\mathcal{C}_P = \left\{\begin{array}{l}\mathsf{pub}(\mathit{ska}),\ \mathsf{pub}(\mathit{skb}),\ \mathsf{pub}(\mathit{skc}),\ \mathsf{aenc}(\langle n_a, \mathsf{pub}(\mathit{ska})\rangle, \mathsf{pub}(\mathit{skb}))) \vdash^? x \\ \mathsf{proj}_2(\mathsf{adec}(x, \mathit{skb})) =^? \mathsf{pub}(\mathit{ska})\end{array}\right.$$

For any solution of the constraint, the trace consists of one message

$$\Phi_P = \mathsf{aenc}(\langle \mathsf{proj}_1(\mathsf{adec}(x, \mathit{skb})), \langle n_b, \mathsf{pub}(\mathit{skb})\rangle\rangle, \mathsf{pub}(\mathit{ska})).$$

Otherwise, the trace is empty. The traces of $B(b, c)$ are represented in a similar way by the constraint $\mathcal{C}_Q$:

$$\mathcal{C}_Q = \left\{\begin{array}{l}\mathsf{pub}(\mathit{ska}),\ \mathsf{pub}(\mathit{skb}),\ \mathsf{pub}(\mathit{skc}),\ \mathsf{aenc}(\langle n_a, \mathsf{pub}(\mathit{ska})\rangle, \mathsf{pub}(\mathit{skb}))) \vdash^? x \\ \mathsf{proj}_2(\mathsf{adec}(x, \mathit{skb})) =^? \mathsf{pub}(\mathit{skc})\end{array}\right.$$

For any solution of the constraint, the trace consists of one message

$$\Phi_Q = \mathsf{aenc}(\langle \mathsf{proj}_1(\mathsf{adec}(x, skb)), \langle n_b, \mathsf{pub}(skb)\rangle\rangle, \mathsf{pub}(skc)).$$

Otherwise, the trace is empty. In this particular case, $B(b, a)$ and $B(b, c)$ are trace equivalent if and only if:

1. *the sets of solutions of the two constraint systems are identical (otherwise, there is an attacker input, for which one of the traces is empty and the other is not empty)*
2. *for any solution of either constraint systems, the two output messages are indistinguishable (formally, they are statically equivalent).*

This is what is formalized (in a general setting) by the equivalence of constraint systems.

In this very example, $x = \mathsf{aenc}(\langle \mathsf{pub}(ska), \mathsf{pub}(ska)\rangle, \mathsf{pub}(skb))$ is a solution of $\mathcal{C}_P$ and not of $\mathcal{C}_Q$, thus the sets of solutions do not coincide.

In general, the situation is however more complex since, for two attacker's actions yielding two solutions of $\mathcal{C}_P$, the process Q may move in different ways. This depends in general on additional properties of the attacker's input: the actions of the attacker are split into the solutions of $\mathcal{C}_Q^1, \mathcal{C}_Q^2$.... Now, we need to consider not only the equivalence of constraint systems, but also the equivalence of *sets* of constraint systems.

Example 4. *Consider now the protocol PrivAuth2: the privacy is expressed as the trace equivalence of $B'(b, a)$ and $B'(b, c)$. The traces of $B'(b, a)$ consist in a single message:*

1. *the message $\mathsf{aenc}(\langle \mathsf{proj}_1(\mathsf{adec}(x, skb)), \langle n_b, \mathsf{pub}(skb)\rangle\rangle, \mathsf{pub}(ska))$ if x is a solution of the constraint $\mathcal{C}_1(ska)$ where:*

$$\mathcal{C}_1(\alpha) \stackrel{\mathsf{def}}{=} \left\{ \begin{array}{l} \mathsf{pub}(ska),\ \mathsf{pub}(skb),\ \mathsf{pub}(skc),\ \mathsf{aenc}(\langle n_a, \mathsf{pub}(ska)\rangle, \mathsf{pub}(skb))) \vdash^? x \\ \mathsf{proj}_2(\mathsf{adec}(x, skb)) =^? \mathsf{pub}(\alpha) \end{array} \right.$$

2. *the message $\mathsf{aenc}(\langle n_b, n_b\rangle, \mathsf{pub}(ska)))$ if x is a solution of the constraint $\mathcal{C}_2(ska)$ where:*

$$\mathcal{C}_2(\alpha) \stackrel{\mathsf{def}}{=} \left\{ \begin{array}{l} \mathsf{pub}(ska),\ \mathsf{pub}(skb),\ \mathsf{pub}(skc),\ \mathsf{aenc}(\langle n_a, \mathsf{pub}(ska)\rangle, \mathsf{pub}(skb))) \vdash^? x \\ \mathsf{proj}_2(\mathsf{adec}(x, skb)) \neq^? \mathsf{pub}(\alpha) \end{array} \right.$$

Now, $B'(b, a)$ and $B'(b, c)$ are trace equivalent if, for every x,

1. *$\mathcal{C}_1(ska) \vee \mathcal{C}_2(ska)$ and $\mathcal{C}_1(skc) \vee \mathcal{C}_2(skc)$ have the same solutions (when there is an output on one side, there is also an output on the other side).*
2. *For any solution of either sets of constraint systems, the output messages are statically equivalent.*

In this very example, the equivalence boils down to the static equivalence of the two following (sequences of) messages:

- $\mathsf{aenc}(\langle \mathsf{proj}_1(\mathsf{adec}(x, skb)), \langle n_b, \mathsf{pub}(skb)\rangle\rangle, \mathsf{pub}(ska))$ *(when x is a solution of $\mathcal{C}_1(ska)$),*
- $\mathsf{aenc}(\langle n_b, n_b\rangle, \mathsf{pub}(ska)))$.

This example shows already the use of sets of constraints. Let us also emphasize another important feature of our (sets of) constraint systems. In the context of equivalence problems, the relevant notion of solutions of constraint systems are not the assignments to the free variables (as it is the case in [24, 25, 26] for instance), but the *recipes* used to get such assignments, as illustrated by the following example.

Example 5. *Consider the following two processes:*

$$P = \mathsf{out}(\langle t_1, t_2\rangle).\mathsf{in}(x).\mathsf{if}\ x = t_1\ \mathsf{then}\ \mathsf{out}(s_1)\ \mathsf{else\ if}\ x = t_2\ \mathsf{then}\ \mathsf{out}(s_2)$$
$$Q = \mathsf{out}(\langle t_2, t_1\rangle).\mathsf{in}(x).\mathsf{if}\ x = t_1\ \mathsf{then}\ \mathsf{out}(s_1)\ \mathsf{else\ if}\ x = t_2\ \mathsf{then}\ \mathsf{out}(s_2)$$

where t_1, t_2 are any distinct messages that are statically equivalent, e.g., *two random numbers freshly generated.*

Let $\mathcal{C}_P^1, \mathcal{C}_P^2$ (resp. $\mathcal{C}_Q^1, \mathcal{C}_Q^2$) be the constraint systems associated with the two branches of P (resp. Q). The same assignments to x satisfy respectively $\mathcal{C}_P^1 \vee \mathcal{C}_P^2$ and $\mathcal{C}_Q^1 \vee \mathcal{C}_Q^2$. And for any such assignment, the output messages are identical. Yet, the processes are trace equivalent only if s_1, s_2 are statically equivalent. Indeed, the attacker may either forward (as x) the first or the second projection of the first output message. If he forwards the first projection, he will get s_1 in the first experiment and s_2 in the second experiment. This example shows that the relevant notion of solution *of a constraint system is not the assignment of x, but rather the way x is constructed, which we will call a* recipe.

In summary, each constraint system comes with a *frame*, recording the output messages. For instance, in Example 3, the constraint system $\mathcal{C}_P$ comes with the frame

$$\mathsf{pub}(ska);\ \mathsf{pub}(skb);\ \mathsf{pub}(skc);\ \mathsf{aenc}(\langle n_a, \mathsf{pub}(ska)\rangle, \mathsf{pub}(skb)); \Phi_P$$

and the constraint system $\mathcal{C}_Q$ comes with a frame

$$\mathsf{pub}(ska);\ \mathsf{pub}(skb);\ \mathsf{pub}(skc);\ \mathsf{aenc}(\langle n_a, \mathsf{pub}(ska)\rangle, \mathsf{pub}(skb)); \Phi_Q.$$

Later, we will define formally a more general notion of frame.

Two sets of constraint systems $\mathcal{S}$ and $\mathcal{S}'$ are equivalent if (and only if) for any solution of a constraint $\mathcal{C} \in \mathcal{S}$ and any possible way θ (recipe) to construct this solution, there is a constraint $\mathcal{C}' \in \mathcal{S}'$ such that θ also yields a solution of $\mathcal{C}'$ and the corresponding instance of the frame associated with $\mathcal{C}$ is statically equivalent to the corresponding instance of the frame associated with $\mathcal{C}'$.

In a companion paper [14], we show how the trace equivalence of processes without replication, but that may contain non trivial conditional branching and non deterministic choices, can be effectively reduced to the equivalence of sets of constraint systems. The focus of this paper is on the decision of the equivalence of such sets of constraint systems. Though, we will illustrate our techniques using examples coming from process equivalence problems. In addition, as explained in [27, 28], we will consider constraint systems that do not contain destructors (no projection nor decryption for instance). For the cryptographic primitives that we consider, this is not a restriction, since, using a narrowing technique [29], it is always possible to get rid of them, possibly at the price of introducing new variables.

Example 6. *Consider the constraint $\mathcal{C}_1(\alpha)$ in Example 4. The equality constraint* $\mathsf{proj}_2(\mathsf{adec}(x, skb)) =^? \mathsf{pub}(\alpha)$ *contains destructors. It can however be narrowed to eliminate the destructors, replacing the equation with*

$$x =^? \mathsf{aenc}(\langle x_1, x_2\rangle, \mathsf{pub}(skb)) \ \wedge \ x_2 =^? \mathsf{pub}(\alpha).$$

Overview of our procedure. The general idea of our decision algorithm for the equivalence of (sets of) constraint systems is borrowed from earlier work on deducibility constraints: we simplify the constraints until we get a simple form, on which the equivalence problem should be easy. Since we are interested in equivalence properties, there are two main differences. First, we need to consider pairs of (sets of) constraint systems. The simplification rules should be applied on both (sets of) systems at the same time; when this corresponds to guessing an attacker action, it should be the same rule, which is applied on both (sets of) systems. The second main difference concerns the equivalence checking: we have to keep track of an extended frame, recording some of the deductions of the attacker, and check the static equivalence of all instances, when the constraints are in solved form.

In comparison to previous constraint solving algorithms, there are many additional difficulties, which we will emphasize along the paper. One of the problems is that, when applying the rules in a naive way, the two (sets of) constraint systems do not necessarily reach a solved form at the same time. So, we may need to apply further rules, even when one of the systems is in solved form, which causes termination issues.

Finally, along the algorithm, we guess for instance whether or not a key is deducible. This introduces negative deducibility constraints, which might be hard to solve. We turn around the difficulty, keeping track of previous choices (*e.g.,* whether a key was deducible or not). This yields matrices of constraint systems: the different columns correspond to constraint systems that share the same structure, but may yield different outputs of the protocol, whereas the different rows correspond to different guesses of deducibility along the constraint simplification. This complication in the syntax allows some simplifications in the algorithm, since we may take advantage of the bookkeeping of different rows.

Outline. In this paper, we decide to focus on the algorithm itself, and we only give some hints about the soundness, completeness and termination of our algorithm. The interested reader will find detailed proofs of these results in appendix.

In Section 2, we introduce most of the definitions together with a few examples. The algorithm is explained in Section 3. We start with single constraint systems, before extending the rules to pairs of (sets of) constraint systems, and later matrices of constraint systems. Section 4 is probably the most technical one; it is devoted to the description of the strategy that is used to ensure soundness, completeness, and termination of our transformation rules. We describe our strategy and we illustrate the main difficulties we encountered using several examples. The procedure has been implemented in a tool called APTE, and we provide with a short summary of the experiments in Section 5.

This paper can be seen as an extended and enriched version of a part of [27]. In [27], it was shown that trace equivalence is decidable for a large class of processes. The core of [27] is the design of an algorithm for equivalence of sets of constraint systems. However, due to space limitations, the algorithm is only briefly presented. In this paper, a detailed description is given with many examples for illustration purposes. The strategy described in this paper is not exactly the same as the one presented in [27, 28]. Actually, we manage to simplify the last steps of the strategy.

2. Messages, constraint systems, and symbolic equivalence

In this section, we introduce most of the definitions together with a few examples. In particular, we define formally the problem we are interested in, *i.e.,* symbolic equivalence between sets of constraint systems.

2.1. Messages

To model messages, we consider an infinite set of *names* $\mathcal{N} = \{a, b, \ldots, sk, k, n, m, \ldots\}$, which are used to model atomic data. We consider $\mathcal{X}^1 = \{x, y, \ldots\}$ an infinite set of *first-order variables*, as well as a *signature* $\mathcal{F}$, *i.e.,* a set of *function symbols.* More precisely, we consider $\mathcal{F} = \mathcal{F}_c \uplus \mathcal{F}_d$ where:

$$\begin{array}{l}\mathcal{F}_c = \{\mathsf{senc}/2,\ \mathsf{aenc}/2,\ \mathsf{pub}/1,\ \mathsf{sign}/2,\ \mathsf{vk}/1,\ \langle\,\rangle/2,\ \mathsf{h}/1\}\\ \mathcal{F}_d = \{\mathsf{sdec}/2,\ \mathsf{adec}/2,\ \mathsf{check}/2,\ \mathsf{proj}_1/1,\ \mathsf{proj}_2/1\}.\end{array}$$

These function symbols model signature, pairing, hash function, symmetric and asymmetric encryptions. Symbols in $\mathcal{F}_c$ are *constructors* and those in $\mathcal{F}_d$ are *destructors.*

Terms are defined as names, variables, and function symbols applied to other terms. For any $\mathsf{F} \subseteq \mathcal{F}$, $\mathsf{N} \subseteq \mathcal{N}$ and $\mathsf{V} \subseteq \mathcal{X}^1$, the set of terms built from N and V by applying function symbols in F is denoted by $\mathcal{T}(\mathsf{F}, \mathsf{N} \cup \mathsf{V})$. We denote by $vars^1(u)$ the set of (first-order) variables occurring in a term u. A term u is *ground* if $vars^1(u) = \emptyset$. We denote by $st(u)$ the set of subterms of u. A *constructor term*, resp. *ground constructor term*, is a term belonging to $\mathcal{T}(\mathcal{F}_c, \mathcal{N} \cup \mathcal{X}^1)$, resp. to $\mathcal{T}(\mathcal{F}_c, \mathcal{N})$. A ground constructor term is also called a *message.*

Example 7. *Going back to Example 1,* $m = \mathsf{aenc}(\langle n_a, \mathsf{pub}(ska)\rangle, \mathsf{pub}(skb))$ *is a message and* $t = \mathsf{aenc}(\langle \mathsf{proj}_1(\mathsf{adec}(x, skb)), \langle n_b, \mathsf{pub}(skb)\rangle\rangle, \mathsf{pub}(ska))$ *is a non ground term.*

In order to take into account the properties of our cryptographic primitives, we consider the following term rewriting system.

$$\begin{array}{rcl@{\qquad}rcl@{\qquad}rcl}\mathsf{sdec}(\mathsf{senc}(x,y),y) & \to & x & \mathsf{proj}_1(\langle x,y\rangle) & \to & x & & & \\ \mathsf{adec}(\mathsf{aenc}(x,\mathsf{pub}(y)),y) & \to & x & \mathsf{proj}_2(\langle x,y\rangle) & \to & y & \mathsf{check}(\mathsf{sign}(x,y),\mathsf{vk}(y)) & \to & x\end{array}$$

The rules are standard, for instance, the first column states that the decryption of a ciphertext with the appropriate decryption key gives back the plaintext. Symmetric and asymmetric encryptions are respectively considered in each of the two rules. These rules define a convergent term rewriting system [30], and $t\!\downarrow$ denotes the normal form of t.

Example 8. *Continuing Example 7, and considering an honest execution (the one where the attacker does not interfere) of the protocol described in Example 1, the variable x will be instantiated with the message* $m = \mathsf{aenc}(\langle n_a, \mathsf{pub}(ska)\rangle, \mathsf{pub}(skb))$*, and* $t\{x \mapsto m\}\!\downarrow = \mathsf{aenc}(\langle n_a, \langle n_b, \mathsf{pub}(skb)\rangle\rangle, \mathsf{pub}(ska))$*, which is a message.*

We now consider a set $\mathcal{X}^2 = \{X, Y, \ldots\}$ of *recipe variables* and we write $vars^2(\cdot)$ the function that returns the set of recipe variables occurring in its argument. A *recipe* is a term built on $\mathcal{F}_c, \mathcal{F}_d$, a set of *parameters* $\mathcal{AX} = \{ax_1, \ldots, ax_n, ...\}$, that can be seen as pointers to the hypotheses (or known messages), and variables in $\mathcal{X}^2$. As in the applied pi-calculus, all the function symbols are public, *i.e.,* available to the attacker. Moreover, names are excluded from recipes: names that are known to the attacker must be given explicitly as hypotheses. We denote by Π the set of recipes, *i.e.,* $\Pi = \mathcal{T}(\mathcal{F}, \mathcal{AX} \cup \mathcal{X}^2)$. A *ground recipe* ζ is a recipe that does not contain variables ($vars^2(\zeta) = \emptyset$). We denote by $param(\zeta)$ the set of parameters that occur in ζ. Intuitively, a ground *recipe* records the attacker's computation. It is used as a witness of how a deduction has been performed.

Example 9. *As seen in Example 3, to mount an attack against the simplified version of the private authentication protocol, the attacker can build the message:*

$$\mathsf{aenc}(\langle \mathsf{pub}(ska), \mathsf{pub}(ska)\rangle, \mathsf{pub}(skb)).$$

This is indeed possible using the ground recipe $\mathsf{aenc}(\langle ax_1, ax_1\rangle, ax_2)$ *(assuming that* ax_1 *and* ax_2 *are pointers to* $\mathsf{pub}(ska)$ *and* $\mathsf{pub}(skb)$*).*

2.2. Frames

In [3] (and subsequent papers) a *frame* is used to record the sequence of messages (or terms in a symbolic execution) that have been sent by the participants of the protocol. It can be written, using the formalism that we introduce below, as a sequence

$$\{ax_1, 1 \rhd u_1; \ldots; ax_n, n \rhd u_n\}$$

where ax_i are parameters, $1, ..., n$ are the numbering of successive outcomes (the stages of the execution) and $u_1, \ldots, u_n$ are the corresponding output messages. We extend this notion to record some additional information on attacker's deductions. Typically $(\mathsf{sdec}(X, \zeta), i \rhd u)$ records that, using a decryption with the recipe ζ, on top of a recipe X, allows one to get u (at stage i). After recording this information in the frame, we may rely on this bookkeeping, and no longer consider a decryption on top of X.

With such an extension, members of the frame may look like $\zeta, i \rhd v$ and the same stage may appear several times. However, if $\zeta = ax_j$, then we still have $i = j$. The recipes ζ that are added to the frame will always different from the previous ones, which allows to define a substitution associated with a frame, as we explain below.

Definition 1 (frame)**.** *A* frame Φ *(resp.* a closed frame*) is a sequence of the form* $\{\zeta_1, i_1 \rhd u_1; \ldots; \zeta_n, i_n \rhd u_n\}$ *where:*

- $u_1, \ldots, u_n$ *are constructor terms (resp. ground constructor terms),*
- $i_1, \ldots, i_n$ *are integers, and*
- $\zeta_1, \ldots, \zeta_n$ *are distinct general recipes (resp. ground recipes).*

The domain *of the frame* Φ *is* $\mathrm{dom}(\Phi) = \mathcal{AX} \cap \{\zeta_1, \ldots, \zeta_n\}$. *It must be equal to* $\{ax_1, \ldots, ax_m\}$ *for some* m*, and* m *is called the* size *of* Φ*. Moreover, we assume that for all* $(\zeta, i \rhd u) \in \Phi$, $i \leq m$ *and if* $\zeta = ax_j$ *then* $i = j$.

The indices $i_1, \ldots, i_n$ represent the stages at which a message is known. An attacker could indeed distinguish two processes, simply because some message can be computed earlier in one of the processes than in the other: the stage at which messages are available is a relevant information.

A frame Φ of size m defines a substitution on dom(Φ): if dom(Φ) $= \{ax_1, \ldots, ax_m\}$ and, for $i = 1, \ldots, m$, we have that $(ax_i, i \rhd v_i) \in \Phi$, then we write again Φ the substitution $\{ax_1 \mapsto v_1, \ldots, ax_m \mapsto v_m\}$. We denote by $\mathsf{Init}(\Phi)$ the frame Φ restricted to its domain dom(Φ). A closed frame Φ is *consistent* if, for every $(\zeta, i \rhd u) \in \Phi$, we have that $(\zeta\Phi)\!\downarrow = u$. Lastly, an *initial frame* is a frame of the form $\{ax_1, 1 \rhd u_1; \ldots; ax_m, m \rhd u_m\}$ where $ax_1, \ldots, ax_m \in \mathcal{AX}$, *i.e.* a frame such that $\Phi = \mathsf{Init}(\Phi)$.

Example 10. *Consider the following initial frame:*

$$\Phi = \{ax_1, 1 \rhd \mathsf{aenc}(\langle n_a, \mathsf{pub}(ska)\rangle, \mathsf{pub}(skb)));\ ax_2, 2 \rhd \mathsf{aenc}(n_b, \mathsf{pub}(skb))\}$$

Let $\Phi' = \Phi \uplus \{ax_3, 3 \rhd skb;\ \mathsf{adec}(ax_1, ax_3), 3 \rhd \langle n_a, \mathsf{pub}(ska)\rangle\}$. Φ' *is a closed frame. The intermediate component* $\mathsf{adec}(ax_1, ax_3), 3 \rhd \langle n_a, \mathsf{pub}(ska)\rangle$ *records the deduction of* $\langle n_a, \mathsf{pub}(ska)\rangle$ *using the recipe* $\mathsf{adec}(ax_1, ax_3)$ *at stage* 3.

Actually, we do not need to consider recipes that make unnecessary detours, or yield always junk messages. We introduce therefore a restricted set of recipes Π_r:

$$\Pi_r = \{\xi \in \Pi \mid \forall \mathsf{f} \in \mathcal{F}_d, \forall \xi_1, \ldots, \xi_n \in \Pi, \mathsf{f}(\xi_1, \ldots, \xi_n) \in st(\xi) \Rightarrow \mathsf{root}(\xi_1) \notin \mathcal{F}_c\}.$$

where $\mathsf{root}(u)$ is the root symbol of u.

Example 11. *The recipe* $\mathsf{sdec}(\mathsf{senc}(ax_1, ax_2), ax_2)$ *is not in normal form, and thus not in* Π_r*, whereas* $\mathsf{sdec}(\mathsf{senc}(ax_1, ax_1), ax_2)$*, though in normal form, is not in* Π_r*. Intuitively, this recipe, when applied to a frame, will either not yield a message or yield the message pointed by* ax_1*. In the latter case, there is a simpler recipe consisting in* ax_1 *alone.*

We define below the static equivalence in a way similar to [3]. We make explicit the success (or the failure) of decrypting or checking a signature.

Definition 2 (static equivalence)**.** *Two closed frames* Φ *and* Φ' *are* statically equivalent*, written* $\Phi \sim \Phi'$*, if they have the same size* m *and*

1. *for any ground recipe* $\zeta \in \Pi_r$ *such that* $param(\zeta) \subseteq \{ax_1, \ldots, ax_m\}$*,*
$$\zeta\Phi\!\downarrow \in \mathcal{T}(\mathcal{F}_c, \mathcal{N}) \text{ if, and only if, } \zeta\Phi'\!\downarrow \in \mathcal{T}(\mathcal{F}_c, \mathcal{N})$$
2. *for any ground recipes* $\zeta, \zeta' \in \Pi_r$ *such that* $param(\{\zeta, \zeta'\}) \subseteq \{ax_1, \ldots, ax_m\}$*, and the terms* $\zeta\Phi\!\downarrow$*,* $\zeta'\Phi\!\downarrow$ *are in* $\mathcal{T}(\mathcal{F}_c, \mathcal{N})$*,*
$$\zeta\Phi\!\downarrow = \zeta'\Phi\!\downarrow \text{ if, and only, if } \zeta\Phi'\!\downarrow = \zeta'\Phi'\!\downarrow.$$

We could have stated the definition with arbitrary recipes in Π. The definition would have been equivalent (see Lemma 6.7 in [28]). We chose, without loss of generality, to consider recipes in Π_r only, because it simplifies the following.

Example 12. *From Example 1, we consider the two closed frames:*

- $\Phi_1 = \Phi_0 \uplus \{ax_3, 3 \rhd m;\ ax_4, 4 \rhd \mathsf{aenc}(\langle n_a, \langle n_b, \mathsf{pub}(skb)\rangle\rangle, \mathsf{pub}(ska))\}$*, and*

- $\Phi_2 = \Phi_0 \uplus \{ax_3, 3 \rhd m;\ ax_4, 4 \rhd \mathsf{aenc}(n_b, \mathsf{pub}(ska))\}$

with $\Phi_0 = \{ax_1, 1 \rhd \mathsf{pub}(ska);\ ax_2, 2 \rhd \mathsf{pub}(skb)\}$ *and* $m = \mathsf{aenc}(\langle n_a, \mathsf{pub}(ska)\rangle, \mathsf{pub}(skb))$. *They are statically equivalent. Indeed, for any recipe* $\zeta \in \Pi_r$*, there is no redex in either* $\zeta\Phi_1$ *or* $\zeta\Phi_2$*. Moreover, it is not possible to build any ciphertext present in the frame from its components (since each ciphertext involves at least one fresh nonce which is not available to the attacker).*

Example 13. *Assume* b, c *are names. Consider the two following frames:*

$$\Phi_1 = \{ax_1, 1 \rhd a;\ ax_2, 2 \rhd \mathsf{senc}(b, a)\} \qquad \Phi_2 = \{ax_1, 1 \rhd a;\ ax_2, 2 \rhd \mathsf{senc}(c, a)\}$$

They are statically equivalent. Though the recipe $\zeta = \mathsf{sdec}(ax_1, ax_2)$ *yields a non trivial reduction when applied to* Φ_1 *(resp.* Φ_2*), the results* b *and* c *are indistinguishable. Actually, only trivial equalities can be derived on both sides. Now, if we disclose explicitly* b *(or* c*, or both), as in the frames*

$$\Phi_1' = \Phi_1 \uplus \{ax_3, 3 \rhd b\} \qquad \Phi_2' = \Phi_2 \uplus \{ax_3, 3 \rhd b\}$$

then the frames are not statically equivalent. Choosing the recipes $\zeta = \mathsf{sdec}(ax_1, ax_2)$ *and* $\zeta' = ax_3$*, we have that* $\zeta\Phi_1'{\downarrow} = \zeta'\Phi_1'{\downarrow}(= b)$*, while* $\zeta\Phi_2'{\downarrow} \neq \zeta'\Phi_2'{\downarrow}$*. The attacker may observe an equality on the first frame, which does not hold on the second frame.*

The first condition in the definition of static equivalence is also important: the attacker may observe the success of some operation on one of the frames, while it fails on the other.

Example 14. *Consider the two following frames:*

$$\Phi_1 = \{ax_1, 1 \rhd \mathsf{sign}(a, b); ax_2, 2 \rhd \mathsf{vk}(b)\},\ \textit{and}\ \Phi_2 = \{ax_1, 1 \rhd \mathsf{sign}(a, b); ax_2, 2 \rhd \mathsf{vk}(c)\}.$$

The attacker can only observe trivial equalities on both frames. However, if we let $\zeta = \mathsf{check}(ax_1, ax_2)$*, then* $\zeta\Phi_1{\downarrow} \in \mathcal{T}(\mathcal{F}_c, \mathcal{N})$ *and* $\zeta\Phi_2{\downarrow} \notin \mathcal{T}(\mathcal{F}_c, \mathcal{N})$*. The attacker observes the success of checking the signature in one case and its failure in the other case.*

2.3. Constraint systems

As explained in the introduction, our decision algorithm will rely on deducibility constraints, as a means to represent symbolically sets of traces of a protocol. The following definitions are consistent with [25]. In particular, the so-called monotonicity and origination properties are expressed through item 3 in the next definition. Since we are interested here in equivalence properties, we do not only need to represent sets of traces, but also to record some information on the attacker's actions that led to these traces. That is why we also include equations between recipes and a set NoUse of obsolete elements in the frame; roughly, a component of the frame is obsolete when the attacker used another recipe to get the message, at an earlier stage. Finally, we also consider negative constraints, in order to enable splitting the set of traces into disjoint sets.

Definition 3. *A* constraint system *is either* $\bot$ *or a tuple* $(S_1; S_2; \Phi; D; E; E_\Pi; ND; \mathsf{NoUse})$ *where:*

1. S_1 *(resp.* S_2*) is a set of variables in* $\mathcal{X}^1$ *(resp.* $\mathcal{X}^2$*);*
2. Φ *is a frame, whose size is some* m *and* NoUse *is a subset of* Φ*;*

3. D is a sequence $X_1, i_1 \vdash^? t_1;\ \ldots;\ X_n, i_n \vdash^? t_n$ where

- *$X_1, \ldots, X_n$ are distinct variables in $\mathcal{X}^2$*
- *$t_1, \ldots, t_n$ are constructor terms*
- *$1 \leq i_1 \leq \ldots \leq i_n \leq m$.*
- *for every $(\xi, i \rhd t) \in \Phi$, $vars^1(t) \subseteq \bigcup_{i_j < i} vars^1(t_j)$;*
- *for every $(\xi, i \rhd t) \in \Phi$, $param(\xi) \subseteq \{ax_1, \ldots, ax_i\}$ and $vars^2(\xi) \subseteq \{X_k \mid i_k \leq i\}$.*

4. $E = \bigwedge_k u_k =^? v_k \wedge \bigwedge_i \forall \tilde{x_i} \cdot [\bigvee_j u_{i,j} \neq^? v_{i,j}]$ where $u_k, v_k, u_{i,j}$ and $v_{i,j}$ are constructor terms.

5. $E_\Pi = \bigwedge_i \zeta_i =^? \zeta'_i \wedge \bigwedge_j \xi_j \neq^? \xi'_j \wedge \bigwedge_k \mathsf{root}(\beta_k) \neq^? \mathsf{f}_k$ where ζ_i, ζ'_i, ξ_j, ξ'_j, β_k are recipes in Π_r and f_k are constructor symbols.

6. $ND = \bigwedge_i \forall \tilde{x}_i.[u_i \neq^? v_i \vee \bigvee_j k_{i,j} \not\vdash^? w_{i,j}]$ where $u_i, v_i, w_{i,j}$ are constructor terms and $k_{i,j} \in \mathbb{N}$.

We say that a constraint system is initial *if* $\mathsf{NoUse} = \emptyset$*,* $ND = \emptyset$*,* $E_\Pi = \emptyset$*,* $vars^2(D) = S_2$ *and* Φ *is an initial frame.*

Intuitively, S_1 is the set of free variables in $\mathcal{X}^1$; we may have to introduce auxiliary variables, that will be (implicitly) existentially quantified, as well as (explicitly) universally quantified variables. Similarly, S_2 is a set of main recipe variables (in $\mathcal{X}^2$) of the constraint. For readability, we will sometimes omit some of the components of the constraint system, because they are either straightforward from the context or empty. We also write $\Phi_{\mathcal{C}}$ for the frame part of a constraint system $\mathcal{C}$. We will later call the component D the *deducibility part* of the constraint system.

Example 15. *The constraints, as displayed in Example 2, follow another (simpler) syntax. However, as already explained, we need not only to reason about the attacker's inputs, but also on* how *he computed these values. Furthermore, as we explained in the introduction, the terms are assumed to be narrowed, so as to eliminate the destructors; in the Example 2, the variable x has been narrowed to $\mathsf{aenc}(\langle x, y\rangle, \mathsf{pub}(skb))$: for each occurrence $d(s_1, \ldots, s_n)$ of a destructor d, we unify $d(s_1, \ldots, s_n)$ with a left hand side of a rewrite rule $d(v_1, \ldots, v_n) \to w$, then replace $d(s_1, \ldots, s_n)$ with w and add the equations $s_i =^? v_i$.*

According to the syntax of the above definition, the first constraint system of that example should be written:

$$\begin{array}{rl} \Phi = & \{ax_1, 1 \rhd \mathsf{pub}(ska);\ ax_2, 2 \rhd \mathsf{pub}(skb);\ ax_3, 3 \rhd \mathsf{aenc}(\langle n_a, \mathsf{pub}(ska)\rangle, \mathsf{pub}(skb)))\} \\ D = & \{X, 3 \vdash^? \mathsf{aenc}(\langle x, y\rangle, \mathsf{pub}(skb))\} \\ E = & \{y =^? \mathsf{pub}(ska)\} \end{array}$$

Implicitly $S_1 = \{x, y\}$, $S_2 = \{X\}$, and the set E_Π, NoUse *and ND are empty. This is an initial constraint system.*

More examples will be given later. From now, $vars^1(\mathcal{C})$ will denote the set of free first-order variables of $\mathcal{C}$ (while it was, according to the Section 2.1 the set of all variables occurring in $\mathcal{C}$).

Before defining the semantics of such extended constraint systems, we need first to consider the components ND, E, and E_Π, and also to introduce the notion of *path* (see Definition 5). The semantics of ND, E and E_Π is obtained from the interpretation of atomic formulas, using the usual interpretation of logical connectives. Hence we focus on the semantics of atomic formulas in the next definition.

Definition 4 (solution of side constraints)**.** *Let θ be a substitution mapping $vars^2(\mathcal{C})$ to ground recipes, and σ be a substitution mapping $vars^1(\mathcal{C})$ to ground constructor terms.*

1. *$\sigma \vDash (i \nvdash^? u)$ if, and only if, there is no ground recipe $\xi \in \Pi_r$, with $param(\xi) \subseteq \{ax_1, \dots, ax_i\}$, such that $\xi(\Phi\sigma){\downarrow} = u\sigma{\downarrow}$.*
2. *$\sigma \vDash u \neq^? v$ if, and only if, $u\sigma \neq v\sigma$.*
3. *$\theta \vDash \xi_1 =^? \xi_2$ (resp. $\theta \vDash \xi_1 \neq^? \xi_2$) if, and only if, $\xi_1\theta = \xi_2\theta$ (resp. $\xi_1\theta \neq \xi_2\theta$).*
4. *$\theta \vDash \mathsf{root}(\xi) \neq^? \mathsf{f}$ if, and only if, $\mathsf{root}(\xi\theta) \neq \mathsf{f}$.*

Note that, in items 2 and 3, we only check that the equalities and disequalities hold syntactically. Actually, some additional information about systems obtained along our procedure allow us to ensure that resulting terms are in normal form (and thus rewriting is not needed here).

Example 16. *Let $\Phi = \{ax_1, 1 \rhd \mathsf{senc}(a, x);\ ax_2, 2 \rhd b\}$ and $\sigma = \{x \mapsto b\}$. We have that $\sigma \models (1 \nvdash^? a)$ whereas $\sigma \not\models (2 \nvdash^? a)$ since $\mathsf{sdec}(ax_1, ax_2)(\Phi\sigma){\downarrow} = a$.*

There are possibly several ways to compute the same message, given a frame. All possible ways of computing a given message are the observable equalities that are used in the static equivalence. Checking static equivalence will be part of the procedure for the decision of symbolic trace equivalence. Therefore, we may consider only one way (a "canonical" recipe) to get a message from a frame. We choose our recipe according to its *path*, which is the sequence of destructors applied on its leftmost argument. This sequence determines the result, regardless of other arguments. Let us make this point more precise.

Definition 5 (path)**.** *Let $\xi \in \Pi_r$ be such that $\mathsf{root}(\xi) \notin \mathcal{F}_c$. The* path *of ξ, denoted $\mathsf{path}(\xi)$, is a word in the language defined by the regular expression $\mathcal{F}_d^* \cdot (\mathcal{AX} + \mathcal{X}^2)$ over an infinite alphabet. In other words, it is recursively defined as follows:*

$$\mathsf{path}(\xi) = \xi \text{ when } \xi \in \mathcal{AX} \cup \mathcal{X}^2, \text{ and } \mathsf{path}(\mathsf{f}(\xi_1, \dots, \xi_n)) = \mathsf{f} \cdot \mathsf{path}(\xi_1) \text{ otherwise.}$$

Note that, if $\mathsf{f}(\xi_1, \dots, \xi_n) \in \Pi_r$ and $\mathsf{f} \notin \mathcal{F}_c$, then $\mathsf{root}(\xi_1)$ is a destructor, by definition of Π_r. Hence, if $\mathsf{root}(\xi) \notin \mathcal{F}_c$, $\mathsf{path}(\xi)$ is a sequence of destructors, followed by an element of $\mathcal{AX} + \mathcal{X}^2$.

Example 17. *Let $\xi = \mathsf{sdec}(\mathsf{sdec}(ax_2, ax_1), \mathsf{sdec}(ax_1, ax_2))$. We have that $\mathsf{path}(\xi) = \mathsf{sdec}{\cdot}\mathsf{sdec}{\cdot}ax_2$. Assuming that the computation will lead to a message, this path determines the result of the computation.*

NoUse is a subset of the frame whose use is forbidden, because we changed the canonical recipe. This happens only in the course of our algorithm when we discover that a message can actually be computed at an earlier stage. The following defines the restrictions on the recipes that we consider.

Definition 6 (ξ conforms to Φ)**.** *Let Φ be a closed frame,* NoUse *be a subset of Φ, and ξ be a ground recipe in Π_r. We say that ξ* conforms to *the frame Φ w.r.t.* NoUse *if :*

- $\forall \zeta \in st(\xi), \forall (\zeta', i \rhd u) \in \Phi, \mathsf{path}(\zeta) = \mathsf{path}(\zeta') \Rightarrow \zeta = \zeta'$.
- $\forall (\zeta, i \rhd u) \in \mathsf{NoUse}$, $\zeta \notin st(\xi)$

Example 18. *Consider the following frame Φ:*

$$\{ax_1, 1 \rhd \langle a, b\rangle;\ ax_2, 2 \rhd \mathsf{senc}(a, b);\ ax_3, 3 \rhd b;\ ax_4, 4 \rhd a;\ ax_5, 5 \rhd \mathsf{senc}(c, a)\}$$

At some point (stage 2), we may choose a canonical way of computing a, for instance decrypting the second message with the second component of the first one. Then we record this choice in the frame $\Phi^+ = \Phi \cup \{\mathsf{sdec}(ax_2, \mathsf{proj}_2(ax_1)), 2 \rhd a\}$ as well as this commitment to the recipe used to get a: $\mathsf{NoUse} = \{ax_4, 4 \rhd a\}$*.*

The recipe $\mathsf{sdec}(ax_2, ax_3)$*, which yields a, does not conform to Φ^+ w.r.t.* NoUse *because of the first condition in the Definition 6. The recipe* $\mathsf{sdec}(ax_5, ax_4)$ *(that yields c) does not conform to Φ^+ w.r.t.* NoUse *because of the second condition. However, the recipes* $\mathsf{sdec}(ax_2, \mathsf{proj}_2(ax_1))$, $\mathsf{proj}_1(ax_1)$*, and* $\mathsf{sdec}(ax_5, \mathsf{sdec}(ax_2, \mathsf{proj}_2(ax_1)))$ *are conform to Φ^+ w.r.t.* NoUse.

Definition 7 (solution)**.** *A* solution *of $\mathcal{C} = (S_1; S_2; \Phi; D; E; E_\Pi; ND; \mathsf{NoUse})$ consists of a substitution σ mapping $vars^1(\mathcal{C})$ to ground constructor terms and a substitution θ mapping $vars^2(\mathcal{C})$ to ground recipes in Π_r, such that:*

1. *for every $X \in vars^2(\mathcal{C})$, $X\theta$ conforms to $\Phi\theta\sigma$ w.r.t.* $\mathsf{NoUse}\theta$*;*
2. *for every $X, j \vdash^? u$ in D, $X\theta(\Phi\sigma){\downarrow} = u\sigma{\downarrow}$ and $param(X\theta) \subseteq \{ax_1, \dots, ax_j\}$;*
3. *$\sigma \models ND \wedge E$ and $\theta \models E_\Pi$.*

We denote by $\mathsf{Sol}(\mathcal{C})$ *the set of solutions of $\mathcal{C}$. By convention,* $\mathsf{Sol}(\perp) = \emptyset$*. A pair (σ, θ) that only satisfies the two first items is a pre-solution of $\mathcal{C}$.*

Example 19. *Consider the constraint of Example 15. $\sigma = \{x \mapsto n_a, y \mapsto \mathsf{pub}(ska)\}$, $\theta = \{X \mapsto ax_3\}$ is the obvious solution of the constraint. Another solution is the pair (σ', θ') with $\sigma' = \{x \mapsto \mathsf{pub}(ska), y \mapsto \mathsf{pub}(ska)\}$ and $\theta' = \{X \mapsto \mathsf{aenc}(\langle ax_1, ax_1\rangle, ax_2)\}$.*

Example 20. *Consider the frame Φ^+ of Example 18, together with*

$$X, 5 \vdash^? \mathsf{senc}(x, a);\ \ x \neq^? a \wedge x \neq^? b \wedge \forall y_1, y_2.(x \neq^? \langle y_1, y_2\rangle \wedge x \neq^? \mathsf{senc}(y_1, y_2)).$$

One possible solution is $\sigma = \{x \mapsto c\}$ with $\theta = \{X \mapsto ax_5\}$.

2.4. Sets of constraint systems

Before moving to the equivalence of constraint systems, we need to consider sets of constraint systems, as explained in the introduction. We do not have however to consider arbitrary sets of constraint systems, but only constraint systems that have the same *structure*. Roughly, two systems have the same structure if they correspond to the same attacker's actions, but do not necessarily correspond to the same frame nor the same side constraints. As shown in Example 4, we needed to move from constraint systems to sets of constraint systems, because of non-deterministic choices and non-trivial conditional branchings: for the same attacker's recipes, several outcomes are possible. In such a case, the different constraint systems share the same structure, as defined below.

Definition 8 (structure). *Let $\mathcal{C} = (S_1; S_2; \Phi; D; E; E_\Pi; ND; \mathsf{NoUse})$ be a constraint system. The* structure *of $\mathcal{C}$ is given by the following sets:*

$$S_2,\ E_\Pi,\ \{(X,i) \mid X, i \vdash^? u \in D\},\ \{(\xi,i) \mid \xi, i \rhd u \in \Phi\} \text{ and } \{(\xi,i) \mid \xi, i \rhd u \in \mathsf{NoUse}\}.$$

Two constraint systems $\mathcal{C}$ and $\mathcal{C}'$ have the same structure *if their underlying structures are identical. By convention, the constraint system $\bot$ has the same structure as any other constraint system.*

Definition 9. Sets of constraint systems *are sequences of constraint systems sharing the same structure.*

Example 21. *Back to Example 4, the initial set of constraint systems is given by the pair $[\mathcal{C}_1(ska), \mathcal{C}_2(ska)]$:*

$$\mathcal{C}_1(\alpha) = \begin{cases} \Phi_1 = & \Phi_0 \uplus \{ax_4, 4 \rhd m;\ ax_5, 5 \rhd \mathsf{aenc}(\langle x, \langle n_b, \mathsf{pub}(skb)\rangle\rangle, \mathsf{pub}(ska))\} \\ D_1 = & \{X, 4 \vdash^? \mathsf{aenc}(\langle x, y\rangle, \mathsf{pub}(skb))\} \\ E_1 = & \{y =^? \mathsf{pub}(\alpha)\} \end{cases}$$

$$\mathcal{C}_2(\alpha) = \begin{cases} \Phi_2 = & \Phi_0 \uplus \{ax_4, 4 \rhd m;\ ax_5, 5 \rhd \mathsf{aenc}(n_b, \mathsf{pub}(skb))\} \\ D_2 = & \{X, 4 \vdash^? \mathsf{aenc}(\langle x, y\rangle, \mathsf{pub}(skb))\} \\ E_2 = & \{y \neq^? \mathsf{pub}(\alpha)\} \end{cases}$$

where $\Phi_0 = \{ax_1, 1 \rhd \mathsf{pub}(ska);\ ax_2, 2 \rhd \mathsf{pub}(skb);\ ax_3, 3 \rhd \mathsf{pub}(skc)\}$, and $m = \mathsf{aenc}(\langle n_a, \mathsf{pub}(ska)\rangle, \mathsf{pub}(skb))$. Both constraint systems are initial constraint systems and they have the same structure.

2.5. Symbolic equivalence

We come finally to the symbolic equivalence, the property that we want to decide.

Definition 10 (symbolic equivalence $\approx_s$). *Let $\mathcal{S}$ and $\mathcal{S}'$ be two sets of constraint systems. $\mathcal{S} \subseteq_s \mathcal{S}'$ if, for every $\mathcal{C} \in \mathcal{S}$, for every $(\sigma, \theta) \in \mathsf{Sol}(\mathcal{C})$, there exists $\mathcal{C}' \in \mathcal{S}'$ and a substitution σ' such that $(\sigma', \theta) \in \mathsf{Sol}(\mathcal{C}')$ and $\Phi_{\mathcal{C}}\sigma \sim \Phi_{\mathcal{C}'}\sigma'$.*

If $\mathcal{S} \subseteq_s \mathcal{S}'$ and $\mathcal{S}' \subseteq_s \mathcal{S}$, then we say that $\mathcal{S}$ and $\mathcal{S}'$ are in symbolic equivalence*, which we write $\mathcal{S} \approx_s \mathcal{S}'$.*

Example 22. *Using the notations of Example 21, the two sets of constraint systems $[\mathcal{C}_1(ska), \mathcal{C}_2(ska)]$ and $[\mathcal{C}_1(skc), \mathcal{C}_2(skc)]$ are symbolically equivalent (this is a non-trivial equivalence).*

The decision of symbolic equivalence between sets of constraint systems (the problem that is solved in this paper) is exactly the crucial piece for the decision of privacy-type security properties:

Theorem 1 ([14, 28]). *If symbolic equivalence between sets of constraint systems is decidable, then trace equivalence between processes with non determinism and conditional branching (but without replication) is decidable.*

3. Our algorithm

As explained in the introduction, our algorithm which decides symbolic equivalence between sets of constraint systems is based on transformations of such systems until a solved formed is reached. We start by defining and explaining these rules on a single constraint system and then explain how it is extended to pairs of sets (and actually even matrices) of constraint systems.

3.1. The transformation rules

The transformation rules are split in two parts. They are displayed in Figure 1 and Figure 2 respectively, and we start by explaining the rules on a single constraint system. For sake of readability, we only write the components of the constraint system that are modified by an application of an instance of a rule. Moreover, in all the following examples, we apply eagerly some simplifications (such simplifications are formalised and explained in Section 3.2).

The rules of Figure 1 aim at simplifying the deducibility constraints, until they only involve variables. In some respect, this amounts to enumerate (relevant) recipes. The only subtle point is that such recipes are constructed both from bottom (DEST) and from top (CONS, AXIOM). Therefore, they may either yield instantiations of the recipe variables (construction from the top) or new elements in the frame (postpone the instantiation). These rules are necessary, but not sufficient, for the decision of equivalence properties, because of possible observable identities that hold on a constraint system and not on the other. The obvious case is an equality between two members of the frame, but there are more subtle examples. The complementary rules are described in Figure 2 and will be commented further.

Transformation rules for satisfiability (Figure 1). A simple idea would be to guess the top function symbol of a recipe and replace the recipe variable with the corresponding instance. When the head symbol of a recipe is a constructor and the corresponding term is not a variable, we can simplify the constraint, breaking it into pieces. This is the purpose of the rule CONS. CONS(X, f) rule simply makes a case distinction on whether the top symbol of the recipe variable X is a constructor f. Either it is, and then we can decompose the constraint, or it is not and we add a disequation of the form $\mathsf{root}(X) \neq^? \mathsf{f}$ forbidding X to start with f.

Example 23. *Consider the constraint $\mathcal{C}_1(ska)$ of Example 21:*

$$X, 4 \vdash^? \mathsf{aenc}(\langle x, \mathsf{pub}(ska)\rangle, \mathsf{pub}(skb))$$

CONS(X, aenc) *can be applied to the constraint, guessing whether or not the attacker computed the term $t = \mathsf{aenc}(\langle x, \mathsf{pub}(ska)\rangle, \mathsf{pub}(skb))$ by applying an asymmetric encryption on two previously computed messages. This yields the two constraint systems:*

$$\mathcal{C}_{11} := \left\{ \begin{array}{l} X_1, 4 \vdash^? x_1;\ X_2, 4 \vdash^? x_2 \\ t =^? \mathsf{aenc}(x_1, x_2) \ \wedge\ y =^? \mathsf{pub}(ska) \\ X =^? \mathsf{aenc}(X_1, X_2) \end{array} \right. \qquad \mathcal{C}_{12} := \left\{ \begin{array}{l} X, 4 \vdash^? t \\ y =^? \mathsf{pub}(ska) \\ \mathsf{root}(X) \neq^? \mathsf{aenc} \end{array} \right.$$

The first constraint system can be simplified, solving equations and performing replacements, which yields:

$$X_1, 4 \vdash^? \langle x, \mathsf{pub}(ska)\rangle;\ X_2, 4 \vdash^? \mathsf{pub}(skb); \quad y =^? \mathsf{pub}(ska); \quad X =^? \mathsf{aenc}(X_1, X_2)$$

$$\begin{tikzcd} & \begin{array}{l} S'_2;\ X_1, i \vdash^? x_1; \cdots; X_n, i \vdash^? x_n; \\ E \wedge t =^? \mathsf{f}(x_1, \ldots, x_n); \\ E_\Pi \wedge X =^? \mathsf{f}(X_1, \ldots, X_n) \end{array} \\ \underline{\textsc{Cons}(X, \mathsf{f})}:\ S_2; X, i \vdash^? t; E; E_\Pi \arrow[ur] \arrow[dr] & \\ & X, i \vdash^? t\ ;\ E;\ E_\Pi \wedge \mathsf{root}(X) \neq^? \mathsf{f} \end{tikzcd}$$

where: • $x_1, \ldots, x_n, X_1, \ldots X_n$ are fresh variables, and
• $S'_2 = S_2 \cup \{X_1, \ldots, X_n\}$ if $X \in S_2$ and $S'_2 = S_2$ otherwise.

$$\begin{tikzcd} & \Phi;\ E \wedge u =^? v;\ E_\Pi \wedge X =^? \xi \\ \underline{\textsc{Axiom}(X, \mathsf{p})}:\ \Phi;\ X, i \vdash^? u;\ E;\ E_\Pi \arrow[ur] \arrow[dr] & \\ & \Phi;\ X, i \vdash^? u;\ E;\ E_\Pi \wedge X \neq^? \xi \end{tikzcd}$$

If Φ contains $\xi, j \rhd v$ with $i \geq j$, $\mathsf{path}(\xi) = \mathsf{p}$ and $(\xi, j \rhd v) \notin \mathsf{NoUse}$.

$$\begin{tikzcd} & \begin{array}{l} \Phi,\ \mathsf{f}(\xi, X_2, \ldots, X_n), i \rhd w;\ E \wedge v =^? u_1 \\ X_2, i \vdash^? u_2;\ \ldots X_n, i \vdash^? u_n; \mathit{ND} \end{array} \\ \underline{\textsc{Dest}(\xi, l \to r, i)}:\ \Phi; E; \mathit{ND} \arrow[ur] \arrow[dr] & \\ & \begin{array}{l} \Phi;\ E; \\ \mathit{ND} \wedge \forall \tilde{x} \cdot [v \neq u_1 \vee i \nvdash^? u_2 \vee \ldots \vee i \nvdash^? u_n] \end{array} \end{tikzcd}$$

If Φ contains $\xi, j \rhd v$ with $j \leq i$ and $(\xi, j \rhd v) \notin \mathsf{NoUse}$. We denote by $\tilde{x}$ the set of variables that occur in $\mathsf{f}(u_1, \ldots, u_n) \to w$, a fresh renaming of $l \to r$. $X_2, \ldots, X_n$ are fresh variables.

Figure 1: Transformation rules for satisfiability

The rule Axiom also makes a case distinction on whether a trivial recipe (a left member of the frame, typically an axiom ax_i) can be applied. If so, the constraint can simply be removed. Otherwise, we also add a disequation between recipes forbidding it.

Example 24. *Continuing with the two constraints (respectively named $\mathcal{C}_{11}$ and $\mathcal{C}_{12}$), obtained in the previous example, $\mathcal{C}_{11}$ yields, by application of* Axiom(X_2, ax_2),

$$\mathcal{C}_{111} := \left\{ \begin{array}{l} X_1, 4 \vdash^? \langle x, \mathsf{pub}(\mathit{ska}) \rangle \\ \mathsf{pub}(\mathit{skb}) =^? \mathsf{pub}(\mathit{skb}) \\ X =^? \mathsf{aenc}(X_1, X_2) \wedge X_2 =^? ax_2 \end{array} \right. \qquad \mathcal{C}_{112} := \left\{ \begin{array}{l} X_1, 4 \vdash^? \langle x, \mathsf{pub}(\mathit{ska}) \rangle \\ X_2, 4 \vdash^? \mathsf{pub}(\mathit{skb}) \\ X =^? \mathsf{aenc}(X_1, X_2) \wedge X_2 \neq^? ax_2 \end{array} \right.$$

Again, $\mathcal{C}_{111}$ can be simplified as follows:

$$X_1, 4 \vdash^? \langle x, \mathsf{pub}(\mathit{ska}) \rangle; \qquad X =^? \mathsf{aenc}(X_1, ax_2)\ \wedge\ X_2 =^? ax_2$$

Trying to apply Axiom *to other deducibility constraints of $\mathcal{C}_{11}$ or using other members of the frame yields a failure of unification (we get $X_1 \neq^? ax_1, X_2 \neq^? ax_1, \ldots$). When applied to the constraint $\mathcal{C}_{12}$, the only case where the two branches are non-trivial is the* *application of* $\textsc{Axiom}(X, ax_4)$*:*

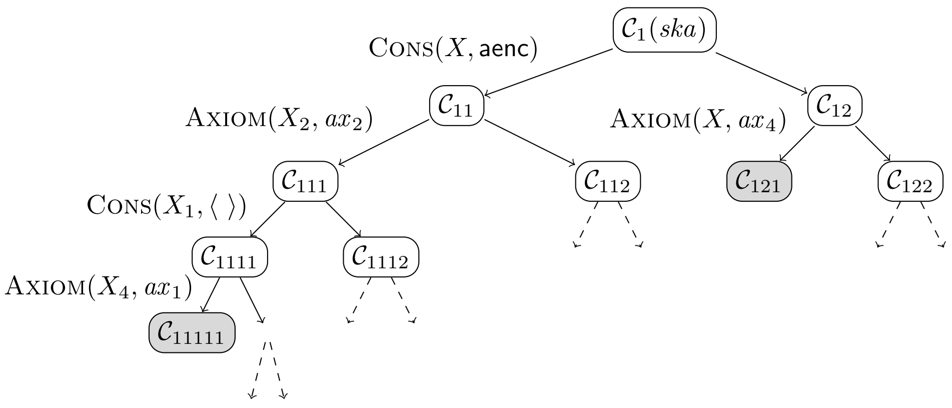
$$\mathcal{C}_{121} := \left\{\begin{array}{l} \mathsf{aenc}(\langle x, \mathsf{pub}(ska)\rangle, \mathsf{pub}(skb)) =^? \mathsf{aenc}(\langle n_a, \mathsf{pub}(ska)\rangle, \mathsf{pub}(skb)) \\ X =^? ax_4 \wedge \mathsf{root}(X) \neq^? \mathsf{aenc} \end{array}\right.$$

$$\mathcal{C}_{122} := \left\{\begin{array}{l} X, 4 \vdash^? \mathsf{aenc}(\langle x, \mathsf{pub}(ska)\rangle, \mathsf{pub}(skb)) \\ X \neq^? ax_4 \ \wedge\ \mathsf{root}(X) \neq^? \mathsf{aenc} \end{array}\right.$$

This can be simplified as follows:

$$\mathcal{C}_{121} = \left\{\begin{array}{l} x =^? n_a \wedge y =^? \mathsf{pub}(ska) \\ X =^? ax_4 \end{array}\right. \qquad \mathcal{C}_{122} = \left\{\begin{array}{l} X, 4 \vdash^? \mathsf{aenc}(\langle x, \mathsf{pub}(ska)\rangle, \mathsf{pub}(skb)) \\ X \neq^? ax_4 \ \wedge\ \mathsf{root}(X) \neq^? \mathsf{aenc} \end{array}\right.$$

An overview of our procedure applied to the constraint system $\mathcal{C}_1(ska)$ is given below:



The constraint systems $\mathcal{C}_{11}$ and $\mathcal{C}_{12}$ are those described in Example 23 whereas the constraint systems $\mathcal{C}_{111}$, $\mathcal{C}_{112}$, $\mathcal{C}_{121}$ and $\mathcal{C}_{122}$ are given in Example 24. The system $\mathcal{C}_{121}$ is actually in solved form (no more rule can be applied) and it admits a solution:

$$\sigma = \{x \mapsto n_a, y \mapsto \mathsf{pub}(ska)\} \text{ with } \theta = \{X \mapsto ax_4\}.$$

The dashed arrows indicate that even if some rules can still be applied, they would lead to constraint systems with no solution (*i.e.,* $\perp$). The constraint system $\mathcal{C}_{112}$ is not yet in solved form but $\mathcal{C}_{112}$ can not be satisfied because the only way to deduce $\mathsf{pub}(skb)$ is to use ax_2 which is forbidden by $X \neq^? ax_2$. Regarding the constraint system $\mathcal{C}_{122}$ the two possible ways to deduce a term of the form $\mathsf{aenc}(\langle x, \mathsf{pub}(ska)\rangle, \mathsf{pub}(skb))$ is to build it using aenc (but this is forbidden by the constraint $\mathsf{root}(X) \neq^? \mathsf{aenc}$) or to use ax_4 (also forbidden due to $X \neq^? ax_4$).

Regarding the left branch of the tree, we can apply the rule $\textsc{Cons}(X_1, \langle\ \rangle)$ on $\mathcal{C}_{111}$ to obtain the constraint systems $\mathcal{C}_{1111}$ and $\mathcal{C}_{1112}$ (which has actually no solution):

$$\mathcal{C}_{1111} := \left\{\begin{array}{l} X_3, 4 \vdash^? x;\ X_4, 4 \vdash^? \mathsf{pub}(ska); \\ X =^? \mathsf{aenc}(\langle X_3, X_4\rangle, ax_2) \\ X_2 =^? ax_2 \wedge X_1 =^? \langle X_3, X_4\rangle \end{array}\right. \qquad \mathcal{C}_{1112} := \left\{\begin{array}{l} X_1, 4 \vdash^? \langle x, \mathsf{pub}(ska)\rangle; \\ X =^? \mathsf{aenc}(X_1, ax_2) \\ X_2 =^? ax_2 \wedge \mathsf{root}(X_1) \neq^? \langle\ \rangle \end{array}\right.$$

Lastly, we may apply the rule $\textsc{Axiom}(X_4, ax_1)$ on $\mathcal{C}_{1111}$ and obtain on the left branch the (solved) constraint system $\mathcal{C}_{11111}$:

$$X_3, 4 \vdash^? x; \quad X =^? \mathsf{aenc}(\langle X_3, ax_1\rangle, ax_2) \wedge X_2 =^? ax_2 \wedge X_1 =^? \langle X_3, ax_1\rangle \wedge X_4 =^? ax_1$$

This system has several solutions among which $\sigma = \{x \mapsto \mathsf{pub}(ska),\ y \mapsto \mathsf{pub}(ska)\}$ obtained by mapping X_3 to ax_1. This means that the recipe $X = \mathsf{aenc}(\langle ax_1, ax_1\rangle, ax_2)$ can be used to build a message that will satisfy all the requirements: the message $\mathsf{aenc}(\langle \mathsf{pub}(ska), \mathsf{pub}(ska)\rangle, \mathsf{pub}(skb))$ is indeed of the expected form, *i.e.,* of the form $\mathsf{aenc}(\langle x, \mathsf{pub}(ska)\rangle, \mathsf{pub}(skb))$.

Now, when the top symbol of a recipe is a destructor, we can not apply the same transformation since the resulting constraint systems will become more complex, introducing new terms, which yields non-termination. Thus, our strategy is different. We switch from the top position of the recipe to the *redex position* using the rule $\textsc{Dest}$. If v is term of the frame, that can be unified with a non variable subterm of a left-hand side of a rewrite rule (for instance v is a ciphertext), we guess whether the rule can be applied to v. This corresponds to the equation $v =^? u_1$ occurring in the $\textsc{Dest}$ rule, that yields an instance of w, the right member of the rewrite rule, provided that the other left members are also deducible. Typically, in case of symmetric encryption, if a ciphertext is in the frame, we will give a direct access to the plaintext by adding a new element in the frame. For this, we have to ensure that the decryption key is deducible. The index i corresponds to the stage at which we try to deduce the key. Note that a key that is not deducible at stage i may become deducible later on, *i.e.,* at stage $j > i$. Thus, we may need to apply this rule several times on the same frame element (at different stages).

Example 25. *Consider the constraint system, that includes the frame*

$$\Phi = \{ax_1, 1 \rhd \mathsf{senc}(\langle a, b\rangle, c);\ ax_2, 2 \rhd c;\ ax_3, 3 \rhd \mathsf{senc}(c, a)\}$$

and the constraint $X, 3 \vdash^? b$. *Applying* $\textsc{Dest}(ax_1, \mathsf{sdec}(\mathsf{senc}(x, y), y) \to x, 2)$*, we get:*

$$\left\{\begin{array}{l} \Phi,\ \mathsf{sdec}(ax_1, X_2), 2 \rhd x \\ X_2, 2 \vdash^? y;\ X, 3 \vdash^? b \\ \mathsf{senc}(x, y) =^? \mathsf{senc}(\langle a, b\rangle, c) \end{array}\right. \qquad \left\{\begin{array}{l} \Phi \\ X, 3 \vdash^? b \\ \forall x, y.(\mathsf{senc}(x, y) \neq \mathsf{senc}(\langle a, b\rangle, c) \vee 2 \not\vdash^? y) \end{array}\right.$$

Basically, we guess here whether the key c can be deduced at stage 2.

The second constraint is unsatisfiable, while the first one can be simplified to:

$$\Phi,\ \mathsf{sdec}(ax_1, X_2), 2 \rhd \langle a, b\rangle \qquad X_2, 2 \vdash^? c;\ X, 3 \vdash^? b$$

These transformation rules are rather schemes of rules: the side conditions both may impose some restrictions on the parameters of the rule and on the constraint, on which it is applied. They also specify the resulting constraint. In other words, we can see the side conditions as a way to schematize a (possibly infinite, yet recursive) set of rules.

Of course, these transformation rules will not be applied without restriction, otherwise we would roughly enumerate all possible attackers recipes and, though this would be complete, this would certainly not terminate. For instance, we will avoid to apply $\textsc{Cons}(X, \mathsf{f})$ to $X, i \vdash^? t$ when t is a variable, or $\textsc{Dest}(\xi, l \to r, i)$ to $\xi, j \rhd v$ when v is a variable. These restrictions will be explained at the beginning of Section 4.

Transformation rules for static equivalence (Figure 2). For equivalence properties, it is necessary to ensure that the observable identities are the same on both systems. Let us illustrate this point on an example.

Example 26. *Consider the frames:*

- $\Phi_1 = \{ax_1, 1 \rhd a;\;\; ax_2, 2 \rhd k_1;\;\; ax_3, 3 \rhd \mathsf{senc}(x, k);\;\; ax_4, 4 \rhd \mathsf{senc}(\mathsf{senc}(a, k_1), k)\}$
- $\Phi_2 = \{ax_1, 1 \rhd a;\;\; ax_2, 2 \rhd k_1;\;\; ax_3, 3 \rhd \mathsf{senc}(x, k);\;\; ax_4, 4 \rhd \mathsf{senc}(b, k)\}$.

If $x = \mathsf{senc}(a, k_1)$, then the two frames are not statically equivalent since $ax_3 = ax_4$ is an equality satisfied on the first frame and not on the second. If $x \neq \mathsf{senc}(a, k_1)$ on the first frame, and $x \neq b$ on the second one, then the two frames are statically equivalent. If, for instance, the deducibilityy constraint associated with both frames is $X, 2 \vdash^? x$, then the rules of Figure 1 will not help in finding the witness of non-equivalence.

Another set of rules, the *equality rules* described in Figure 2, will be used in such a situation. The purpose of these equality rules is to distinguish between cases, in which some equalities hold and cases, in which they do not hold. The relevant equalities that we have to consider concern the right-hand sides of deducibility constraints and/or members of the frame. These rules do not correspond to attacker's actions and they are not necessary if we are only interested in reachability properties.

The first set of rules allowed roughly to reduce any deducibility constraint to a conjunction of constraints $X, i \vdash^? x$ where X, x are variables. It allows further to assume that any relevant application of destructors to the frame has been recorded in the frame. We still miss immediately observable equalities between members of the frame: such equalities must hold (or not) simultaneously on two equivalent contraint systems. This is the purpose of the rule Eq-frame-frame, that makes a case distinction, depending on equalities between members of the frame.

This rule has to be complemented with a rule that computes other deducible subterms, that are not obtained from the frame by applications of destructors (see Example 30 and the rule Ded-st).

There are still a few ways to distinguish between two constraint systems:

- if a constraint imposes that two recipes yield the same result, then it must also be the case in the other constraint (see Example 29). Similarly if a constraint allows (for some instance of the variables) that two recipes yield the same result, this must be true on the other constraint (see Example 29 again). This is the purpose of the rule Eq-ded-ded, which investigates possible equalities relating the members of deducibility constraints.

- if some instance of a variable can be computed at an earlier stage than what is imposed by the deducibility constraint, it has to be an an earlier stage also in the other constraint system. The rule Eq-frame-ded therefore investigates whether or not some member of a deducibility constraint can be obtained at an earlier stage.

As we prove later, this finally covers all the possible situations.

$$\begin{tikzcd}[column sep=small]
 & E \wedge u_1 =^? u_2 \\
\underline{\textsc{Eq-frame-frame}(\xi_1,\xi_2)} : \; E \arrow[ur] \arrow[dr] & \\
 & E \wedge u_1 \neq^? u_2
\end{tikzcd}$$

where $\xi_1, i_1 \rhd u_1,\ \xi_2, i_2 \rhd u_2 \in \Phi$ for some ξ_1, ξ_2, i_1, i_2

$$\begin{tikzcd}[column sep=small]
 & E \wedge u_1 =^? u_2, \mathsf{NoUse} \cup (\xi_1, i_1 \rhd u_1) \\
\underline{\textsc{Eq-frame-ded}(\xi_1,X_2)} : \; E, \mathsf{NoUse} \arrow[ur] \arrow[dr] & \\
 & E \wedge u_1 \neq^? u_2, \mathsf{NoUse}
\end{tikzcd}$$

where $\xi_1, i_1 \rhd u_1 \in \Phi$ and $X_2, i_2 \vdash^? u_2 \in D$, with $i_2 < i_1$ and $X_2 \in S_2$ for some ξ_1, ξ_2, u_1, u_2

$$\begin{tikzcd}[column sep=small]
 & E \wedge u =^? v;\ E_\Pi \wedge X =^? \xi \\
\underline{\textsc{Eq-ded-ded}(X,\xi)} : \; X, i \vdash^? u;\ E;\ E_\Pi \arrow[ur] \arrow[dr] & \\
 & X, i \vdash^? u;\ E \wedge u \neq^? v;\ E_\Pi
\end{tikzcd}$$

where $\xi \in \mathcal{T}(\mathcal{F}_c, \mathrm{dom}(\alpha))$, $v = \xi\alpha$ with $\alpha = \{Y \mapsto w \mid (Y, j \vdash^? w) \in D \wedge j \leq i \wedge Y \in S_2\}$. Moreover, we assume that:

- if $\mathsf{root}(\xi) = \mathsf{f}$ then $E_\Pi \not\models \mathsf{root}(X) \neq^? \mathsf{f}$
- if $\xi = Y$ then for all $\mathsf{f} \in \mathcal{F}_c$, $E_\Pi \models \mathsf{root}(X) \neq^? \mathsf{f}$ is equivalent to $E_\Pi \models \mathsf{root}(Y) \neq^? \mathsf{f}$.

$$\begin{tikzcd}[column sep=small]
 & \begin{array}{l} \Phi;\ X_1, \mathsf{s}_{max} \vdash^? x_1;\ \cdots;\ X_n, \mathsf{s}_{max} \vdash^? x_n \\ E \wedge u =^? \mathsf{f}(x_1, \ldots, x_n);\ ND \end{array} \\
\underline{\textsc{Ded-st}(\xi,\mathsf{f})} : \Phi; E; ND \arrow[ur] \arrow[dr] & \\
 & \begin{array}{l} \Phi;\ E;\ ND \\ \forall \tilde{x} \cdot [u \neq^? \mathsf{f}(x_1, \ldots, x_n) \vee \mathsf{s}_{max} \not\vdash^? x_1 \vee \ldots \vee \mathsf{s}_{max} \not\vdash^? x_n] \end{array}
\end{tikzcd}$$

If Φ contains $\xi, i \rhd u$ and $(\xi, i \rhd u) \notin \mathsf{NoUse}$. The sequences $\tilde{x} = x_1, \ldots, x_n$, and $X_1, \ldots, X_n$ are sequences of fresh variables and s_{max} represents the maximal index that occurs in $\mathcal{C}$.

Figure 2: Additional transformation rules for static equivalence

Example 27. *Let us come back to Example 26. Applying* Eq-frame-frame(ax_3, ax_4) *to the first constraint system, we get:*

$$\left\{ \begin{array}{l} \Phi_1, \quad X, 2 \vdash^? x \\ \mathsf{senc}(x, k) =^? \mathsf{senc}(\mathsf{senc}(a, k_1), k) \end{array} \right. \qquad \left\{ \begin{array}{l} \Phi_1, \quad X, 2 \vdash^? x \\ \mathsf{senc}(x, k) \neq^? \mathsf{senc}(\mathsf{senc}(a, k_1), k) \end{array} \right.$$

The case $x = \mathsf{senc}(a, k_1)$ is now distinguished from the case $x \neq \mathsf{senc}(a, k_1)$.

Here, we could additionally put the frame element $ax_4 \rhd \mathsf{senc}(\mathsf{senc}(a, k_1), k)$ of the first constraint system in the set NoUse (thus forbidding the use of this element). However, as illustrated by the following example (Example 28), this is not always possible, and thus this feature is not present in the rule Eq-frame-frame contrary to what is done in the rule Eq-frame-ded.

Example 28. *Let* $\Phi = \{ax_1, 1 \rhd \langle k, \mathsf{senc}(k,k)\rangle\}$. *After applying some transformation rules, assume that we reach the frame:*

$$\Phi^+ = \{ax_1, 1 \rhd \langle k, \mathsf{senc}(k,k)\rangle;\ \mathsf{proj}_1(ax_1), 1 \rhd k;\ \mathsf{proj}_2(ax_1), 1 \rhd \mathsf{senc}(k,k);\ \mathsf{sdec}(\mathsf{proj}_2(ax_1), \mathsf{proj}_1(ax_1)), 1 \rhd k\}.$$

Applying Eq-frame-frame(ξ_1, ξ_2) *with* $\xi_1 = \mathsf{proj}_1(ax_1)$ *and* $\xi_2 = \mathsf{sdec}(\mathsf{proj}_2(ax_1), \mathsf{proj}_1(ax_1))$, *we will be tempted to put* $\mathsf{proj}_1(ax_1) \rhd k$ *in* NoUse. *If so, in order to deduce* k*, we now need to use the recipe* $\xi_2 = \mathsf{sdec}(\mathsf{proj}_2(ax_1), \mathsf{proj}_1(ax_1))$ *which is not conform to* Φ^+ *since* $\mathsf{proj}_1(ax_1) \in st(\xi_2)$ *(see Definition 6). When we put a frame element in the set* NoUse*, we have to be sure that it has not been already used to build another frame element.*

Example 29. *Consider the two constraint systems:*

$$\Phi_1 = \{ax_1, 1 \rhd a;\ ax_2, 2 \rhd b;\ ax_3, 3 \rhd x_1\},\ \ D_1 = \{X, 1 \vdash^? x_1;\ Y, 2 \vdash^? x_1;\ Z, 3 \vdash^? y_1\}$$
$$\Phi_2 = \{ax_1, 1 \rhd a;\ ax_2, 2 \rhd b;\ ax_3, 3 \rhd x_2\},\ \ D_2 = \{X, 1 \vdash^? x_2;\ Y, 2 \vdash^? y_2;\ Z, 3 \vdash^? y_2\}$$

There are redundant constraints in each individual system. However, we need $x_1 = y_1$ *and* $x_2 = y_2$ *in order to get equivalent systems, since the recipes* X, Y *must yield the same value, according to the first system (hence* $x_2 = y_2$*) and the recipes* Y, Z *yield the same value, according to the second system (hence* $x_1 = y_1$*). The rule* Eq-ded-ded *takes care of such situations: we guess whether different recipe variables yield the same value and record the result of the guess in the constraint.*

Actually, we sometimes need to apply Eq-ded-ded *with a recipe* ξ *which is not reduced to a variable. In particular, this is the case when we have to handle disequations between terms. Consider the two constraint systems:*

$$D_3 := \{Z, 1 \vdash^? z;\ Y, 2 \vdash^? y;\ X, 3 \vdash^? x\},\ \text{with } E_3 :=\ x \neq^? \langle y, z\rangle$$
$$D_4 := \{Z, 1 \vdash^? z;\ Y, 2 \vdash^? y;\ X, 3 \vdash^? x\}$$

The deducibility constraints are identical in both systems but the disequations in E_3 *should be satisfied by both constraint systems in order to be equivalent. By applying the rule* Eq-ded-ded *with* X *and* $\xi = \langle Y, Z\rangle$*, we split the solutions of both constraint systems into two disjoint sets: those that satisfy the equation* $x =^? \langle y, z\rangle$*, and those that do not satisfy this equation.*

As it is displayed, the rule Eq-ded-ded has infinitely many instances, as the recipe ξ is an arbitrary constructor recipe. Our strategy will later restrict the situations, in which the rule has to be applied, so that only specific instances are needed.

Finally, the last transformation rule of Figure 2 investigates the possible deducible subterms of the frame. This is a necessary step to capture static equivalence in presence of non-invertible primitives such as hash function and asymmetric encryption.

Example 30. *Consider the two constraint systems:*

- $\Phi_1 = \{ax_1, 1 \rhd \mathsf{pub}(ska);\ ax_2, 2 \rhd \mathsf{aenc}(x, \mathsf{pub}(ska))\}$ *and* $D_1 = \{X, 1 \vdash^? x\}$*;*
- $\Phi_2 = \{ax_1, 1 \rhd \mathsf{pub}(ska);\ ax_2, 2 \rhd \mathsf{aenc}(b, \mathsf{pub}(ska))\}$ *and* $D_2 = \{X, 1 \vdash^? x\}$.

Intuitively, the transformation rules we have seen so far do not help in simplifying any of the two constraint systems. The only relevant possibility would be to try decrypting $\mathsf{aenc}(x, \mathsf{pub}(sk_a))$*, but the private key ska is not deducible. Nevertheless, the two constraint systems are not equivalent since the attacker can construct* $\mathsf{aenc}(x, \mathsf{pub}(sk_a))$ *(using the recipe* $\mathsf{aenc}(X, ax_1)$*) and therefore observe the identity* $\mathsf{aenc}(X, ax_1) = ax_2$ *on* Φ_1*, which is not possible on* Φ_2*. This is the reason of the rule* DED-ST*, that guesses the subterms of the frame that can be constructed by the attacker. In the above example, the first constraint system would become:*

$$\Phi_1, \quad \{X, 1 \vdash^? x;\ X_1, 2 \vdash^? x;\ X_2, 2 \vdash^? \mathsf{pub}(ska)\}$$

(the other branch is unsatisfiable), while on the second constraint system, we get:

$$\Phi_2, \quad \{X, 1 \vdash^? x;\ X_1, 2 \vdash^? b;\ X_2, 2 \vdash^? \mathsf{pub}(ska)\}$$

Eventually, this last constraint will be proven unsatisfiable, witnessing the non-equivalence of the constraint systems.

Our rule DED-ST only considers a one-step intruder deduction, in order to incrementally check equalities and disequalities, avoiding unnecessary blow ups.

Now, before explaining how to apply the rules on pairs of sets of constraint systems, we formalise what we used implicitly in all our examples, *i.e.,* normalisation of constraint systems after the application of a transformation rule.

3.2. Normalisation

The normalisation consists mainly in simplifying the equations and disequations and performing the replacements when relevant. The *normalisation rules* are displayed in Figure 3. As usual, substitutions are confused with solved conjunctions of equations. We also switch sometimes the order of the components of a constraint system in order to ease the display, and omit irrelevant parts of the constraint system.

Here, mgu is a function that maps any conjunction of equations to their most general unifier. Such a unifier is confused with a solved system of equations, which is either $\perp$ or a conjunction $x_1 =^? t_1 \wedge \ldots \wedge x_n =^? t_n$, where $x_1, \ldots, x_n$ are variables that appear only once. Furthermore, the variables of $\mathsf{mgu}(e)$ are contained in the variables of e and $\mathsf{mgu}(\mathsf{mgu}(e)) = \mathsf{mgu}(e)$.

The first four rules simplify equations between terms/recipes. The last four rules simplify the disequations on recipes, removing them when they are trivially satisfied, or replacing the whole system with $\perp$ when they are trivially inconsistent. The remaining rules simplify disequations between (first-order) terms taking care of variables that are universally quantified. The soundness of the rules follows from complete axiomatisations of the free term algebra (see *e.g.,* [31]); let us recall that these formulas are interpreted in the free constructor algebra, according to Definition 4. If the simplification rules are applied only when they modify a constraint, then the set of rules is strongly terminating, i.e., any rewriting strategy is terminating.

We further apply two normalisation rules, that are displayed in Figure 4. Before explaining these rules, we say that the application of the rule CONS(X, f), AXIOM(X, path), or DEST$(\xi, l \to r, i)$ is *useless on a constraint system* $\mathcal{C}$ if such a rule is not applicable or if a similar instance has already been applied along the same branch. In case of the rule CONS and AXIOM, by similar, we mean that exactly the same instance has already been

$$
\begin{array}{lcl}
\Phi; D; E_\Pi; ND; \mathsf{NoUse}; E \wedge \bigwedge_{i=1}^n u_i =^? v_i & \rightsquigarrow & \Phi\sigma; D\sigma; E_\Pi; ND\sigma; \mathsf{NoUse}\sigma; E\sigma \wedge \sigma \\
 & & \text{if } \sigma = \mathsf{mgu}(\bigwedge_{i=1}^n u_i =^? v_i) \\
\Phi; D; E_\Pi; ND; \mathsf{NoUse}; E \wedge \bigwedge_{i=1}^n u_i =^? v_i & \rightsquigarrow & \bot \qquad \text{if } \mathsf{mgu}(\bigwedge_{i=1}^n u_i =^? v_i) = \bot \\
\Phi; D; E; ND; \mathsf{NoUse}; E_\Pi \wedge \bigwedge_{i=1}^n \zeta_i =^? \xi_i & \rightsquigarrow & \Phi\theta; D; E; ND; \mathsf{NoUse}\theta; E_\Pi\theta \wedge \theta \\
 & & \text{if } \theta = \mathsf{mgu}(\bigwedge_{i=1}^n \zeta_i =^? \xi_i) \\
\Phi; D; E; ND; \mathsf{NoUse}; E_\Pi \wedge \bigwedge_{i=1}^n \zeta_i =^? \xi_i & \rightsquigarrow & \bot \qquad \text{if } \mathsf{mgu}(\bigwedge_{i=1}^n \zeta_i =^? \xi_i) = \bot
\end{array}
$$

$$
\begin{array}{lcll}
E \wedge \forall \tilde{x}.[\bigvee_{i=1}^n u_i \neq^? v_i] & \rightsquigarrow & E & \text{if } \mathsf{mgu}(\bigwedge_{i=1}^n u_i =^? v_i) = \bot \\
E \wedge \forall \tilde{x}.[E' \vee u \neq^? u] & \rightsquigarrow & E \wedge \forall \tilde{x}.E' & \\
E \wedge \forall \tilde{x}.[E' \vee x \neq^? u] & \rightsquigarrow & E \wedge \forall \tilde{x} \smallsetminus \{x\}.E'\sigma & \text{if } x \in \tilde{x} \smallsetminus \mathit{vars}^1(u) \text{ and } \sigma = \{x \to u\} \\
E \wedge \forall \tilde{x}.\forall x.E' & \rightsquigarrow & E \wedge \forall \tilde{x}.E & \text{if } x \notin \mathit{vars}^1(E) \\
E \wedge \forall \tilde{x}.[E' \vee \mathsf{f}(u_1, \dots, u_n) \neq^? \mathsf{f}(v_1, \dots v_n)] & \rightsquigarrow & E \wedge \forall \tilde{x}.[E' \vee \bigvee_{i=1}^n u_i \neq^? v_i] & \\
E \wedge u \neq^? v \wedge \forall \tilde{x}.[E' \vee u \neq^? v] & \rightsquigarrow & E \wedge u \neq^? v &
\end{array}
$$

$$
\begin{array}{lcll}
E_\Pi \wedge \zeta \neq^? \xi & \rightsquigarrow & E_\Pi & \text{if } \mathsf{mgu}(\zeta, \xi) = \bot \\
E_\Pi \wedge \zeta \neq^? \zeta & \rightsquigarrow & \bot & \\
E_\Pi \wedge \mathsf{root}(\mathsf{f}(\xi_1, \dots, \xi_n)) \neq \mathsf{f} & \rightsquigarrow & \bot & \\
E_\Pi \wedge \mathsf{root}(\mathsf{f}(\xi_1, \dots, \xi_n)) \neq \mathsf{g} & \rightsquigarrow & E_\Pi & \text{if } \mathsf{f} \neq \mathsf{g}
\end{array}
$$

Figure 3: Normalisation rules for side constraints

$E \wedge \forall \tilde{x}.[E' \vee x \neq^? a] \rightsquigarrow E$
if $a \in \mathcal{N}$, $(X, i \vdash^? x) \in D$, $\textsc{Axiom}(X, \mathsf{path})$ is useless for any path and
$\textsc{Dest}(\xi, l \to r, i)$ is useless for any $\xi, l \to r$, and
for all $(\zeta, j \rhd v) \in \Phi$, $j \leq i$ and $v \in \mathcal{X}^1$ implies $(\zeta, j \rhd v) \in \mathsf{NoUse}$

$D \wedge X, i \vdash^? u \rightsquigarrow \bot$
if $\textsc{Cons}(X, \mathsf{f})$ is useless for all $\mathsf{f} \in \mathcal{F}_c$; and $\textsc{Axiom}(X, \mathsf{path})$ is useless for any path; and
$\textsc{Dest}(\xi, l \to r, i)$ is useless for all $\xi, l \to r$; and
for all $(\zeta, j \rhd v) \in \Phi$, $j \leq i$ and $v \in \mathcal{X}^1$ implies $(\zeta, j \rhd v) \in \mathsf{NoUse}$

Figure 4: Two additional normalisation rules

applied. In case of the rule $\textsc{Dest}$, additional instances are actually useless. Indeed, there is no need to apply *e.g.,* $\textsc{Dest}(\xi, l \to r, 3)$ on the branch where a "successful" application of $\textsc{Dest}(\xi, l \to r, 2)$ has been considered. We illustrate this concept through an example.

Example 31. *Continuing with the constraint systems named $\mathcal{C}_1^{\textsc{Dest}}$ and $\mathcal{C}_2^{\textsc{Dest}}$, obtained in Example 25, we have that:*

- $\textsc{Dest}(ax_1, l \to r, 2)$ *and* $\textsc{Dest}(ax_1, l \to r, 3)$ *are useless on $\mathcal{C}_1^{\textsc{Dest}}$ since an element of the form $\mathsf{sdec}(ax_1, \_), 2 \rhd \_$ is already present in the frame;*
- $\textsc{Dest}(ax_1, l \to r, 2)$ *is useless on $\mathcal{C}_2^{\textsc{Dest}}$ since the associated non-deducibility constraint already occurs in the constraint system;*
- $\textsc{Dest}(ax_3, l \to r, 1)$ *and* $\textsc{Dest}(ax_3, l \to r, 2)$ *are useless on both $\mathcal{C}_1^{\textsc{Dest}}$ and $\mathcal{C}_2^{\textsc{Dest}}$ since they do not contain any frame element of the form $ax_3, j \rhd \_$ with $j \leq 2$;*

where $(l, r) = (\mathsf{sdec}(\mathsf{senc}(x, y), y), x)$.

Now, we can explain rules displayed in Figure 4. Intuitively, the first rule states that x cannot be the name a, if it has to be deducible and cannot be obtained by $\textsc{Axiom}$ or $\textsc{Dest}$. The second rule states that, in order to deduce a message, the attacker has either to construct it from deducible messages, or retrieve it from the frame and deducible messages. In other words, any attacker's ground recipe is built using $\mathcal{F}_c$, $\mathcal{F}_d$ and the ax_i.

Definition 11 (normalisation)**.** *If $\mathcal{C}$ is a constraint system, we let $\mathcal{C}\!\downarrow$ be an irreducible form of $\mathcal{C}$, w.r.t. the rules of Figures 3 and 4.*

In what follows we assume that every constraint system is eagerly normalised.

3.3. From constraint systems to pairs of sets of constraint systems

Given two constraint systems, we cannot only simplify them independently and then check for their equivalence, because actions and tests must be performed by the attacker in the same way in both experiments.

Example 32. *Consider the two following frames:*

$\Phi_1 = \{ax_1, 1 \rhd a;\ ax_2, 2 \rhd a;\ ax_3, 3 \rhd b\}$, *and* $\Phi_2 = \{ax_1, 1 \rhd a; ax_2, 2 \rhd b; ax_3, 3 \rhd a\}$.

For sake of simplicity, assume there are no deducibility constraints. These two frames are not statically equivalent. Indeed, choosing $\zeta_1 = ax_1$ and $\zeta_2 = ax_2$, we have that $\zeta_1\Phi_1\!\downarrow = \zeta_2\Phi_1\!\downarrow$, while $\zeta_1\Phi_2\!\downarrow \neq \zeta_2\Phi_2\!\downarrow$.

On the other hand, we may only apply the $\textsc{Eq-frame-frame}$ *rule, whose (in each case) one of the branches yields $\perp$ and on the other branch the constraint system is unchanged. Hence the rules do not help witnessing the non-equivalence, unless we apply the same instance of* $\textsc{Eq-frame-frame}$ *simultaneously on both frames. For instance* $\textsc{Eq-frame-frame}(ax_1, ax_2)$ *applied to Φ_1 and Φ_2 simultaneously yields Φ_1 and $\perp$ on one branch, and $\perp$ and Φ_2 on the other.*

Intuitively, each transformation rule of Figure 1 corresponds to an action of the attacker and each rule of Figure 2 corresponds to a test of the attacker. Two constraints systems are therefore symbolically equivalent if, and only if, applying the *same rule* on both systems yields (on each branch) symbolically equivalent constraint systems.

In the introduction we motivate the need of considering sets of constraint systems, and more precisely pairs of sets of constraint systems. We explain now how our transformation rules will be applied in such a setting. Remember that sets of constraint systems are sequences of constraint systems sharing the same structure (see Definition 9). The basic idea is to apply the same transformation rule (with the same parameters) on each constraint system of each set. Note that, the parameters of a transformation rule only depend on the structure of the underlying constraint system. Thanks to this, the simultaneous application of a transformation rule can be defined in a natural way.

Given $\mathcal{S} = [\mathcal{C}_1, \ldots, \mathcal{C}_n]$ and $\mathcal{S}' = [\mathcal{C}'_1, \ldots, \mathcal{C}'_{n'}]$ two sets of size n (resp. n') of constraint systems having the same structure, and $\textsc{Rule}(\tilde{p})$ an instance of a transformation rule. The application of $\textsc{Rule}(\tilde{p})$ on the pair $(\mathcal{S}, \mathcal{S}')$ yields two pairs of sets of constraint systems $(\mathcal{S}_1, \mathcal{S}'_1)$ and $(\mathcal{S}_2, \mathcal{S}'_2)$ such that:

$$\begin{tikzcd} & (\mathcal{S},\mathcal{S}') \arrow[dl] \arrow[dr] & \\ ([\mathcal{C}_{1,1},...,\mathcal{C}_{1,n}],[\mathcal{C}'_{1,1},...,\mathcal{C}'_{1,n'}]) =: (\mathcal{S}_1,\mathcal{S}'_1) & & (\mathcal{S}_2,\mathcal{S}'_2) := ([\mathcal{C}_{2,1},...,\mathcal{C}_{2,n}],[\mathcal{C}'_{2,1},...,\mathcal{C}'_{2,n'}]) \end{tikzcd}$$

where:

- for all $i \in \{1 \ldots n\}$, the constraint systems $\mathcal{C}_{1,i}$ and $\mathcal{C}_{2,i}$ are those obtained by application of $\textsc{Rule}(\tilde{p})$ on $\mathcal{C}_i$; and
- for all $i \in \{1 \ldots n'\}$, the constraint systems $\mathcal{C}'_{1,i}$ and $\mathcal{C}'_{2,i}$ are those obtained by application of $\textsc{Rule}(\tilde{p})$ on $\mathcal{C}'_i$.

3.4. Matrices of constraint systems

The transformation of pairs of sets of constraint systems make successive case distinctions. If we only work with pairs of sets of constraint systems, we loose the relationships between the two complementary cases. This is harmless when the case distinction roughly corresponds to attacker's actions (later called *external applications*). There are however situations, in which we wish to remember that two constraints originate from the same one, and hence are complementary. We give some examples below, as well as in Section 4.

Keeping track of complementary constraints is achieved through *matrices* of constraint systems: each row corresponds to a set (now a sequence) of constraint systems, while the columns correspond to complementary choices. The matrices can be seen as a merge of some branches of the case distinction tree of the previous section.

The non-deducibility constraints introduced by the rules Dest and Ded-st allow us to properly divide the solutions of a constraint system. However, no transformation rule solves these non-deducibility constraints. When a non-deducibility constraint is introduced, the idea is to take advantage of informations that are collected, while solving the constraint in the other branch. This requires however gathering together in the same structure the constraint systems that result from the application of a rule introducing non-deducibility constraints.

Example 33. *Let $\mathcal{C}$ be a constraint system with the deducibility constraint $X, 2 \vdash^? \mathsf{senc}(a, a)$, and the frame $\Phi = \{ax_1, 1 \rhd a;\ ax_2, 2 \rhd \mathsf{senc}(b, a)\}$.* $\textsc{Dest}(ax_2, \mathsf{sdec}(\mathsf{senc}(x, y), y) \to x, 2)$ *applied on $\mathcal{C}$ gives us:*

$$\mathcal{C}_1 = \left\{ \begin{array}{l} \Phi,\ \mathsf{sdec}(ax_2, Y), 2 \rhd b \\ X, 2 \vdash^? \mathsf{senc}(a, a);\ Y, 2 \vdash^? a \end{array} \right. \quad \mathcal{C}_2 = \left\{ \begin{array}{l} \Phi;\ \ X, 2 \vdash^? \mathsf{senc}(a, a) \\ \forall x_1, x_2.[\mathsf{senc}(x_1, x_2) \neq^? \mathsf{senc}(b, a) \vee 2 \not\vdash^? x_2] \end{array} \right.$$

To solve the non-deducibility constraint in $\mathcal{C}_2$, we will use the information we get from the transformation rules applied on $\mathcal{C}_1$. Applying $\textsc{Axiom}(Y, ax_1)$ *on $\mathcal{C}_1$, we get:*

$$\mathcal{C}_3 = \left\{ \begin{array}{l} \Phi,\ \mathsf{sdec}(ax_2, ax_1), 2 \rhd b \\ X, 2 \vdash^? \mathsf{senc}(a, a) \\ Y =^? ax_1 \end{array} \right. \qquad \mathcal{C}_4 = \left\{ \begin{array}{l} \Phi,\ \mathsf{sdec}(ax_2, Y), 2 \rhd b \\ X, 2 \vdash^? \mathsf{senc}(a, a);\ Y, 2 \vdash^? a \\ Y \neq^? ax_1 \end{array} \right.$$

Using successive applications of $\textsc{Cons}(Y, \mathsf{f})$ *and* $\textsc{Axiom}(Y, \mathsf{path})$ *for any* f *and* path*, $\mathcal{C}_4$ is eventually reduced to $\bot$. Now, the formula $\forall x_1, x_2.[\mathsf{senc}(x_1, x_2) \neq^? \mathsf{senc}(b, a) \vee 2 \not\vdash^? x_2]$ has no free variable and its negation is a consequence of $\mathcal{C}_1$ (since the rule splits the set of solutions). The satisfiability of $\mathcal{C}_3$ therefore implies the unsatisfiability of $\mathcal{C}_2$.*

If we wish to perform such an inference, we need to keep in the same structure the constraints $\mathcal{C}_2$ and $\mathcal{C}_3$ (instead of solving them independently).

As illustrated in Example 33 above, solving the non-deducibility constraints will rely on the information obtained from the application of the rules on other constraint systems. That is why we gather together sets of constraint systems, and organise them into matrices, each row being a set of constraint systems.

Infering the unsatisfiability of $\mathcal{C}_2$ in Example 33 requires some properties of $\mathcal{C}_3$, typically that $\mathcal{C}_3$ is in pre-solved form. In more general situations, in which $\mathcal{C}_2$ contains (free first-order) variables that appear also in $\mathcal{C}_1$, we will also be able to apply such an inference, thanks to an invariant: if $\mathcal{C}$, $\mathcal{C}'$ are two constraint systems in the same column such that the frame of $\mathcal{C}$ is (roughly) a strict superset of the frame of $\mathcal{C}'$, then, for any assignment σ of the free first-order variables of $\mathcal{C}$, the solutions of $\mathcal{C}\sigma$ and $\mathcal{C}'\sigma$ are disjoint.

This relationship between constraints in the same column is eventually exploited in the step e of Section 4.

Example 34. *Applying* $\textsc{Dest}(ax_2, \mathsf{sdec}(\mathsf{senc}(x, y), y) \to x, 2)$ *on (the set) $[\mathcal{C}]$ given in Example 33, a priori yields two sets $[\mathcal{C}_1]$ and $[\mathcal{C}_2]$ where $\mathcal{C}_1, \mathcal{C}_2$ are the constraint systems of Example 33. We group however the resulting systems in the following matrix:*

$$\begin{bmatrix} \mathcal{C}_1 \\ \mathcal{C}_2 \end{bmatrix}$$

Applying $\textsc{Axiom}(Y, ax_1)$*, we get now the following matrix of constraint systems:*

$$\begin{bmatrix} \mathcal{C}_3 \\ \mathcal{C}_4 \\ \mathcal{C}_2 \end{bmatrix}$$

In order to check the non-deducibility constraint in $\mathcal{C}_2$, we only need the information provided by the constraint systems in the same column as $\mathcal{C}_2$. We will see later on how to exploit this information (see Section 4 - Phase 1 / Step e).

Note that in Example 34, the constraint systems $\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$, and $\mathcal{C}_4$ have the same set S_1 (resp. S_2) of first-order (resp. recipe) variables. Moreover, they have the same shape according to the following definition.

Definition 12 (shape)**.** *Let* $\mathcal{C} = (S_1; S_2; \Phi; D; E; E_\Pi; ND; \mathsf{NoUse})$ *be a constraint system. The* shape *of* $\mathcal{C}$ *is given by* S_2, *and* $\{(X,i) \mid X, i \vdash^? u \in D \text{ and } X \in S_2\}$.

Intuitively, the shape of a constraint system only takes into account the free recipe variables, *i.e.,* those that represent the actions of the attacker. By convention, we assume that the constraint system $\bot$ has the same shape as any other constraint system.

We extend the notion of *same structure* to matrices of constraint systems as follows: $\mathcal{M}$ (n rows, m columns) and $\mathcal{M}'$ (n' rows, m' columns) have the *same structure* if:

- all the constraint systems stored in $\mathcal{M}$ and $\mathcal{M}'$ have the same shape;
- $n = n'$, *i.e.,* $\mathcal{M}$ and $\mathcal{M}'$ have the same number of rows; and
- for all $i \in \{1 \dots n\}$, the constraint systems stored in the i^{th} row of the matrices $\mathcal{M}$ and $\mathcal{M}'$ have the same structure.

In fact, introducing matrices of constraint systems serves a greater purpose than just solving the non-deducibility constraints. Indeed, deciding the symbolic equivalence of sets of constraint systems contains two main issues:

- matching an existing solution from one set to the other;
- and deciding whether the two resulting frames are statically equivalent or not.

The idea behind matrices with several rows is to keep all the guesses on static equivalence into a single matrix. Intuitively, when we guess the form of the solutions, we split the matrix into two matrices. However, when we guess an equality between terms or a property on static equivalence, we gather the information in a single matrix. We thus consider two kinds of application: *internal* and *external.*

The transformation rules Ded-st, Eq-frame-frame, Eq-frame-ded and Dest will be applied internally whereas Cons(X, f), Axiom(X, path) and Eq-ded-ded(X, ξ) will be applied externally when $X \in S_2$ and internally otherwise (*i.e.,* $X \notin S_2$).

Internal/external application of a transformation rule. Let $(\mathcal{M}, \mathcal{M}')$ be a pair of matrices of constraint systems having the same structure. In particular, $\mathcal{M}$ and $\mathcal{M}'$ have the same number of rows, say n. Let $\mathcal{M} = [\mathcal{S}_1, \dots, \mathcal{S}_n]$ and $\mathcal{M}' = [\mathcal{S}'_1, \dots, \mathcal{S}'_n]$. Let Rule$(\tilde{p})$ be an instance of a transformation rule and i be an index representing a row, *i.e.,* $1 \leq i \leq n$.

An *internal application* of Rule$(\tilde{p})$ to the i^{th} row of the pair $(\mathcal{M}, \mathcal{M}')$ yields a pair of matrices $(\tilde{\mathcal{M}}, \tilde{\mathcal{M}'})$ such that:

$$\tilde{\mathcal{M}} = [\mathcal{S}_1, \dots, \mathcal{S}_{i-1}, \mathcal{T}_{1,i}, \mathcal{T}_{2,i}, \mathcal{S}_{i+1}, \mathcal{S}_n] \qquad \tilde{\mathcal{M}'} = [\mathcal{S}'_1, \dots, \mathcal{S}'_{i-1}, \mathcal{T}'_{1,i}, \mathcal{T}'_{2,i}, \mathcal{S}'_{i+1}, \mathcal{S}'_n]$$

where $(\mathcal{T}_{1,i}, \mathcal{T}'_{1,i})$ and $(\mathcal{T}_{2,i}, \mathcal{T}'_{2,i})$ are the pair of row matrices obtained by applying Rule$(\tilde{p})$ on $(\mathcal{S}_i, \mathcal{S}'_i)$. Note that, since the two matrices $\mathcal{M}$ and $\mathcal{M}'$ have the same structure, the two sets $\mathcal{S}_i$ and $\mathcal{S}'_i$ have the same structure too and we have already seen how to apply a transformation rule in such a situation. We actually obtain two matrices with $n+1$ rows. We say that an instance $\textsc{Rule}(\tilde{p})$ of a rule is *internally applicable* on $(\mathcal{M}, \mathcal{M}')$ on row i if $\textsc{Rule}(\tilde{p})$ is applicable on $(\mathcal{S}_i, \mathcal{S}'_i)$.

An *external application* of $\textsc{Rule}(\tilde{p})$ on $(\mathcal{M}, \mathcal{M}')$ yields two pairs of matrices $(\tilde{\mathcal{M}}_1, \tilde{\mathcal{M}}'_1)$ and $(\tilde{\mathcal{M}}_2, \tilde{\mathcal{M}}'_2)$ such that:

$$\begin{array}{ll} \tilde{\mathcal{M}}_1 = [\mathcal{T}_{1,1}, \dots, \mathcal{T}_{1,n}] & \qquad \tilde{\mathcal{M}}'_1 = [\mathcal{T}'_{1,1}, \dots, \mathcal{T}'_{1,n}] \\ \tilde{\mathcal{M}}_2 = [\mathcal{T}_{2,1}, \dots, \mathcal{T}_{2,n}] & \qquad \tilde{\mathcal{M}}'_2 = [\mathcal{T}'_{2,1}, \dots, \mathcal{T}'_{2,n}] \end{array}$$

where $(\mathcal{T}_{1,i}, \mathcal{T}'_{1,i})$ and $(\mathcal{T}_{2,i}, \mathcal{T}'_{2,i})$ are the pairs of sets obtained by applying $\textsc{Rule}(\tilde{p})$ on $(\mathcal{S}_i, \mathcal{S}'_i)$ for each $i \in \{1, \dots, n\}$. Each resulting pair of matrices has exactly the same numbers of rows and columns as the original one $(\mathcal{M}, \mathcal{M}')$.

Remark. Unfortunately, all the constraint systems in $\mathcal{M}$ and $\mathcal{M}'$ do not necessarily have the same structure, but only the same shape. When the external application involved is an instance of a rule $\textsc{Cons}$, it is easy to see that having the same shape will ensure that the rule can be applied on each set, *i.e.,* on each row of the matrix. Regarding an external application of the rule $\textsc{Axiom}(X, \mathsf{path})$, we have to be careful. Since the constraint systems have the same shape and we know that $X \in S_2$, we can ensure that X occurs in each constraint system. However, it could happen that some rows do not contain the required frame element. By convention, in such a pair $(\mathcal{S}_i, \mathcal{S}'_i)$ of row matrices, the resulting pairs of row matrices are $(\mathcal{T}_{1,i}, \mathcal{T}'_{1,i}) \overset{\mathsf{def}}{=} (\bot, \bot)$ and $(\mathcal{T}_{2,i}, \mathcal{T}'_{2,i}) \overset{\mathsf{def}}{=} (\mathcal{S}_i, \mathcal{S}'_i)$.

Example 35. *All the rules applied in Example 34 are internal rules.*

Since our algorithm manipulates matrices of constraint systems, we extend the notion of symbolic equivalence accordingly. Given a matrix $\mathcal{M}$ having n rows and m columns, we denote by $\mathcal{M}_{i,j}$ the constraint system stored in the i^{th} row and j^{th} column, and we denote by $\Phi_{i,j}$ its associated frame.

Definition 13 (symbolic equivalence $\approx_s$)**.** *Let $\mathcal{M}$ and $\mathcal{M}'$ be two matrices of constraint systems having the same structure and of size $(n \times m)$ and $(n \times m')$ respectively. We have that $\mathcal{M} \subseteq_s \mathcal{M}'$ if for all $1 \le i \le n$, for all $1 \le j \le m$, for all $(\sigma, \theta) \in \mathsf{Sol}(\mathcal{M}_{i,j})$, there exists $1 \le k \le m'$ and a substitution σ' such that $(\sigma', \theta) \in \mathsf{Sol}(\mathcal{M}'_{i,k})$ and $\Phi_{i,j}\sigma \sim \Phi'_{i,k}\sigma'$.*

If $\mathcal{M} \subseteq_s \mathcal{M}'$ and $\mathcal{M}' \subseteq_s \mathcal{M}$, then we say that $\mathcal{M}$ and $\mathcal{M}'$ are in symbolic equivalence*, denoted by $\mathcal{M} \approx_s \mathcal{M}'$.*

In the following section, we will describe a quite complex strategy $\mathcal{S}$ that always terminates on sets of initial constraint systems. Before describing it, we state some soundness and completeness results, and we explain the test that is performed on the leaves to decide symbolic equivalence.

Our transformation rules yield a finite tree labeled with pairs of matrices of constraint systems. Actually, if we follow the strategy $\mathcal{S}$, we can show that our notion of equivalence is preserved through application of our transformation rules: for any transformation rule, we have that symbolic equivalence holds for the father if, and only if, symbolic equivalence holds for the children. Formally, we distinguish the case of an application of an internal rule from the one of an external rule.

Theorem 2. *[soundness and completeness for internal rules] Let $\mathcal{M}_1$, $\mathcal{M}'_1$ be two matrices of constraint systems obtained from a pair of sets of initial constraint systems by following the strategy $\mathcal{S}$. Let* $\textsc{Rule}(\tilde{p})$ *be an internal transformation rule applicable on $(\mathcal{M}_1, \mathcal{M}'_1)$ on the i^{th} row. Let $(\mathcal{M}_2, \mathcal{M}'_2)$ be the resulting pair of matrices of constraint systems obtained by the application of* $\textsc{Rule}(\tilde{p})$. *We have that:*

$$\mathcal{M}_2 \approx_s \mathcal{M}'_2 \text{ is equivalent to } \mathcal{M}_1 \approx_s \mathcal{M}'_1$$

Theorem 3. *[soundness and completeness for external rules] Let $\mathcal{M}$, $\mathcal{M}'$ be two matrices of constraint systems obtained from a pair of sets of initial constraint systems by following the strategy $\mathcal{S}$. Let* $\textsc{Rule}(\tilde{p})$ *be an external transformation rule applicable on $(\mathcal{M}, \mathcal{M}')$. Let $(\mathcal{M}_1, \mathcal{M}'_1)$ and $(\mathcal{M}_2, \mathcal{M}'_2)$ be the two resulting pairs of matrices of constraint systems obtained by the application of* $\textsc{Rule}(\tilde{p})$. *We have that:*

$$\mathcal{M}_1 \approx_s \mathcal{M}'_1 \text{ and } \mathcal{M}_2 \approx_s \mathcal{M}'_2 \text{ is equivalent to } \mathcal{M} \approx_s \mathcal{M}'$$

The proof of the soundness and completeness theorems stated above are done by a case analysis on the transformation rules. Basically, we assume the existence of a solution for a given constraint system (satisfying some properties due to our strategy $\mathcal{S}$), and we show how to transform this solution to obtain a solution for a slightly different constraint system (typically we consider a solution of a given constraint system and we have to show that, after application of a transformation rule, this solution still exists in one of its sons). In most cases, the transformation consists in replacing a recipe by another one that allows one to deduce the same message. The main issue of this replacement is to guarantee that the new recipe also satisfies all the needed properties (*e.g.,* the recipe has to be in Π_r, conformity of the recipe w.r.t. the frame, ...).

Example 36. *Consider the following constraint system $\mathcal{C}$.*

$$\mathcal{C} = \left\{ \begin{array}{l} ax_1, 1 \rhd a;\; ax_2, 2 \rhd \mathsf{senc}(a,a);\; ax_3, 3 \rhd a \\ X, 1 \vdash^? \mathsf{senc}(a,a);\;\; Y, 3 \vdash^? a \end{array} \right.$$

A solution of such a constraint system is $\theta = \{X \mapsto \mathsf{senc}(ax_1, ax_1); Y \mapsto \mathsf{sdec}(ax_2, ax_3)\}$. Note that the recipes are in Π_r and conform to the frame.

Applying the rule $\textsc{Eq-frame-ded}(ax_2, X)$, *the frame element $ax_2, 2 \rhd \mathsf{senc}(a,a)$ is added to the set* NoUse, *and thus now we have to replace ax_2 by $X\theta$ (note that both deduce the same term $\mathsf{senc}(a,a)$).*

In such a situation, $Y\theta$ will become $\mathsf{sdec}(\mathsf{senc}(ax_1, ax_1), ax_3)$ which is not a recipe in Π_r anymore. To cope with this problem, we have to replace ax_2 by $\mathsf{senc}(ax_1, ax_1)$, but also $Y\theta$ by ax_1. Thus, we get $\theta' = \{X \mapsto \mathsf{senc}(ax_1, ax_1);\; Y \mapsto ax_1\}$.

3.5. Test on leaves

By applying the rules on a pair of sets of initial constraint systems, we obtain a tree whose nodes (including the leaves) are labeled by a pair of matrices. The idea behind our transformation rules (given in Figure 1 and Figure 2) is to transform constraint systems into simpler ones, so that deciding symbolic equivalence will become trivial. Typically, as it is done in [32], we want to consider systems in which right-hand sides of deducibility constraints are distinct variables. However, in presence of disequations, putting the systems in such a form does not guarantee anymore that the two resulting systems will be in symbolic equivalence. Let us illustrate this using a simple example.

Example 37. *Consider the pair $(\mathcal{C}, \mathcal{C}')$ of sets of initial constraint systems given below (each set is reduced to a singleton):*

$$\mathcal{C} = \left\{ \ \Phi = \{ax_1, 1 \rhd a\}; \ \ X, 1 \vdash^? x \right. \qquad \mathcal{C}' = \left\{ \begin{array}{l} \Phi = \{ax_1, 1 \rhd a\}; \ \ X, 1 \vdash^? x \\ x \neq^? \langle a, a \rangle \end{array} \right.$$

Although these two systems have the expected form, they are not in symbolic equivalence. To see this, consider for instance the substitution $\theta = \{X \mapsto \langle ax_1, ax_1 \rangle\}$. We have that $\theta \in \mathsf{Sol}(\mathcal{C})$ but $\theta \notin \mathsf{Sol}(\mathcal{C}')$ due to the presence of the disequation.

Thus, once the system is put in this kind of "pre-solved form", the basic idea will be to continue to apply our transformation rules to "match" disequations of each constraint system. For this, we need to transform the disequations in which some names or universally quantified variables occur until obtaining disequations that only contain free variables and public function symbols. This will guarantee that there exists a recipe associated to this term and this gives us the way to match it in another constraint system. Once the system is transformed into such a new kind of "solved form" (*i.e.,* distinct variables on the right-hand side of deducibility constraints as well as matched disequations), we can now easily conclude. Indeed, since we also take care of static equivalence on the resulting frames, disequations that correspond to public disequality tests are easily transferable from one constraint system to another without any additional checks.

Example 38. *Continuing Example 37 and assuming that the pairing operator is the only constructor symbol, we will go on, applying* $\textsc{Cons}(X, \langle \ \rangle)$*. The resulting pair on the left branch will be the pair $(\mathcal{C}_1, \mathcal{C}_1')$ where:*

$$\mathcal{C}_1 = \left\{ \begin{array}{l} \Phi; \ \ X = \langle X_1, X_2 \rangle \\ X_1, 1 \vdash^? x_1; \ X_2, 1 \vdash^? x_2 \end{array} \right. \qquad \mathcal{C}_1' = \left\{ \begin{array}{l} \Phi; \ \ X = \langle X_1, X_2 \rangle \\ X_1, 1 \vdash^? x_1; \ X_2, 1 \vdash^? x_2 \\ x_1 \neq^? a \vee x_2 \neq^? a \end{array} \right.$$

Now, by applying the $\textsc{Axiom}$ *rule twice, the resulting pair on the left branch will be the pair $(\mathcal{C}_{11}, \mathcal{C}_{11}')$ where:*

$$\mathcal{C}_{11} = \{\Phi; \ \ X = \langle ax_1, ax_1 \rangle \qquad \mathcal{C}_{11}' = \Big\{\Phi; \ \ X = \langle ax_1, ax_1 \rangle; \ \ a \neq^? a \vee a \neq^? a$$

The disequations occurring in $\mathcal{C}_{11}'$ are trivially not satisfied, thus we have that $\mathcal{C}_{11}'\!\downarrow = \bot$. These two constraint systems are trivially not in symbolic equivalence.

As illustrated above, our goal is to reach pairs of (sets) of constraint systems in *solved form.* Each constraint system is either $\bot$ or satisfies the following conditions:

1. for all $X, i \vdash^? u \in D$, we have $X \in S_2$ and u is a variable distinct from the right-hand side of any other deducibility constraint;

2. the set E does not contain any variable that is universally quantified, and for all $u \neq^? v$ in E, we have that u, v do not contain any names. Moreover, the disequations are "the same" on each constraint system occurring in the pair.

The first phase of our strategy consists in applying transformation rules to fulfil the first condition (without taking care of the disequations) and obtain a system in *pre-solved* form, whereas the second phase of our strategy will reduce the constraint systems into *solved* form. The next section is dedicated to the description of the strategy $\mathcal{S}$ that has been designed with a lot of care to ensure the termination of our procedure.

Once solved forms are reached, the test performed on each leaf labeled $(\mathcal{M}, \mathcal{M}')$ consists of checking that for each row of the matrices, either both matrices contain a constraint system different form $\perp$, or both matrices contain only $\perp$ on the whole row. More formally, we have that:

Definition 14 (test $\mathsf{LeafTest}$)**.** *Let $(\mathcal{M}, \mathcal{M}')$ be a pair of matrices of constraint systems with n rows and m (resp. m') columns.* $\mathsf{LeafTest}(\mathcal{M}, \mathcal{M}') = \mathsf{true}$ *if and only if for each row $i \in \{1 \dots n\}$, we have that:*

$$\exists j \in \{1, \dots, m\} \text{ with } \mathcal{M}_{i,j} \neq \perp \text{ if, and only if, } \exists j' \in \{1, \dots, m'\} \text{ with } \mathcal{M}'_{i,j'} \neq \perp .$$

Theorem 4. *Let $(\mathcal{M}_0, \mathcal{M}'_0)$ be a pair of sets of initial constraint systems and $(\mathcal{M}, \mathcal{M}')$ be a leaf of the tree whose root is labeled with $(\mathcal{M}_0, \mathcal{M}'_0)$ and which is obtained following the strategy $\mathcal{S}$. We have that $\mathcal{M} \approx_s \mathcal{M}'$ if, and only if,* $\mathsf{LeafTest}(\mathcal{M}, \mathcal{M}') = \mathsf{true}$.

The proof of this theorem is done relying on the two following properties that are satisfied by each leaf $(\mathcal{M}, \mathcal{M}')$ of the tree.

1. Any constraint system $\mathcal{C}$ occurring in $\mathcal{M}$ (resp. $\mathcal{M}'$) different from $\perp$ admits a solution, *i.e.,* $\mathsf{Sol}(\mathcal{C}) \neq \emptyset$.
2. For any constraint systems $\mathcal{C}$, $\mathcal{C}'$ that occur in the same row (possibly of the same matrice) and that are different from $\perp$, we have that $\mathcal{C} \approx_s \mathcal{C}'$.

Corollary 1 (main result)**.** *Let $\mathcal{S}$ and $\mathcal{S}'$ be two sets of initial constraint systems. We have that $\mathcal{S} \approx_s \mathcal{S}'$ if, and only if,* $\mathsf{LeafTest}(\mathcal{M}, \mathcal{M}') = \mathsf{true}$ *for any leaf of the tree whose root is labeled with $(\mathcal{S}, \mathcal{S}')$ and which is obtained following the strategy $\mathcal{S}$.*

4. Strategy

Our goal is to reduce (matrices of) constraint systems to solved forms. We proceed in two steps: first reduce the (matrices of) constraint systems to pre-solved forms, i.e., taking care on deducibility constraints only and then reduce them to solved form, considering the disequality constraints. Let us consider the first step. We may restrict the rules applications, as long as there is always at least a rule that can be applied when (some) constraint system is not in pre-solved form. That is how we design our strategy; we restrict the rules applications, while ensuring that an unsolved contraint system can be reduced by at least one rule. This is what we later call "strong application". Such a strategy is terminating when applied to a single constraint system, and yields a pre-solved form.

We are however working on pairs of (matrices of) constraint systems. In that case, we must focus on one unsolved constraint system. Otherwise, since the same rule is applied on all the constraint systems of the pair, we may go back and forth and never reach a pair in which all the systems are in solved form. We will give an example of such a phenomenon: we could go back and forth forever, simplifying alternatively the left and the right component of a pair. That is why, we need to design a more elaborate strategy on pairs of matrices. It will, roughly, require to focus on one unsolved constraint system, and to ensure that the rule is a strong application for this system. Once this system is in pre-solved form, other rule applications (applied to put the other systems in pre-solved form) may not, as a side effet, impair this pre-solved form.

We say that a rule is *strongly applicable* on $\mathcal{C} = (S_1; S_2; \Phi; D; E; E_\Pi; ND; \mathsf{NoUse})$ when the following extra conditions are fulfilled:

- Rule $\textsc{Cons}(X, \mathsf{f})$: either the term t is not a variable, or there exists an atomic statement $(\mathsf{root}(X) \neq^? \mathsf{g})$ in E_Π such that $\mathsf{g} \in \mathcal{F}_c$ and $\mathsf{g} \neq \mathsf{f}$;
- Rule $\textsc{Axiom}(X, \mathsf{path})$: the term v is not a variable or there exists $\mathsf{f} \in \mathcal{F}_c$ such that $(\mathsf{root}(X) \neq^? \mathsf{f})$ in E_Π;
- Rule $\textsc{Dest}$: the term v is not a variable;
- Rule $\textsc{Ded-st}$: the term u is not a variable;
- Rule $\textsc{Eq-frame-frame}$: no additional condition;
- Rule $\textsc{Eq-frame-ded}$: the terms u_1, u_2 are the same variable.
- Rule $\textsc{Eq-ded-ded}$: $\xi \in \mathit{vars}^2(D)$, and u and v are the same variable.

Since we have to apply simultaneously our transformation rules on several constraint systems, we can not guarantee that each application will be a strong one. As illustrated by Example 39, this yields some termination issues.

Example 39. *Consider the pair $(\mathcal{C}, \mathcal{C}')$ of sets of initial constraint systems given below (each set is actually reduced to a singleton):*

$$\mathcal{C} = \{\ \Phi;\ X, 1 \vdash^? \mathsf{senc}(x_1, x_2);\ Y, 2 \vdash^? x_1 \qquad \mathcal{C}' = \{\ \Phi;\ X, 1 \vdash^? y_1;\ Y, 2 \vdash^? \mathsf{senc}(y_1, y_2)$$

We may apply $\textsc{Cons}(X, \mathsf{senc})$ on the system $\mathcal{C}$ yielding (on the left branch):

$$\mathcal{C}_1 = \begin{cases} \Phi;\ X =^? \mathsf{senc}(X_1, X_2) \\ X_1, 1 \vdash^? x_1 \\ X_2, 1 \vdash^? x_2 \\ Y, 2 \vdash^? x_1 \end{cases} \qquad \mathcal{C}'_1 = \begin{cases} \Phi;\ X =^? \mathsf{senc}(X_1, X_2) \\ X_1, 1 \vdash^? z_1; \\ X_2, 1 \vdash^? z_2 \\ Y, 2 \vdash^? \mathsf{senc}(\mathsf{senc}(z_1, z_2), y_2) \end{cases}$$

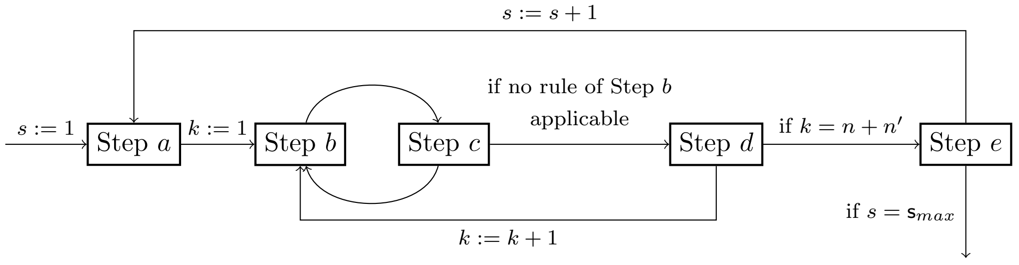
Then, again using a strong application of $\textsc{Cons}(Y, \mathsf{senc})$ on the system $\mathcal{C}'_1$, we obtain (on the left branch):

$$\mathcal{C}_{11} = \begin{cases} \Phi;\ X =^? \mathsf{senc}(X_1, X_2) \\ X_1, 1 \vdash^? \mathsf{senc}(x_{11}, x_{12}) \\ X_2, 1 \vdash^? x_2 \\ Y =^? \mathsf{senc}(Y_1, Y_2) \\ Y_1, 2 \vdash^? x_{11} \\ Y_2, 2 \vdash^? x_{12} \end{cases} \qquad \mathcal{C}'_{11} = \begin{cases} \Phi;\ X =^? \mathsf{senc}(X_1, X_2) \\ X_1, 1 \vdash^? z_1 \\ X_2, 1 \vdash^? z_2 \\ Y =^? \mathsf{senc}(Y_1, Y_2) \\ Y_1, 2 \vdash^? \mathsf{senc}(z_1, z_2) \\ Y_2, 2 \vdash^? y_2 \end{cases}$$

Thus, we get back to a subproblem of the original deducibility constraints.

4.1. Taking care of deducibility constraints

The first phase of our strategy consists of applying transformation rules to put constraint systems in "pre-solved" form. As depicted below, this first phase is a cycle of several steps.



The integer s indicates the *support of the rules* that are applied during the cycle. This notion of *support of a rule* is formally defined as follows:

- the support of $\textsc{Cons}(X, \mathsf{f})$ (resp. $\textsc{Axiom}(X, \mathsf{path})$, $\textsc{Eq-ded-ded}(X, \xi)$) is i where $X, i \vdash^? u \in D$;
- the support of $\textsc{Dest}(\xi, \ell \to r, i)$ (resp. $\textsc{Ded-st}(X, \xi)$) is i (resp. s_{max} *i.e.,* the maximal index that occurs in $\mathcal{C}$);
- the support of $\textsc{Eq-frame-frame}(\xi_1, \xi_2)$ is $\max(i_1, i_2)$ where $\xi_1, i_1 \rhd u_1 \in \Phi$ and $\xi_2, i_2 \rhd u_2 \in \Phi$;
- the support of $\textsc{Eq-frame-ded}(\xi_1, X_2)$ is i_1 where $\xi_1, i_1 \rhd u_1 \in \Phi$;

The integer n (resp. n') is the number of columns in matrix $\mathcal{M}$ (resp. $\mathcal{M}'$) and m is the size of the frames that occur in $\mathcal{M}$ and $\mathcal{M}'$. The integer k indicates the column of the matrix on which we are currently working. By convention, when $k > n$, *i.e.,* k is strictly greater than the number of columns in the matrix $\mathcal{M}$, this means that we work on the $(k - n)^{\text{th}}$ column of the matrix $\mathcal{M}'$.
We now explain in more detail each of these steps.

Step a: frame analysis. We apply the rules $\textsc{Dest}$ and $\textsc{Eq-frame-ded}$, with support equal to s, as long as possible with priority on the rule $\textsc{Eq-frame-ded}$. The application of those rules has to be a strong application for at least one constraint system that occurred in the row of the matrix on which we apply the rule.

The main idea is to work on the frame to learn the deducible subterms (this is the purpose of the rule $\textsc{Dest}$). However, when we encounter a frame element of the form $\xi, i \rhd x$, we can not apply the $\textsc{Dest}$ rule on it. The purpose of using the rule $\textsc{Eq-frame-ded}$ is to "discard" this frame element by adding it into the set NoUse. To ensure that rule $\textsc{Eq-frame-ded}$ will be applicable each time we are in such a situation, it is important to work by increasing support. Indeed, by definition of constraint system, we know that the variable x will appear in a deducibility constraint of support less than i. Putting deducibility constraints of support less than i in pre-solved form allows us to ensure that there exists $X, j \vdash^? x$ with $j < i$, and thus $\textsc{Eq-frame-ded}$ is applicable.

In order to satisfy some necessary properties, when $\textsc{Dest}(\tilde{p})$ or $\textsc{Eq-frame-ded}(\tilde{p})$ is applied on one row of the matrix, we will apply the same rule with similar parameter on each row of the matrix. More specifically,

- if $\textsc{Dest}(\xi, \ell \to r, s)$ is applied on a row $(\mathcal{S}, \mathcal{S}')$ of $(\mathcal{M}, \mathcal{M}')$ where (ξ, i) (with $i \leq s$) belongs to the structure of the constraint systems in $(\mathcal{S}, \mathcal{S}')$, then for each row $(\mathcal{T}, \mathcal{T}')$ of $(\mathcal{M}, \mathcal{M}')$ where there exists ξ' such that (ξ', i) belongs to the structure of the constraint systems in $(\mathcal{T}, \mathcal{T}')$ and $\mathsf{path}(\xi') = \mathsf{path}(\xi)$, we also apply $\textsc{Dest}(\xi', \ell \to r, s)$ on $(\mathcal{T}, \mathcal{T}')$.

- if $\textsc{Eq-frame-ded}(\xi, X)$ is applied on a row $(\mathcal{S}, \mathcal{S}')$ of $(\mathcal{M}, \mathcal{M}')$ where (ξ, i) belongs to the structure of the constraint systems in $(\mathcal{S}, \mathcal{S}')$, then for each row $(\mathcal{T}, \mathcal{T}')$ of $(\mathcal{M}, \mathcal{M}')$ where there exist ξ' such that (ξ', i) belongs to the structure of the constraint systems in $(\mathcal{T}, \mathcal{T}')$ and $\mathsf{path}(\xi') = \mathsf{path}(\xi)$, we apply $\textsc{Eq-frame-ded}(\xi', X)$ on $(\mathcal{T}, \mathcal{T}')$.

At the end of Step a, the frame of any constraint system is fixed for the support s. The existing frame elements (for support s) could be further instantiated, but no frame element will be added (for this support).

To avoid the non-terminating behaviour mentioned in Example 39, we break the symmetry between the different components. The idea is to focus on one column of the matrix and to reduce the constraint systems until reaching “pre-solved form” (distinct variables on the right-hand sides of the deducibility constraints), and then move to the next column of the matrix.

Example 40. *Going back to Example 39, and assuming that $\Phi = \{ax_1, 1 \rhd a;\ ax_2, 2 \rhd b\}$, we can first observe that there is nothing to do regarding Step a. The idea will be to apply $\textsc{Cons}(X, \mathsf{senc})$ as in Example 39, but then we will be forced to work on the constraint system $\mathcal{C}_1$. We can apply $\textsc{Eq-ded-ded}(Y, X_1)$ yielding (on the left branch) the pair $(\mathcal{C}_{11}, \mathcal{C}'_{11})$ where:*

$$\mathcal{C}_{11} = \left\{ \begin{array}{c} \Phi;\ X_1, 1 \vdash^? x_1;\ X_2, 2 \vdash^? x_2 \\ X =^? \mathsf{senc}(X_1, X_2) \wedge Y =^? X_1 \end{array} \right.$$

and $\mathcal{C}'_{11} = \bot$. Indeed, the equality $z_1 =^? \mathsf{senc}(\mathsf{senc}(z_1, z_2), y_2)$ can not be satisfied.

More formally, we have a cycle of three different steps. The parameter of this cycle is the index of the column on which we are currently working. Each of this cycle alternates Step b and Step c, and then ends with Step d.

Steps b and c: dealing with internal deducibility constraints. The purpose of this cycle (Steps *b* and *c*) is to deal with internal deducibility constraints (of support s), *i.e.,* the constraints of the form $X, s \vdash^? u$ with $X \not\in S_2$. During Step *b*, the idea is to put internal deducibility constraints in “pre-solved” form, whereas during Step *c*, the main goal is to remove them. At the end of Step *c*, all the internal deducibility constraints would have disappeared.

Step b. We apply the internal applications of the rules Eq-ded-ded, Eq-frame-frame, Cons, Axiom, and Ded-st with support less than s, as long as possible. To be applied on *e.g.,* the i^{th} row, the application of the rule has to correspond to a strong application w.r.t. the constraint system located at the i^{th} row and k^{th} column.

Step c. Given a constraint system $\mathcal{C} = (S_1; S_2; \Phi; D; E; E_\Pi; ND; \mathsf{NoUse})$, we consider the set $\mathsf{X}_\mathcal{C}$ defined as follows:

$$\mathsf{X}_\mathcal{C} = \{x \in \mathcal{X}^1 \mid (Y, j \vdash^? x) \in D, \text{ and } Y \notin S_2\}$$

The purpose of this set is to contain all the first-order variables that occur on the right-hand side of an internal deducibility constraint. We apply by order of preference:

1. The internal rule Eq-ded-ded(X, Y) with $X \notin S_2$ when the rule is strongly applicable on one constraint system of the k^{th} column.
2. The external rule Cons(X, f) when a variable in $\mathsf{X}_\mathcal{C}$ occurs in the deducibility constraint $X, i \vdash^? u$. This has again to correspond to a strong application.
3. The external rule Axiom(X, path) on $X, i \vdash^? u$ if this correspond to a strong application and i is minimal.

Intuitively, the purpose of Step c is to discard the internal deducibility constraints using the rule Eq-ded-ded. However, when this is not possible (*e.g.,* because the variable that occurs on the right-hand side of the internal constraint does not appear as a right member of an external deducibility constraint), we will try break the term u that contains such a variable using the rule Cons with the hope to be able to apply Eq-ded-ded once the variable will appear at the root position. Once this is done, and if an application of Eq-ded-ded is still not possible, we will use the rule Axiom to instantiate some variables. Doing this at the end allows us to ensure that the variables that will be introduced during the replacement will be "smaller" (*i.e.,* the deducibility constraints that introduce each of these variables have a support smaller than the variables that are removed thanks to the Axiom rule).

Step d: dealing with external deducibility constraints. Now, we have to put the external deducibility constraints in "pre-solved" form. For this, we apply the external application of the rules Eq-ded-ded, Cons and Axiom as long as they are strongly applicable on the constraint system $\mathcal{M}_{i,k}$ (or $\mathcal{M}_{i,k-n}$ when $k > n$) by increasing order on the index i of the row. For instance, if Rule$_1(\tilde{p}_1)$ is strongly applicable on $\mathcal{M}_{i_1,k}$, Rule$_2(\tilde{p}_2)$ is strongly applicable on $\mathcal{M}_{i_2,k}$, and $i_1 \leq i_2$ then we apply the rule Rule$_1(\tilde{p}_1)$ on $(\mathcal{M}, \mathcal{M}')$.

Step e: solving non-deducibility constraints. This last step consists of solving the non-deducibility constraints that occur in the matrices. This is done by replacing some constraint systems with $\perp$. Intuitively, we only keep the constraint systems that have a frame which is "maximal" (*i.e.,* a frame which contains a maximal number of elements).

Formally, for each constraint system $\mathcal{C}$ in the matrix (with its associated frame Φ), if there exists a constraint system $\mathcal{C}'$ (with its associated frame Φ') in the same column as $\mathcal{C}$, a recipe ξ, such that $(\xi, s \rhd u) \in \Phi'$ for some u, whereas Φ does not have such an element – *i.e.,* for all $(\xi', s \rhd v) \in \Phi$, we have that $\mathsf{path}(\xi) \neq \mathsf{path}(\xi')$ – then we replace the constraint system $\mathcal{C}$ in the matrix by $\perp$.

The fact that $\mathcal{C}$ does not contain a frame element for $\mathsf{path}(\xi)$ means that $\mathcal{C}$ contains some non-deducibility constraints instead. However, since $\mathcal{C}'$ is in pre-solved form, we know that the deducibility constraints introduced by the DEST rule have been solved, and thus the non-deducibility constraints in $\mathcal{C}'$ can not be satisfied. During Step e, each constraint system that is replaced by $\bot$ does not have a solution due to the non-deducibility constraints. The same applies to solve the non-deducibility constraints introduced by the rule DED-ST. We illustrate this through an example.

Example 41. *Consider the constraint system $\mathcal{C}$ presented in Example 33 as well as the constraint system $\mathcal{C}'$ made of the deducibility constraint $X, 2 \vdash^? \mathsf{senc}(a, a)$ and the frame $\Phi' = \{ax_1, 1 \rhd a;\ ax_2, 2 \rhd b\}$. We consider the matrices $\mathcal{M} = [\mathcal{C}]$ and $\mathcal{M}' = [\mathcal{C}']$. In Example 33, we have seen that applying the rule* DEST$(ax_2, \mathsf{sdec}(\mathsf{senc}(x, y), y) \to x, 2)$ *on $\mathcal{C}$ yields $\mathcal{C}_1$ and $\mathcal{C}_2$. The application of this rule on $\mathcal{C}'$ yields after normalisation the constraint system $\bot$ and the following constraint system $\mathcal{C}'_2$:*

$$\mathcal{C}'_2 = \left\{ \begin{array}{l} \Phi';\quad X, 2 \vdash^? \mathsf{senc}(a, a) \\ \forall x_1, x_2.[\mathsf{senc}(x_1, x_2) \neq^? b \vee 2 \not\vdash^? x_2] \end{array} \right.$$

Thus the application of the rule DEST$(ax_2, \mathsf{sdec}(\mathsf{senc}(x, y), y) \to x, 2)$ *on $\mathcal{M}$ and $\mathcal{M}'$ yields the pair of matrices $(\mathcal{M}_1, \mathcal{M}'_1)$:*

$$\mathcal{M}_1 = \begin{bmatrix} \mathcal{C}_1 \\ \mathcal{C}_2 \end{bmatrix} \qquad \mathcal{M}'_1 = \begin{bmatrix} \bot \\ \mathcal{C}'_2 \end{bmatrix}$$

Then, we may apply the rule AXIOM(Y, ax_1) *internally (on the first row of the matrix), and we obtain:*

$$\mathcal{M}_2 = \begin{bmatrix} \mathcal{C}_3 \\ \mathcal{C}_4 \\ \mathcal{C}_2 \end{bmatrix} \qquad \mathcal{M}'_2 = \begin{bmatrix} \bot \\ \bot \\ \mathcal{C}'_2 \end{bmatrix}$$

On the constraint system $\mathcal{C}_4$, the successive applications of CONS(Y, f) *and* AXIOM(Y, path) *for any f and path will yield after normalisation the constraint system $\bot$. All these rules are internal, and thus we obtain the following pair of matrices:*

$$\mathcal{M}_3 = \begin{bmatrix} \mathcal{C}_3 \\ \bot \\ \dots \\ \bot \\ \mathcal{C}_2 \end{bmatrix} \qquad \mathcal{M}'_3 = \begin{bmatrix} \bot \\ \bot \\ \dots \\ \bot \\ \mathcal{C}'_2 \end{bmatrix}$$

Then, the successive applications of the rules CONS *and* AXIOM *on $\mathcal{M}_3$ and $\mathcal{M}'_3$ (to solve the remaining deducibility constraints in $\mathcal{C}_3$, $\mathcal{C}_2$, and $\mathcal{C}'_2$) will yield in particular a leaf $(\mathcal{M}_4, \mathcal{M}'_4)$ where:*

$$\mathcal{M}_4 = \begin{bmatrix} \mathcal{C}_5 \\ \bot \\ \dots \\ \bot \\ \mathcal{C}_6 \end{bmatrix} \qquad \mathcal{M}'_4 = \begin{bmatrix} \bot \\ \bot \\ \dots \\ \bot \\ \mathcal{C}'_6 \end{bmatrix}$$

where the constraint systems $\mathcal{C}_5$, $\mathcal{C}_6$ and $\mathcal{C}'_6$ are as follows:

$$\mathcal{C}_5 = \left\{ \begin{array}{l} \Phi,\ \mathsf{sdec}(ax_2, ax_1), 2 \rhd b \\ X =^? \mathsf{senc}(ax_1, ax_1) \\ Y =^? ax_1 \end{array} \right. \qquad \begin{array}{l} \mathcal{C}_6 = \left\{ \begin{array}{l} \Phi;\ \ X =^? \mathsf{senc}(ax_1, ax_1) \\ \forall x_1, x_2 . [\mathsf{senc}(x_1, x_2) \neq^? \mathsf{senc}(b, a) \vee 2 \nvdash^? x_2] \end{array} \right. \\ \\ \mathcal{C}'_6 = \left\{ \begin{array}{l} \Phi';\ \ X =^? \mathsf{senc}(ax_1, ax_1) \\ \forall x_1, x_2 . [\mathsf{senc}(x_1, x_2) \neq^? b \vee 2 \nvdash^? x_2] \end{array} \right. \end{array}$$

Now, following the transformation explained above, the system $\mathcal{C}_6$ will be replaced by $\bot$. Indeed, there exists $\mathcal{C}_5$ in the same column as $\mathcal{C}_6$ that contains the frame element $\mathsf{sdec}(ax_2, ax_1), 2 \rhd b$*, and for which there is no counterpart in $\mathcal{C}_6$. Instead, in $\mathcal{C}_6$, we have a non-deducibility constraint that is actually unsatisfiable.*

Indeed, the existence of a solution for the constraint system $\mathcal{C}_5$ implies that the recipe $\mathsf{sdec}(ax_2, ax_1)$ *yields a message, thus ax_2 is a ciphertext whose key is deducible at stage 2, and so the non-deducibility constraint of $\mathcal{C}_6$ can not be satisfied.*

4.2. Taking care of disequations

After the first phase of our strategy $\mathcal{S}$, the rules Dest, Eq-frame-ded, Eq-frame-frame and Ded-st will never be applicable anymore for any parameter. Thus, the only rules that can be applied during the second phase are Cons, Axiom and Eq-ded-ded. Furthermore these rules will always be applied as external rules. As already explained, the purpose of this phase is to take care of the disequations. For this, we need to match them, and ensure that the same disequations occur in each constraint system. As depicted below, this second phase is made up of three steps.

$$\begin{tikzcd} {} \arrow[r] & \boxed{\text{Step } a} \arrow[r] & \boxed{\text{Step } b} \arrow[r, bend left] & \boxed{\text{Step } c} \arrow[l, bend left] \arrow[r, "\text{if no rule of Step } b \text{ is applicable}"] & {} \end{tikzcd}$$

Step a: getting rid of universally quantified variables. In order to be able to match the disequations, we have to get rid of variables that are universally quantified. For this, we apply the rules $\textsc{Cons}(X, \mathsf{f})$ and $\textsc{Axiom}(X, \mathsf{path})$ as long as, for at least one constraint system occurring in the matrix, this corresponds to a strong application of the rule or there exists an atomic statement $u \neq^? w$ in E (where u is such that $X, i \vdash^? u$ – actually at this stage u will be a variable) for which there exists a variable $y \in vars^1(w)$ which is universally quantified. At the end of this Step a, variables that are universally quantified would have been removed.

Example 42. *Let* $\Phi^+ = \{ax_1, 1 \rhd a;\ ax_2, 2 \rhd \langle b, a\rangle;\ \mathsf{proj}_1(ax_2), 2 \rhd b;\ \mathsf{proj}_2(ax_2) \rhd a\}$*, and consider the following constraint system:*

$$\mathcal{C} = \{\ \Phi^+;\ \ Y, 1 \vdash^? y;\ \ \forall x.\ y \neq^? \langle x, a\rangle$$

For sake of simplicity, we will assume that $\langle\ \rangle$ is the only constructor symbol. In order to get rid of the variable x, the strategy will tell us to apply $\textsc{Cons}(Y, \langle\ \rangle)$*. This gives us:*

$$\mathcal{C}_1 = \left\{ \begin{array}{l} \Phi^+;\ Y_1, 1 \vdash^? y_1;\ Y_2, 1 \vdash^? y_2 \\ \forall x.\ y \neq^? \langle x, a\rangle \\ y =^? \langle y_1, y_2\rangle;\ Y =^? \langle Y_1, Y_2\rangle \end{array} \right. \qquad \mathcal{C}_2 = \left\{ \begin{array}{l} \Phi^+;\ Y, 1 \vdash^? y \\ \forall x.\ y \neq^? \langle x, a\rangle \\ \mathsf{root}(Y) \neq^? \langle\ \rangle \end{array} \right.$$

Using our simplification rules, the first system will be simplified as follows:

$$\mathcal{C}_1\!\downarrow = \begin{cases} \Phi^+;\ Y_1, 1 \vdash^? y_1;\ Y_2, 1 \vdash^? y_2 \\ y_2 \neq^? a \\ y =^? \langle y_1, y_2\rangle;\ Y =^? \langle Y_1, Y_2\rangle \end{cases}$$

Note that $\mathcal{C}_1\!\downarrow$ does not contain any quantified variable anymore. Considering the constraint system $\mathcal{C}_2$, in order to get rid of the variable x, we can for instance apply the rule Axiom(Y, ax_1)*. We obtain (after some simplifications):*

$$\mathcal{C}_{21} = \left\{\ \Phi^+;\ Y = ax_1;\ y = a \right. \qquad \mathcal{C}_{22} = \begin{cases} \Phi^+; Y, 1 \vdash^? y;\ \forall x.\ y \neq^? \langle x, a\rangle \\ \mathsf{root}(Y) \neq^? \langle\ \rangle \wedge Y \neq^? ax_1 \end{cases}$$

The system $\mathcal{C}_{21}$ does not contain any quantified variable anymore. We can pursue like this using Cons(Y, ax_2) *on $\mathcal{C}_{22}$. We obtain (after some simplifications) $\mathcal{C}_{221} = \bot$ and*

$$\mathcal{C}_{222} = \begin{cases} \Phi^+; Y, 1 \vdash^? y;\ \forall x.\ y \neq^? \langle x, a\rangle \\ \mathsf{root}(Y) \neq^? \langle\ \rangle \wedge Y \neq^? ax_1 \wedge Y \neq^? ax_2 \end{cases}$$

We can continue with Cons$(Y, \mathsf{proj}_1(ax_2))$ *and* Cons$(Y, \mathsf{proj}_2(ax_2))$*. The resulting systems on the left branches will not contain any disequations (they are actually trivially satisfied) whereas on the right branch the constraint system will be turned to $\bot$ following the simplification rule given in Figure 4. Thus, at the end, all the quantified variables have been removed.*

Steps b and c: matching disequations. To ensure that we will reach a solved form in which all the disequations are matched, the rule Eq-ded-ded plays an important role. The rule Eq-ded-ded allows one to "externalise" the disjunctions, splitting disjunctive disequations, each of which will appear in different matrices. However, it may happen that the rule Eq-ded-ded can not be applied to get rid of a particular disequation. In such a situation, we will first use the rules Cons and Axiom to simplify it, and allow eventually the application of the rule Eq-ded-ded. The only rules that can be applied during these two steps (b and c) are Cons, Axiom and Eq-ded-ded. However, as illustrated by Example 43, to ensure termination we can not apply them in any order.

Example 43. *We consider a constraint system in "pre-solved" form such that:*

$$E = [x_1 \neq^? y \vee x_2 \neq^? a] \wedge y \neq^? \langle\langle x_1, x_2\rangle, b\rangle.$$

For sake of simplicity, we do not described Φ and D. We simply assume that the frame contains the terms a and b. First, we apply Axiom *on x_2 (with a), on one branch we will obtain $x_1 \neq^? y \ \wedge\ y \neq^? \langle\langle x_1, a\rangle, b\rangle$. Then applying* Cons *twice, we obtain $x_1 \neq^? \langle\langle y_1, y_2\rangle, y_3\rangle \wedge [y_1 \neq^? x_1 \vee y_2 \neq^? a \vee y_3 \neq^? b]$. Lastly, applying* Axiom *on y_3 (with b), we obtain:*

$$x_1 \neq^? \langle\langle y_1, y_2\rangle, b\rangle \wedge [y_1 \neq^? x_1 \vee y_2 \neq^? a]$$

getting back to the original set of disequations.

To avoid such a situation, the main idea is to postpone the use of the Axiom rule. We apply as long as we can the rules Cons and Eq-ded-ded, and only after that we can move to Step c and apply the Axiom rule.

Example 44. *Going back to Example 43 and following our strategy, we will first apply* EQ-DED-DED *to deal with the disequation* $x_1 \neq^? y$*. On the left branch,* i.e., *assuming the equality* $x_1 = y$ *is satisfied, the disequation* $y \neq^? \langle\langle x_1, x_2\rangle, b\rangle$ *becomes* $y \neq^? \langle\langle y, x_2\rangle, b\rangle$*, and disappears since it is trivially satisfied. Hence, we obtain two constraint systems that respectively contains:*

$$x_2 \neq^? a \qquad\qquad x_1 \neq^? y \wedge y \neq^? \langle\langle x_1, x_2\rangle, b\rangle$$

Then, on the resulting system on the left branch, we have no choice, we have to apply an AXIOM *rule. On the right, we pursue using the* CONS *rule on* y *allowing us to simplify (on the left branch) a bit more the disequations – the name* b *is now at the root position:*

$$x_1 \neq^? \langle y_1, y_2\rangle \wedge [y_1 \neq^? \langle x_1, x_2\rangle \vee y_2 \neq^? b]$$

Then, we may apply EQ-DED-DED *to deal with* $y_1 \neq^? \langle x_1, x_2\rangle$*. We get*

$$y_2 \neq^? b \qquad\qquad x_1 \neq^? \langle y_1, y_2\rangle \wedge y_1 \neq^? \langle x_1, x_2\rangle$$

Again, on the left, the disequation $x_1 \neq^? \langle y_1, y_2\rangle$ *has disappeared since after replacing* y_1 *with* $\langle x_1, x_2\rangle$*, it is trivially satisfied. On the right, the disequations now contain free variables and public function symbols and applying* EQ-DED-DED *will be useless. On the left, we now have to apply an instance of the* AXIOM *rule. Thus, this strategy avoids the non termination issue mentioned in the previous example.*

Step b. During this step, we apply as long as we can the rules CONS and EQ-DED-DED. However, as illustrated with the following example, due to the fact that we have to apply simultaneously our transformation rules on several constraint systems, we can get some termination troubles.

Example 45. *Consider the pair* $(\mathcal{C}, \mathcal{C}')$ *of sets of initial constraint systems given below (each set is actually reduced to a singleton):*

$$\mathcal{C} = \left\{ \begin{array}{l} ax_1, 1 \rhd a \\ X, 1 \vdash^? x;\ Y, 1 \vdash^? y \\ x \neq^? \mathsf{h}(y) \wedge x \neq^? y \end{array} \right. \qquad \mathcal{C}' = \left\{ \begin{array}{l} ax_1, 1 \rhd a \\ X, 1 \vdash^? x;\ Y, 1 \vdash^? y \end{array} \right.$$

We could apply CONS(X, h) *replacing* x *with* $\mathsf{h}(x')$ *to simplify the disequation* $x \neq^? \mathsf{h}(y)$ *into* $x' \neq^? y$*. However, this operation will transform the other disequation, namely* $x \neq^? y$ *into* $\mathsf{h}(x') \neq^? y$*. More precisely, this gives (on the left branch):*

$$\mathcal{C}_0 = \left\{ \begin{array}{l} ax_1, 1 \rhd a \\ X', 1 \vdash^? x';\ Y, 1 \vdash^? y \\ x' \neq^? y \wedge \mathsf{h}(x') \neq^? y \\ X =^? \mathsf{h}(X') \end{array} \right. \qquad \mathcal{C}'_0 = \left\{ \begin{array}{l} ax_1, 1 \rhd a \\ X', 1 \vdash^? x';\ Y, 1 \vdash^? y \\ X =^? \mathsf{h}(X') \end{array} \right.$$

This pair $(\mathcal{C}_0, \mathcal{C}'_0)$ *is made up of two systems on which the* CONS *rule is again applicable, and we can go on forever with* CONS.

The main idea is to favour the application of EQ-DED-DED. However, given a particular disequation, necessarily of the form $x \neq^? u$ at this stage, it may happen that EQ-DED-DED can not be applied for two main reasons:

1. either a name occurred in the disequation, *i.e.,* u contains a name;

2. or a "faulty" variable occurred in the disequation, *i.e.,* u contains a variable whose support is greater than the support of x.

In both cases, the idea is to apply the CONS rule to bring the name or the "faulty" variable at the root position. Thus, we authorise the application of the rule $\textsc{Cons}(X, \mathsf{f})$ on $X, i_0 \vdash^? t$ during Step b (actually t is a variable at this stage) if there exists a constraint system $\mathcal{C}$ on which the rule $\textsc{Cons}(X, \mathsf{f})$ is not useless, and such that:

1. either $\textsc{Cons}(X, \mathsf{f})$ is strongly applicable on $\mathcal{C}$ (at this stage, since t is a variable, this means that there exists an atomic statement $(\mathsf{root}(X) \neq \mathsf{g})$ in $E_\Pi(\mathcal{C})$ such that $\mathsf{g} \in \mathcal{F}_c$ and $\mathsf{g} \neq \mathsf{f}$);

2. or there is a disequation of the form $t \neq u$ with $\mathsf{root}(u) = \mathsf{f}$, and u contains a name or a "faulty" variable, *i.e.,* $i_0 < \max \big\{ i \mid x \in \mathit{vars}^1(u) \text{ and } (X, i \vdash^? x) \in D(\mathcal{C}) \big\}$.

Example 46. *Going back to Example 45. Applying* $\textsc{Cons}(X, \mathsf{h})$ *is now forbidden. Instead, we may apply* $\textsc{Eq-ded-ded}(X, \mathsf{h}(Y))$*. This leads us to the pairs* $(\bot; \mathcal{C}'_1)$ *and* $(\mathcal{C}; \mathcal{C}'_2)$ *where:*

$$\mathcal{C}'_1 = \begin{cases} ax_1, 1 \rhd a \\ Y, 1 \vdash^? y \\ x =^? \mathsf{h}(y);\ X =^? \mathsf{h}(Y) \end{cases} \qquad \mathcal{C}'_2 = \begin{cases} ax_1, 1 \rhd a \\ X, 1 \vdash^? x;\ Y, 1 \vdash^? y \\ x \neq^? \mathsf{h}(y) \end{cases}$$

From the pair $(\bot; \mathcal{C}'_1)$ *we will conclude that symbolic equivalence does not hold. Regarding the pair* $(\mathcal{C}; \mathcal{C}'_2)$*, we can go on and reach a solved form by applying* $\textsc{Eq-ded-ded}(X, Y)$ *obtaining again two pairs of constraint systems. The first one will be of the form* $(\bot; \mathcal{C}'_3)$ *and the second one will contain two systems in which all the disequations are matched.*

We have shown that applying the rules CONS (under the additional conditions mentioned above) and EQ-DED-DED in any order will terminate (for Step b). However, to ensure termination of the cycle made of Step b and Step c, we have to work on a disequation which is *maximal*, *i.e.* one that involves variables whose supports are maximal. The necessity of this extra condition and its formal definition will be discussed later on (actually after the description of Step c).

Step c. In this last step, we apply the rule $\textsc{Axiom}(X, \mathsf{path})$ as long as possible, *i.e.,* as long as there is at least one constraint system $\mathcal{C}$ in the pair of matrices on which $\textsc{Axiom}(X, \mathsf{path})$ is strongly applicable on it. Note that, at this stage, the term t in the constraint $X, i \vdash^? t$ is necessarily a variable. Thus a strong application means that $(\mathsf{root}(X) \neq \mathsf{f}) \in E_\Pi(\mathcal{C})$ for some f. When no more instance of the AXIOM rule can be applied, we go back to Step b. It is quite easy to see that Step c alone will terminate. The number of variables decreases on the left branch, and at some point all the possible instances of the AXIOM rule would have been considered.

However, to ensure termination of the cycle made of Step b and Step c, we have to restrict the order on which the rules CONS and EQ-DED-DED are applied during Step b. We prove termination of this cycle under the hypothesis that we always work on the maximal disequation. The measure associated to a disequation $u \neq^? v$ occurring in a constraint system $\mathcal{C}$ is a pair of integers defined by $\mathcal{L}^1_\mathcal{C}(u \neq^? v) = (\mathcal{L}^1_\mathcal{C}(u); \mathcal{L}^1_\mathcal{C}(v))$ where:

$$\mathcal{L}^1_\mathcal{C}(u) = \mathit{max}\ \big(\{ i \mid (X, i \vdash^? x) \in D(\mathcal{C}) \text{ and } x \in \mathit{vars}(u)\} \cup \{0\}\big).$$

We use a "lexicographic" order to compare those pairs, and to decide on which disequation we will work. We have that $(i_1, i_2) >_{\mathsf{lex}} (j_1, j_2)$ if

- either $max(i_1, i_2) > max(j_1, j_2)$;
- or $max(i_1, i_2) = max(j_1, j_2)$ and $min(i_1, i_2) > min(j_1, j_2)$

Note that, using this order, we have that $\mathcal{L}^1_{\mathcal{C}}(u \neq^? v) = \mathcal{L}^1_{\mathcal{C}}(v \neq^? u)$ for any terms u and v.

Example 47. *Let $(\mathcal{C}, \mathcal{C}')$ be two constraint systems obtained at the end of Phase 2/ Step a that only differ by the content of their frame. For sake of simplicity, we also assume that f is a function symbol of arity 2 and the only one that we consider here. (this symbol could be mimicked in our setting using h and $\langle\ ,\ \rangle$). Moreover, regarding disequations, we assume that they contain:*

$$x \neq \mathsf{f}(y, z) \qquad x \neq z \qquad \mathsf{root}(Y) \neq \mathsf{f}.$$

Note that Cons *and* Eq-ded-ded *can not be applied, but due to the presence of $\mathsf{root}(Y) \neq \mathsf{f}$, the rule* Axiom *is strongly applicable on Y. Consider the left branch during such an application, and assume that such an application will instantiate y with $\mathsf{f}(\mathsf{f}(a, a), w)$ on $\mathcal{C}$, and y with $\mathsf{f}(w, \mathsf{f}(a, a))$ on $\mathcal{C}'$. Let $(\mathcal{C}_1, \mathcal{C}'_1)$ be the resulting pair. Such a scenario is possible since even if $\mathcal{C}$ and $\mathcal{C}'$ have the same structure, they may differ on the content of their frame. The constraint systems $\mathcal{C}_1$ and $\mathcal{C}'_1$ will now have different sets of disequations. In particular, we have that*

$$\mathcal{C}_1 = \left\{ \begin{array}{c} \dots \\ x \neq^? \mathsf{f}(\mathsf{f}(\mathsf{f}(a, a), w), z) \\ x \neq^? z \end{array} \right. \qquad \mathcal{C}_1 = \left\{ \begin{array}{c} \dots \\ x \neq^? \mathsf{f}(\mathsf{f}(w, \mathsf{f}(a, a)), z) \\ x \neq^? z \end{array} \right.$$

Here a is a name whereas w is variable, and we necessarily have that $W, i_w \vdash^? w$ (but also $Y, j_y \vdash^? y$) occurs in both $\mathcal{C}$ and $\mathcal{C}'$, and we have also that $i_w < i_y$. Now, on this branch, we have nothing to do regarding Step c, and we go back to Step b. We have still nothing to do regarding the second disequation of each constraint system, but we can apply the Cons *rule on the first one. Consider the left branch during such an application of* Cons(X, f)*, we get $(\mathcal{C}_{11}, \mathcal{C}'_{11})$ where:*

$$\mathcal{C}_{11} = \left\{ \begin{array}{c} \dots \\ x_1 \neq^? \mathsf{f}(\mathsf{f}(a, a), w) \vee x_2 \neq^? z \\ \mathsf{f}(x_1, x_2) \neq^? z \end{array} \right. \qquad \mathcal{C}'_{11} = \left\{ \begin{array}{c} \dots \\ x_1 \neq^? \mathsf{f}(w, \mathsf{f}(a, a)) \vee x_2 \neq^? z \\ \mathsf{f}(x_1, x_2) \neq^? z \end{array} \right.$$

Now, depending on the values of i_w and i_z, we may have a choice. We can either apply Cons *to simplify $x_1 \neq^? \mathsf{f}(\mathsf{f}(a, a), w)$ (and $x_1 \neq^? \mathsf{f}(w, \mathsf{f}(a, a))$) or apply* Eq-ded-ded *on $x_2 \neq^? z$. We consider here the first option (to respect maximality, this is only possible if $i_w \geq i_z$), and we get (on the left branch) the pair $(\mathcal{C}_{111}, \mathcal{C}'_{111})$ where:*

$$\mathcal{C}_{111} = \left\{ \begin{array}{c} \dots \\ x_{11} \neq^? \mathsf{f}(a, a) \vee x_{12} \neq^? w \vee x_2 \neq^? z \\ \mathsf{f}(\mathsf{f}(x_{11}, x_{12}), x_2) \neq^? z \end{array} \right. \quad \mathcal{C}'_{111} = \left\{ \begin{array}{c} \dots \\ x_{11} \neq^? w \vee x_{12} \neq^? \mathsf{f}(a, a) \vee x_2 \neq^? z \\ \mathsf{f}(\mathsf{f}(x_{11}, x_{12}), x_2) \neq^? z \end{array} \right.$$

Now, we may still have some choice, but it is not possible for instance to consider an application of the CONS *rule on* $x_{11} \neq \mathsf{f}(a, a)$. *Indeed,* $\mathcal{L}^1_{\mathcal{C}_{111}}(x_{11} \neq^? \mathsf{f}(a, a))$ *is not maximal since* $\mathcal{L}^1_{\mathcal{C}}(\mathsf{f}(a, a)) = 0$ *in any constraint system* $\mathcal{C}$. *This remark is important to avoid non termination. Indeed, applying* $\textsc{Cons}(X_{11}, \mathsf{f})$ *would allow us to add* $\mathsf{root}(X_{11}) \neq \mathsf{f}$ *on the constraint systems on the right branch, and then applying* EQ-DED-DED *on* $x_{12} \neq^? w$, *and then on* $x_2 \neq^? z$, *and considering the pair of constraint systems obtained along the right branch, we will eventually obtain a pair of constraint systems that will contain:*

$$\mathcal{C}_{\mathsf{right}} = \begin{cases} \dots \\ x_{12} \neq^? w \\ x_2 \neq^? z \\ \mathsf{f}(\mathsf{f}(x_{11}, x_{12}), x_2) \neq^? z \\ \mathsf{root}(X_{12}) \neq \mathsf{f} \end{cases} \quad \mathcal{C}'_{\mathsf{right}} = \begin{cases} \dots \\ x_{12} \neq^? w \\ x_2 \neq^? z \\ \mathsf{f}(\mathsf{f}(x_{11}, x_{12}), x_2) \neq^? z \\ \mathsf{root}(X_{12}) \neq \mathsf{f} \end{cases}$$

Assuming that x, y, *and* z *have the same support,* i.e., $X, i_x \vdash^? x$, $Y, i_y \vdash^? y$ *and* $Z, i_z \vdash^? z$ *are in* $\mathcal{C}$ *and* $\mathcal{C}'$ *with* $i_x = i_y = i_z$, *the situation is quite similar to the pair of constraint systems we consider at the very beginning of this example. We may apply a similar sequence of transformation rules leading to a termination issue. Note however that this sequence does not respect our maximality condition. The application of* $\textsc{Cons}(X_1, \mathsf{f})$ *on* $(\mathcal{C}_{11}, \mathcal{C}'_{11})$ *does not respect our maximality condition. Indeed, by definition of a constraint system, we have that* $i_w < i_y$, *and together with the hypothesis that* $i_x = i_y = i_z$, *this would contradicts the fact that* $i_w \geq i_z$.

Relying on this strategy, we are now able to prove termination of our algorithm.

Theorem 5. *(termination) Applying the transformation rules on a pair of sets of initial constraint systems and following the strategy* $\mathcal{S}$ *always terminates.*

5. Implementation

This decision procedure has been implemented in a tool called APTE. The tool is implemented in Ocaml (around 12 000 lines). APTE is an open source software and is distributed under GNU General Public Licence 3.0. The tool as well as the description of the protocols we have analysed using it are available at:

`http://projects.lsv.ens-cachan.fr/APTE/`.

As expected, APTE checks trace equivalence for processes that use standard primitives (*e.g.,* pairing, signatures, hash functions, symmetric and asymmetric encryptions). We can model in particular conditionals (with non-trivial else branches), private channels, and non-deterministic choices, but we consider processes without replication. In case of failure when establishing trace equivalence, APTE provides a witness of non-equivalence. In terms of protocols, APTE has been used to analyse several protocols among them the private authentication protocol, and some protocols issued from the e-passeport application as described in [33].

Our implementation closely follows the transformation rules that are described along the paper. However, for efficiency reasons, some optimisations have been implemented. In particular, the strategy described in the previous section imposes us to apply the rule

Axiom on each frame element and the rule Cons for each constructor symbol. Thus, giving the attacker some useless capabilities, *e.g.,* by adding some fresh names in the frame, or considering some additional hash functions, will considerably increase the execution time of our algorithm. To cope with these issues, we adapt the strategy: only the relevant instances of the rule Cons and Axiom are applied. We manage to obtain a quite efficient algorithm for checking symbolic equivalence between sets of constraint systems. However, the interleaving step, that is required for moving from symbolic equivalence to trace equivalence, is expensive from the computation point of view. Actually, we are faced with the usual interleaving explosion problem. For example, using such an approach, deciding anonymity for one session of the private authentication protocol amounts to solve 15 symbolic equivalences between pairs of sets of constraint systems (each pair containing between 2 and 8 constraint systems). Around 200 symbolic equivalence have to be solved to deal with 2 sessions, and more than 900 when we want to analyse 3 sessions of this protocol. Moreover, it is worth mentioning that the size of the constraint systems but also the number of constraint systems in each set are increasing, and we thus rapidly reach the limit of the tool.

To illustrate this exponential blow up, we report on Table 5 and Table 6 some experiments that we have performed to analyse the private authentication protocol for 1 and 2 sessions respectively. We indicate the number of symbolic traces of each length, and since a given symbolic trace may lead to several constraint systems, we also indicate this number (on the average). For instance, considering symbolic traces of length 6 (*i.e.,* traces made up of 6 input/output actions), there are 4 different symbolic traces of this length. On the average, one such trace leads to 6 constraint systems, and we will have to launch our algorithm 4 times for checking symbolic equivalence between pairs $(\mathcal{S}, \mathcal{S}')$ of sets of constraint systems (each pair containing 6 constraint systems in average). Checking one symbolic equivalence will require the applications of 428 transformations rules (most of the rules are applied internally) and this will be done in less than a few milliseconds.

	# traces	# systems	# rules	int. - ext.	time (s)
		(in average per trace)	*(in average per symbolic equivalence)*		
1	1	2	6	83% - 17%	0.00
2	1	2	14	86% - 14%	0.00
3	1	2	25	88% - 12%	0.00
4	2	2	41	90% - 10%	0.00
5	2	6	182	93% - 7%	0.00
6	4	6	428	94% - 6%	0.00
7	4	6	1734	95% - 5%	0.01

Figure 5: Results obtained when analysing 1 session of the private authentication protocol.

As indicated in Table 6, analysing two sessions of the private authentication protocol, we reach the limit of our tool. In particular, checking trace equivalence for traces up to length 9 requires us to launch our algorithm for checking symbolic equivalence between pairs of sets of constraint systems more that 90 times. The total time to get an answer is around 1500 seconds but it still remains around one hundred of symbolic equivalences to check and the underlying sets of constraint systems become very huge, *e.g.,* for traces of length 10, each pair contains 360 constraint systems in average.

	# traces	# systems *(in average per trace)*	# rules	int. - ext. *(in average per symbolic equivalence)*	time (s)
1	1	2	6	83% - 17%	0.00
2	1	2	14	86% - 14%	0.00
3	1	2	25	88% - 12%	0.00
4	2	4	41	90% - 10%	0.00
5	4	12	129	90% - 10%	0.00
6	6	41	793	93% - 7%	0.03
7	12	87	2468	94% - 6%	0.29
8	18	204	13324	96% - 4%	4.67
9	36	280	39292	96% - 4%	40.19
10	54	360	...	...	...
11	54	373	...	...	...

Figure 6: Results obtained when analysing 2 sessions of the private authentication protocol.

The results summarised in Table 5 and 6 correspond to what we obtain when enumerating all the symbolic traces and then applying our procedure for checking symbolic equivalence on each resulting symbolic trace. Since checking trace equivalence requires to consider partial symbolic traces, we managed to optimise the tool by exploiting the result of our algorithm launched on symbolic traces of size n when analysing the symbolic traces of size $n+1$. This avoids us to apply some transformations that have already been applied during the analysis of the traces of size n and push a bit the boundaries of our tool but an analysis of the private authentication protocol for 3 sessions is still out of reach.

6. Conclusion

Trace equivalence is a central notion for expressing privacy-type properties. It has been shown that trace equivalence can be reduced to checking equivalence between sets of constraint systems (see *e.g.,* [14]). This reduction result is very general and holds for arbitrary processes (without replication) and for arbitrary equational theories. In this paper, we present a procedure to automatically check equivalence between sets of constraint systems. Altogether, this gives us an algorithm for checking trace equivalence in the applied pi-calculus. The procedure described in this paper has been implemented and performed well on constraint systems. However, the interleaving step that is required for moving from symbolic equivalence to trace equivalence, is performed in a rather naive way and it appears that this step is expensive from the computation point of view.

To cope with the interleaving problem mentioned above, we would like to propose some optimisations to reduce the number of interleavings that have to be considered, and so the number of equivalence between sets of constraint systems that have to be checked. This problem has already been studied in the context of reachability properties [34] but seems to be more challenging for trace equivalence (see *e.g.,* for some preliminary results [35]).

Although our procedure is tailored to a fixed set of primitives, it can probably be extended to primitives that are described by subterm convergent rewriting systems. Similarly, adding more tests (such as equal encryption keys or equal lengths) should not be a problem. We would also like to enrich our algorithm to deal with less standard primitives such as blind signatures or trapdoor commitment functions that are crucial in the context of e-voting protocols but do not fall in any existing decidability results. Handling other associative-commutative primitives, such as exclusive-or, or modular exponentiationm, seems to be extremely challenging for the class of protocols that we consider: for the side constraints only (quantified disequalities) there is no quantifier elimination procedure.

Lastly, we would like also to pursue the study of the constraint systems that are generated by our algorithm. The matrices of constraint systems obtained at the end (*i.e.,* on the leaves) enjoy some nice properties (*e.g.,* existence of a "constructor" solution, one-to-one equivalence between constraint systems that occur on the same row, ...). We think that these properties can be further exploited to decide some more fine grained notion of equivalence. Actually, relying on these nice properties, it has already been shown that the notion of *length trace equivalence*, a notion of equivalence that takes into account the length of messages [22], is decidable (relying on the algorithm presented in this paper).

References

[1] V. Cortier, S. Kremer (Eds.), Formal Models and Techniques for Analyzing Security Protocols, Vol. 5 of Cryptology and Information Security Series, IOS Press, 2011.

[2] V. Cortier, S. Kremer, B. Warinschi, A survey of symbolic methods in computational analysis of cryptographic systems, Journal of Automated Reasoning 46 (3-4) (2010) 225–259.

[3] M. Abadi, C. Fournet, Mobile values, new names, and secure communication, in: Proc. 28th Symposium on Principles of Programming Languages (POPL'01), ACM Press, 2001, pp. 104–115.

[4] A. Armando, R. Carbone, L. Compagna, J. Cuéllar, M. L. Tobarra, Formal analysis of SAML 2.0 web browser single sign-on: breaking the SAML-based single sign-on for Google apps, in: Proc. 6th ACM Workshop on Formal Methods in Security Engineering (FMSE'08), ACM, 2008, pp. 1–10.

[5] P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, B. Roscoe, The Modelling and Analysis of Security Protocols, Addison Wesley, 2000.

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

[7] C. Cremers, The Scyther Tool: Verification, falsification, and analysis of security protocols, in: Proc. 20th International Conference on Computer Aided Verification (CAV'08), Vol. 5123/2008 of LNCS, Springer, 2008, pp. 414–418.

[8] A. Armando, et al., The AVISPA Tool for the automated validation of internet security protocols and applications, in: Proc. 17th International Conference on Computer Aided Verification (CAV'05), Vol. 3576 of LNCS, Springer, 2005, pp. 281–285.

[9] H. Hüttel, Deciding framed bisimulation, in: Proc. 4th International Workshop on Verification of Infinite State Systems INFINITY'02, 2002, pp. 1–20.

[10] B. Blanchet, M. Abadi, C. Fournet, Automated verification of selected equivalences for security protocols, Journal of Logic and Algebraic Programming 75 (1) (2008) 3–51.

[11] S. Meier, B. Schmidt, C. Cremers, D. Basin, The tamarin prover for the symbolic analysis of security protocols, in: Proc. International Conference on Computer Aided Verification (CAV'13), Springer, 2013, pp. 696–701.

[12] D. Basin, J. Dreier, R. Sasse, Automated symbolic proofs of observational equivalence, in: Proc. 22nd Conference on Computer and Communications Security (CCS'15), ACM, 2015, pp. 1144–1155.

[13] V. Cortier, S. Delaune, A method for proving observational equivalence, in: Proc. of 22nd Computer Security Foundations Symposium (CSF'09), IEEE Comp. Soc. Press, 2009, pp. 266–276.

[14] V. Cheval, V. Cortier, S. Delaune, Deciding equivalence-based properties using constraint solving, Theoretical Computer Science 492 (2013) 1–39.

[15] M. Baudet, Deciding security of protocols against off-line guessing attacks, in: Proc. 12th ACM Conference on Computer and Communications Security (CCS'05), ACM Press, 2005.

[16] Y. Chevalier, M. Rusinowitch, Decidability of symbolic equivalence of derivations, Journal of Automated Reasoning.

[17] A. Tiu, J. E. Dawson, Automating open bisimulation checking for the spi calculus, in: Proc. 23rd IEEE Computer Security Foundations Symposium (CSF'10), IEEE Computer Society Press, 2010, pp. 307–321.

[18] R. Chadha, Ş. Ciobâcă, S. Kremer, Automated verification of equivalence properties of cryptographic protocols, in: Proc. 21th European Symposium on Programming (ESOP'12), Vol. 7211 of LNCS, Springer, 2012, pp. 108–127.

[19] M. Arapinis, T. Chothia, E. Ritter, M. Ryan, Analysing unlinkability and anonymity using the applied pi calculus, in: Proc. 23rd IEEE Computer Security Foundations Symposium (CSF'10), IEEE Computer Society Press, 2010, pp. 107–121.

[20] V. Cheval, B. Blanchet, Proving more observational equivalences with Proverif, in: Proc. 2nd International Conference on Principles of Security and Trust (POST'13), LNCS, Springer, 2013, pp. 226–246.

[21] M. Abadi, C. Fournet, Private authentication, Theoretical Computer Science 322 (3) (2004) 427–476.

[22] V. Cheval, V. Cortier, A. Plet, Lengths may break privacy – or how to check for equivalences with length, in: Proc. 25th International Conference on Computer Aided Verification (CAV'13), Vol. 8043 of LNCS, Springer, 2013, pp. 708–723.

[23] G. Bana, H. Comon-Lundh, A computationally complete symbolic attacker for equivalence properties, in: Proc. ACM Conference on Computers and Communications Security, 2014.

[24] J. Millen, V. Shmatikov., Constraint solving for bounded-process cryptographic protocol analysis, in: Proc. of 8th ACM Conference on Computer and Communications Security, 2001.

[25] H. Comon-Lundh, V. Cortier, E. Zalinescu, Deciding security properties of cryptographic protocols. application to key cycles., Transaction on Computational Logic 11 (2).

[26] H. Comon-Lundh, S. Delaune, J. Millen, Constraint solving techniques and enriching the model with equational theories, in: V. Cortier, S. Kremer (Eds.), Formal Models and Techniques for Analyzing Security Protocols, Vol. 5 of Cryptology and Information Security Series, IOS Press, 2011, pp. 35–61.

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

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

[29] H. Comon-Lundh, S. Delaune, The finite variant property: How to get rid of some algebraic properties, in: Proc. 16th International Conference on Rewriting Techniques and Applications (RTA'05), Vol. 3467 of LNCS, 2005.

[30] N. Dershowitz, J.-P. Jouannaud, Rewrite systems, in: Handbook of Theoretical Computer Science, Vol. B, Elsevier, 1990, Ch. 6.

[31] H. Comon, P. Lescanne, Equational problems and disunification, Journal of Symbolic Computation 7 (3/4) (1989) 371–425.

[32] V. Cheval, H. Comon-Lundh, S. Delaune, Automating security analysis: symbolic equivalence of constraint systems, in: Proc. 5th International Joint Conference on Automated Reasoning (IJCAR'10), Vol. 6173 of LNAI, Springer-Verlag, 2010, pp. 412–426.

[33] V. Cheval, Apte: an algorithm for proving trace equivalence, in: Proc. 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14), LNCS, Springer, 2014.

[34] D. Basin, S. Mödersheim, L. Viganò, Constraint Differentiation: A New Reduction Technique for Constraint-Based Analysis of Security Protocols, in: Proc. 10th ACM Conference on Computer and Communications Security (CCS'03), ACM Press, New York, 2003, pp. 335–344.

[35] D. Baelde, S. Delaune, L. Hirschi, A reduced semantics for deciding trace equivalence using constraint systems, in: Proc. 3rd Conference on Principles of Security and Trust POST'14, LNCS, Springer, 2014.